Dominion Energy®

Automated Generator Model Calibration with PredictiveGrid

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

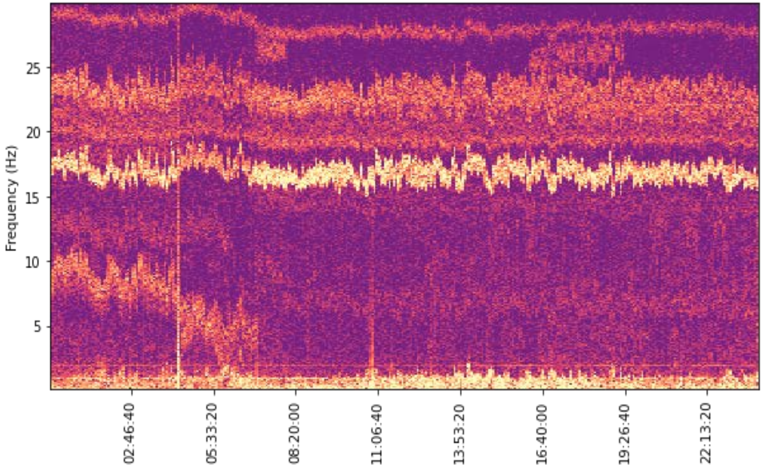
Dominion Energy

Project Overview

- Joint effort of **Dominion Energy Electric Transmission** with **Rensselaer Polytechnic Institute (RPI).**
- 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

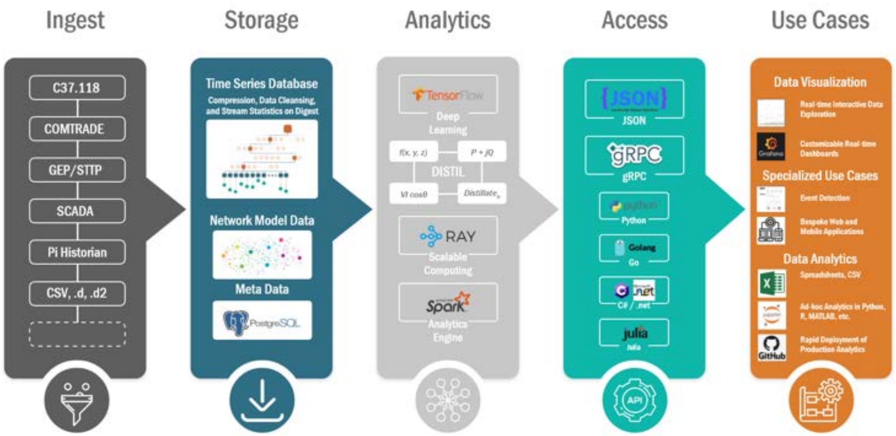
- **Dominion uses the same models used for planning and control design**
- **Modeling challenges**
  - Conventional model validation require events happening but system mostly in ambient conditions.
  - Operation conditions change throughout the day due to changing nature of load, line switching, V setpoint 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.

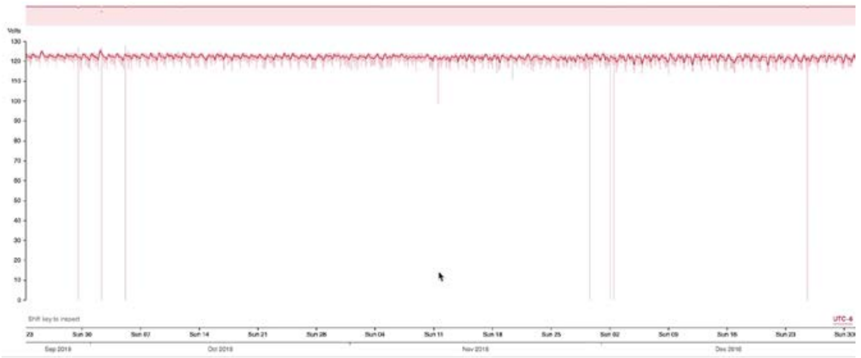


Voltage Magnitude Spectrogram at Unmodeled Generating Unit

Dominion Energy

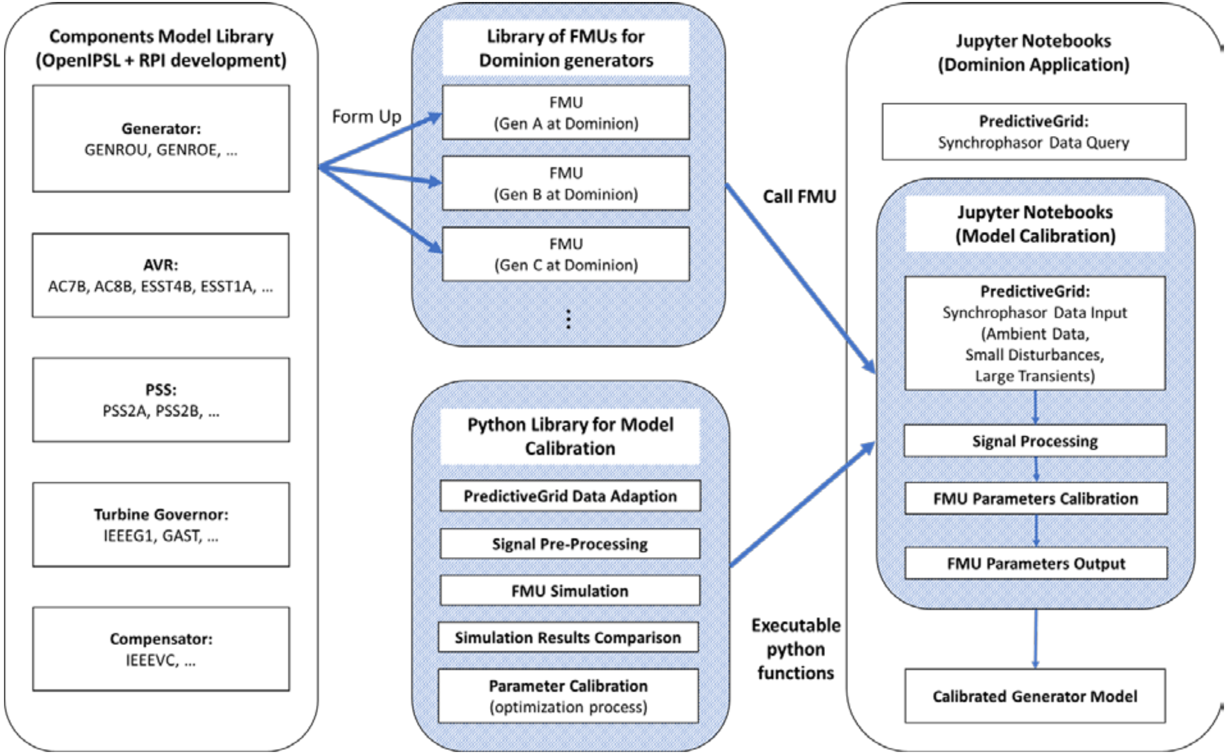
PredictiveGrid





Dominion Energy

Envisioned Toolchain (Design)



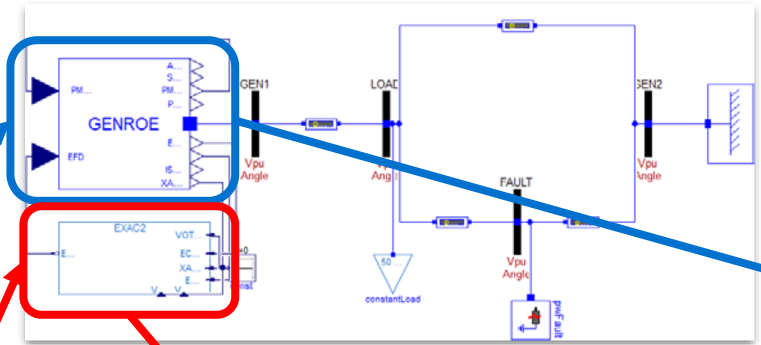
The Modelica Language and the OpenIPSL Library for Power System Modeling and Simulation

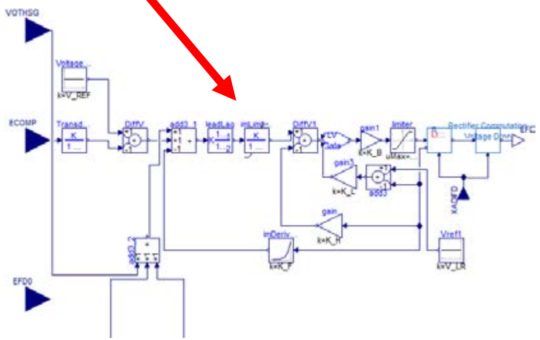
- Non-proprietary, object-oriented, equation-based ***modeling language*** for cyber physical systems .
- Open access (no paywall) & standardized language specification (link), maintained by the Modelica Association
- Open source Modelica Standard Library with more than 1,600 components models.
- *Supported 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

- Machines
  - PSAT
  - PSSE
    - GENSAL
    - GENROU
    - GENROE
- Controls
  - PSAT
  - Simulink
  - PSSE
    - OEL
    - ES
      - ESST4B
      - EXAC2





equation

//Interfacing outputs with the internal variables

$$
\begin{aligned}
\mathrm{XADIFD} &= \mathrm{XadIfd}\\
\mathrm{ISORCE} &= \mathrm{XadIfd}\\
\mathrm{EFD0} &= \mathrm{efd0}\\
\mathrm{PMECH0} &= \mathrm{pm0}\\
\frac{d\,\mathrm{Epq}}{d\,t} &= \frac{1}{\mathrm{Tpd0}}\left(\mathrm{EFD}-\mathrm{XadIfd}\right)\\
\frac{d\,\mathrm{Epd}}{d\,t} &= \frac{1}{\mathrm{Tpq0}}\left(-1\right)\mathrm{XaqIlq}\\
\frac{d\,\mathrm{PSIkd}}{d\,t} &= \frac{1}{\mathrm{Tppd0}}\left(\mathrm{Epq}-\mathrm{PSIkd}-\left(\mathrm{Xpd}-\mathrm{Xl}\right)\mathrm{id}\right)\\
\frac{d\,\mathrm{PSIkq}}{d\,t} &= \frac{1}{\mathrm{Tppq0}}\left(\mathrm{Epd}-\mathrm{PSIkq}+\left(\mathrm{Xpq}-\mathrm{Xl}\right)\mathrm{iq}\right)\\
\mathrm{Te} &= \mathrm{PSId}\cdot\mathrm{iq}-\mathrm{PSIq}\cdot\mathrm{id}\\
\mathrm{PSId} &= \mathrm{PSIppd}-\mathrm{Xppd}\cdot\mathrm{id}\\
\mathrm{PSIq} &= \left(-\mathrm{PSIppq}\right)-\mathrm{Xppq}\cdot\mathrm{iq}\\
\mathrm{PSIppd} &= \mathrm{Epq}\cdot\mathrm{K3d}+\mathrm{PSIkd}\cdot\mathrm{K4d}\\
-\mathrm{PSIppq} &= \left(-\mathrm{Epd}\cdot\mathrm{K3q}\right)-\mathrm{PSIkq}\cdot\mathrm{K4q}\\
\mathrm{PSIpp} &= \sqrt{\mathrm{PSIppd}\cdot\mathrm{PSIppd}+\mathrm{PSIppq}\cdot\mathrm{PSIppq}}\\
\mathrm{XadIfd} &= \mathrm{K1d}\cdot\left(\mathrm{Epq}-\mathrm{PSIkd}-\left(\mathrm{Xpd}-\mathrm{Xl}\right)\mathrm{id}\right)+\mathrm{Epq}\\
\mathrm{XaqIlq} &= \mathrm{K1q}\cdot\left(\mathrm{Epd}-\mathrm{PSIkq}+\left(\mathrm{Xpq}-\mathrm{Xl}\right)\mathrm{iq}\right)+\mathrm{Epd}
\end{aligned}
$$

//change sign for PSIppq 3/3

$$
\begin{aligned}
\mathrm{ud} &= \left(-\mathrm{PSIq}\right)-R_a\cdot\mathrm{id}\\
\mathrm{uq} &= \mathrm{PSId}-R_a\cdot\mathrm{iq}
\end{aligned}
$$

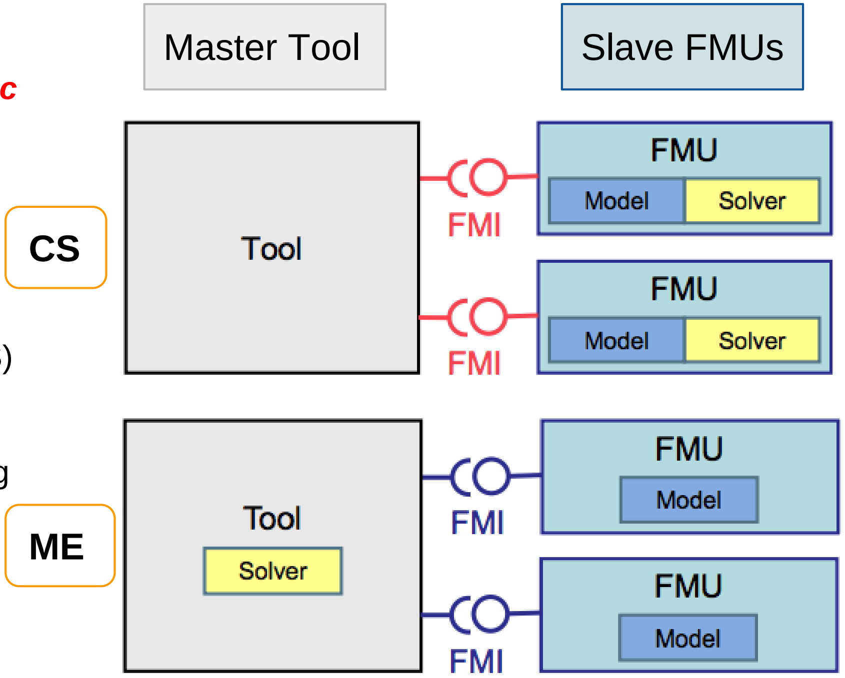
//flow

end GENROE

$$
\begin{cases}
+\mathrm{id}\cdot\left(\mathrm{Xd}-\mathrm{Xpd}\right)+\mathrm{SE\_exp}\left(\mathrm{PSIpp},\mathrm{S10},\mathrm{S12},1,1.2\right)\mathrm{PSIppd}\\
-\mathrm{iq}\cdot\left(\mathrm{Xq}-\mathrm{Xpq}\right)-\dfrac{\mathrm{SE\_exp}\left(\mathrm{PSIpp},\mathrm{S10},\mathrm{S12},1,1.2\right)\left(-1\right)\mathrm{PSIppq}\cdot\left(\mathrm{Xq}-\mathrm{Xl}\right)}{\mathrm{Xd}-\mathrm{Xl}}
\end{cases}
$$

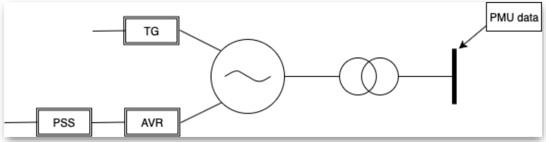
The Functional Mockup Interface Standard

- 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)

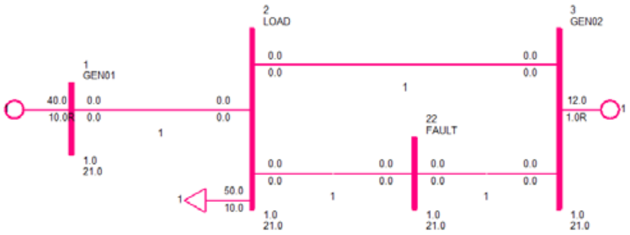
Dominion Energy

Models for Software-to-Software Verification

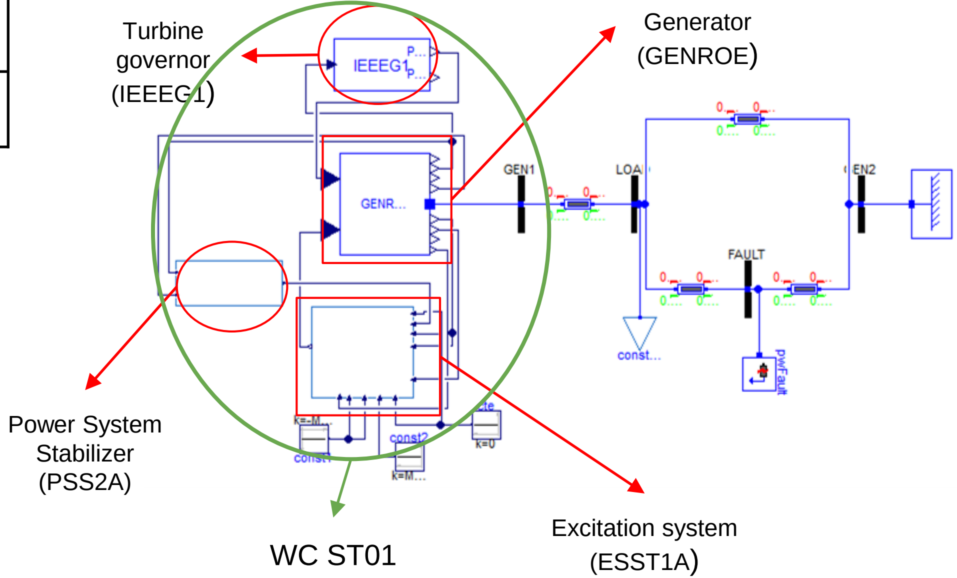
Plant configuration of the reference PSS@E model

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

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



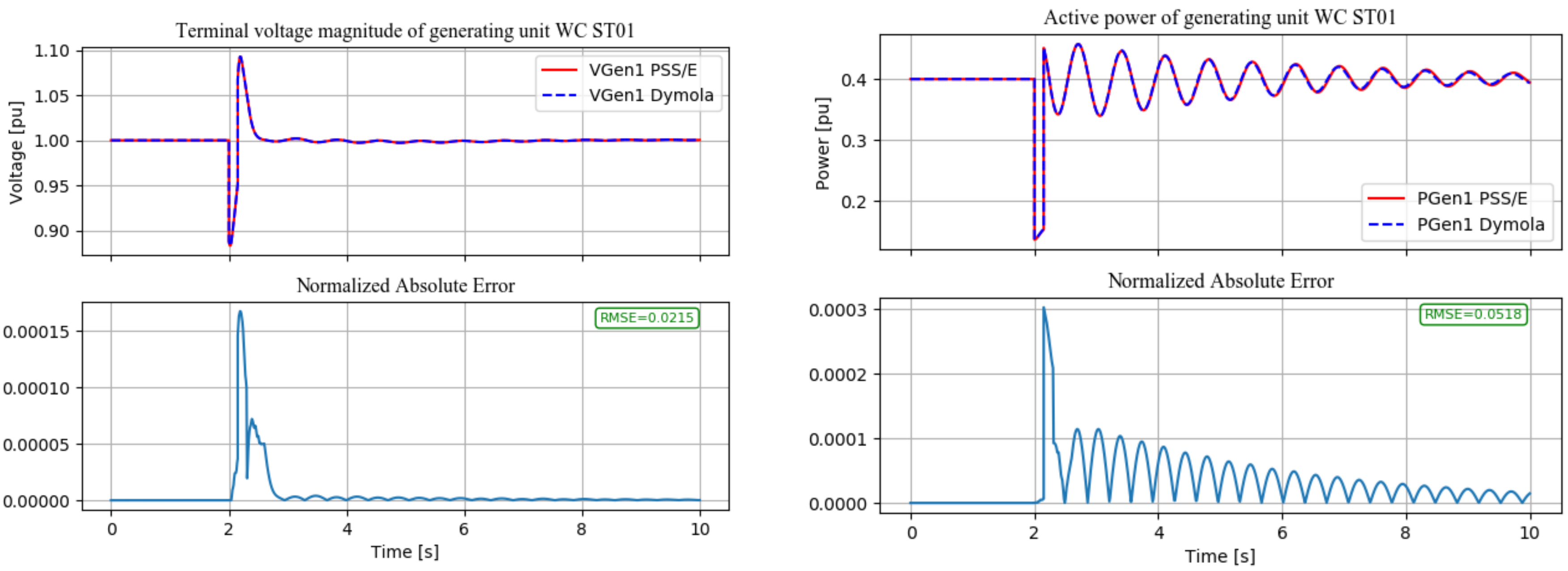
Modelica Implementation using the OpenIPSL Library



Dominion Energy

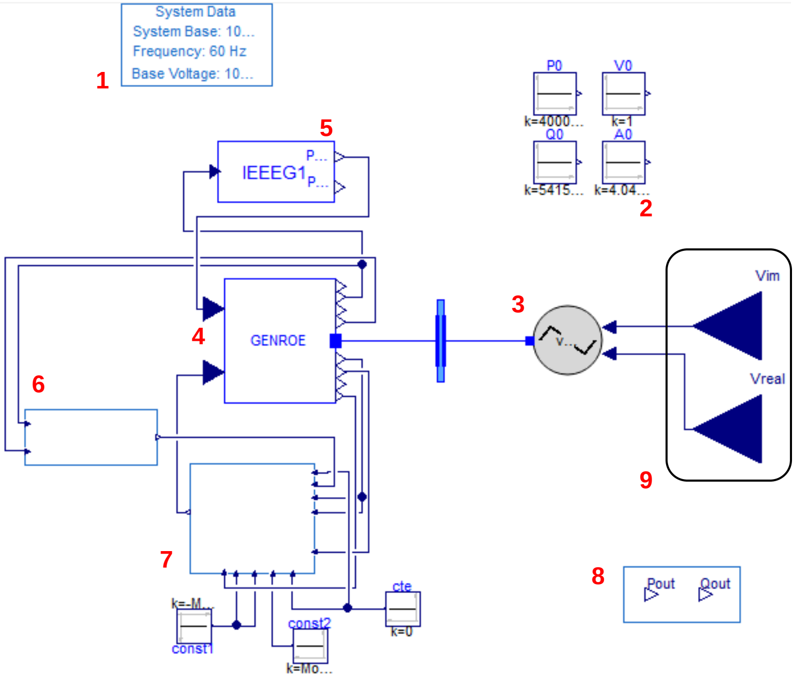
Verification: Modelica (Dymola SW) vs PSS/E

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 passess 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

Sub-station Name and Voltage Level

```
# Determining data:
sub_line_list = [['   kV','VPHM','A',0],
                 ['   kV','VPHM','B',0],
                 ['   kV','VPHM','C',0],
                 ['   kV','VPHA','A',0],
                 ['   kV','VPHA','B',0],
                 ['   kV','VPHA','C',0],
                 ['   kV Delta','IPHM','A',0],
                 ['   kV Delta','IPHM','B',0],
                 ['   kV Delta','IPHM','C',0],
                 ['   kV Delta Ia','IPHA','A',0],
                 ['   kV Delta Ib','IPHA','B',0],
                 ['   kV Delta Ic','IPHA','C',0]]

nline = len(sub_line_list)
# Got all streams
All_Streams = getstreams_DFR(conn,[sub_line_list[ii][0] for ii in range(nline)],
                              [sub_line_list[ii][2] for ii in range(nline)],
                              [sub_line_list[ii][3] for ii in range(nline)],
                              [sub_line_list[ii][1] for ii in range(nline)])
All_Streams = [All_Streams[i][sub_line_list[i][4]] for i in range(nline)]
basevals = get_base(conn,All_Streams)
# Time window
T_window = 1*60 # window size in seconds
tstart = datetime(2020, 8, 26, 20, 58, 0, 0).timestamp()*1e9
trange = np.array([tstart,tstart+T_window*1e9]) # time window
fs = 30.0 # sampling frequency
# Get data
fdatamat_pre,tdata = ExtractData_resample_2(conn, All_Streams, '', trange[0], trange[1], 1/fs, basevals)
```

```
def pre_process_2(datamat,tdat,fs,f_filter):
    mean = [np.mean(datamat[ii])*np.ones(np.shape(datamat[ii])) for ii in range(len(datamat))]
    # Pre-Process
    datamat_process = [(np.array(datamat[ii])-np.mean(datamat[ii])).tolist() for ii in range(len(datamat))]
    datamat_process = butter_filter(datamat_process,'high',f_filter[0],fs) # detrend
    datamat_process = butter_filter(datamat_process,'low',f_filter[1],fs) # denoise
    # add mean again
    datamat_process = (np.array(datamat_process)+mean).tolist()
    if f_filter[1] < fs/2:
        # downsample
        fs_re = 2*f_filter[1]
        tdat_re = np.arange(tdat[0],tdat[-1],1e9/fs_re)# down sample
        datamat_process = [resample_data(datamat_process[i],tdat,tdat_re) for i in range(len(datamat_process))]
    else:
        tdat_re = tdat
        fs_re = fs
    return datamat_process,tdat_re,fs_re
#--- Filter data:
f_filter = [0.01,15]
fdatamat,tdata_re,fs_re = pre_process_2(fdatamat_pre,tdata,fs,f_filter)
```

Model and Toolchain Integration

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

```python
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

```python
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

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

Defining inputs/outputs after signal processing

```python
# 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

```python
# 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)}
```

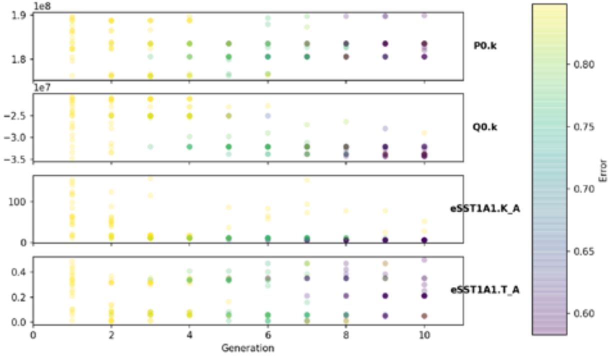
Defining estimation algorithms and settings

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

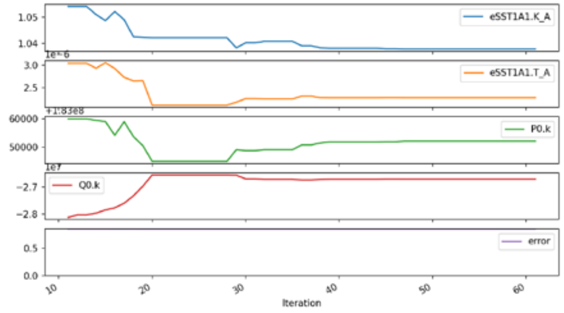
Testing: Parameter Estimation Under Ambient Conditions

- 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

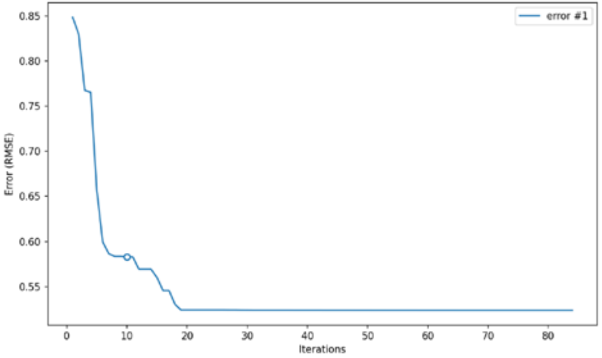


Nelder-Mead



estimation elapsed time ≈ 1431s

Error



GA Nelder-Mead

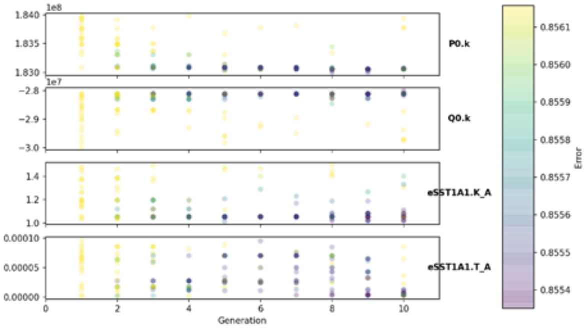
Sequence of algorithms used for the estimation

Dominion Energy

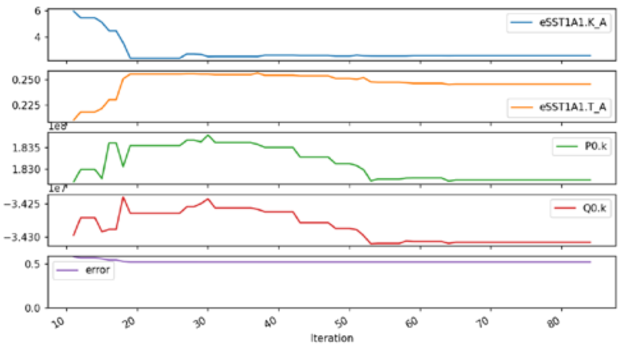
Testing: Parameter Estimation Under a Transient

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

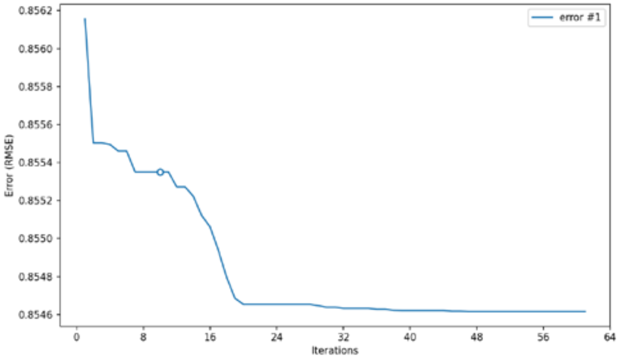
GA algorithm



Nelder-Mead



Error



GA

Nelder-Mead

estimation elapsed time ≈ 447s

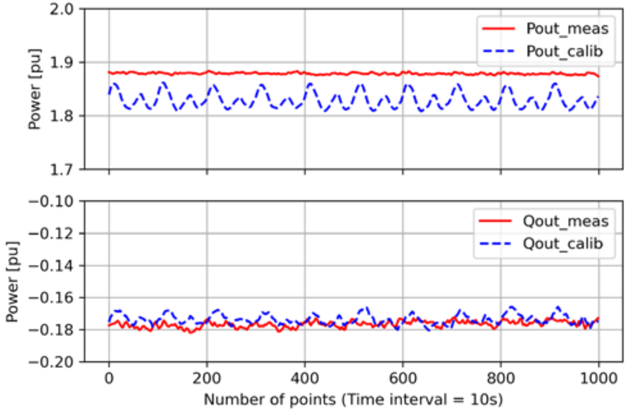
Sequence of algorithms used for the estimation

Dominion Energy

Proof-of-Concept: 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.
- 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.

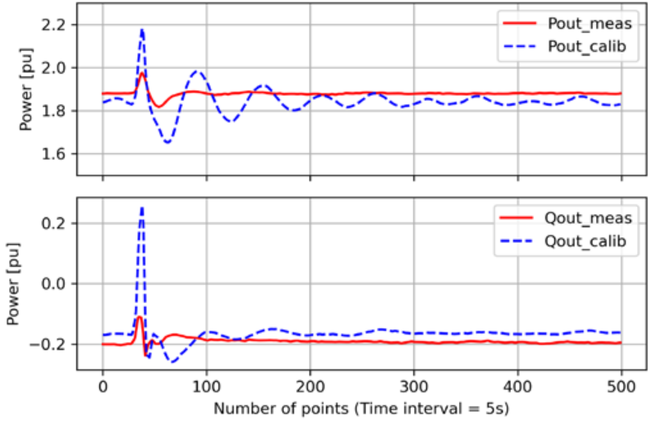
Calibration for Ambient




```
estimates
```

```
{'eSST1A1.K_A': 2.6426506177827225,
 'eSST1A1.T_A': 0.24553453016003393,
 'P0.k': 182750836.3976619,
 'Q0.k': -34307843.08675239}
```

Calibration for Transient Event




```
estimates
```

```
{'eSST1A1.K_A': 1.0379577400856557,
 'eSST1A1.T_A': 2.2805494426998525e-06,
 'P0.k': 183052047.85619536,
 'Q0.k': -26700844.37074689}
```

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.

Dominion Energy

Thank you!