Long-Term Dynamic Simulation of Power Systems using Python, Agent Based Modeling, and Time-Sequenced Power Flows

–Software and Simulated Controls Documentation– ver. 0.4

by

Thad Haines

Montana Technological University

2020

Introduction

PSLTDSim is a power system long-term dynamic simulator that is based on time-sequenced power flows and models additional dynamics such as system frequency and turbine speed governor action. Software development is currently ongoing and represents the crux of a work in progress masters thesis.

This document was compiled from various software explanation sections lifted from said thesis to offer a kind of 'lazy user guide'. Due to the current in-progress nature of thesis writing, sections may be incomplete, vague, janky, or written in a drafty, wandering fashion. Full working codes, which may more clearly illustrate software use, are located on the associated github page at `https://github.com/thadhaines/PSLTDSim/tree/master/testCases` while a .bat file that actually runs the simulation tool (`runPSLTDsim.bat`) is located a folder higher. The very similarly named `runPLTD.bat` creates plots from saved system mirrors.

While efforts have been made to produce a logically written functional software package, reality can often reduce such efforts to merely good intentions. However, some may say that intentions matter.

Questions and/or comments may be sent to jhaines at mtech dot edu.

Table of Contents

List of Tables

List of Figures

List of Equations

Glossary of Terms

Term	Definition
ABM	Agent Based Modeling
ABS	Agent Based Simulation
AC	Alternating Current
ACE	Area Control Error
AGC	Automatic Generation Control
AMQP	Advanced Message Queue Protocol
API	Application Programming Interface
BA	Balancing Authority
BES	Bulk Electrical System
CLR	Common Language Runtime
CTS	Classical Transient Stability
DACE	Distributed ACE
DTC	Definite Time Controller
EIA	United States Energy Information Administration
ERCOT	Electric Reliability Council of Texas
FERC	Federal Energy Regulatory Commission
FTL	Frequency Trigger Limit
GE	General Electric
Hz	Hertz, cycles per second
IACE	Integral of ACE
IPC	Interprocess Communication
IPY	IronPython
ISO	Independent Service Operator
J	Joule, Neton meters, Watt seconds
LFC	Load Frequency Control
LTD	Long-Term Dynamic
NERC	North American Electric Reliability Corporation
ODE	Ordinary Differential Equation

Term	Definition
P	Real Power
PI	Proportional and Integral
PMIO	PSLF Model Information Object
PSDS	PSLF Dynamic subsystem.
PSLF	Positive Sequence Load Flow
PSLTDSim	Power System Long-Term Dynamic Simulator
PSS	Power System Stabilizer
PST	Power System Toolbox
PU	Per Unit
PY3	Python version 3.x
PyPI	Python Package Index
Q	Reactive power
RACE	Reported ACE
RTO	Regional Transmission Organization
SACE	Smoothed ACE
SI	International System of Units
TLB	Tie-Line Bias
TSPF	Time-Sequenced Power Flow
US	United States of America
VAR	Volt Amps Reactive
W	Watt, Joules per second
WECC	Western Electricity Coordinating Council

1 Software Background

1.1 Classical Transient Stability Simulation

Classical transient stability (CTS) simulation is a simulation approach commonly used to test a power system's ability to remain stable after a large step perturbance such as a generator trip. The time frame CTS focuses on is milliseconds to tens of seconds, and as such, requires time steps of milliseconds. While this is an appropriate approach for relatively short periods of time, complicated modeling issues may lead to questionable results over the course of longer simulations.

General Electric's (GE's) Positive Sequence Load Flow (PSLF) is an industry standard power-flow solver and CTS simulation program. The continued development of PSLF has produced a large library of dynamic models and components for use in CTS simulation. PSLFs dynamic library has enabled a wide variety of system models to be created. For example, multiple full WECC base cases have been modeled in PSLF over the past 20 years and are continuously being updated. Certain versions of PSLF offer a Python 3 application programming interface (API) and a .NET API. Despite each API being at various levels of development and functionality, both offer a modern way to communicate with PSLF.

1.2 Python

Python is an interpreted high-level general purpose programming language that utilizes object oriented techniques and emphasizes code readability [34]. Guido Van Rossum first implemented a version of Python in December of 1989 [55]. Python is freely available and distributable for multiple computing platforms [52].

1.2.1 Python Packages

Software modules that expand the functionality of Python are referred to as packages and are freely available from the Python Package Index (PyPI). This work utilizes multiple packages to varying degrees, though SciPy and Pika are among the most heavily used. SciPy

is a collection of packages that include NumPy for numerical computing and Matplotlib for MATLAB style plotting [58]. Additionally, scipy.signal.lsim was used to solve state-space equations that are common in electrical engineering dynamics. To avoid rewriting well known integration routines, scipy.integrate.solve_ivp was used to perform Runge-Kutta integration of the swing equation to find system frequency. The Python implementation of the advanced message queuing protocol (AMQP) used for this project was Pika [20].

1.2.2 Varieties of Python

Python has gone through numerous versions over the course of development and has been ported and adapted according to need. IronPython (IPY) is an open-source .NET compatible version of Python written in C# [1] and is able to use the common language runtime (CLR) package required for properly interacting with the GE PSLF .NET library. As such IPY, more specifically 32-bit IPY, is used to interface with the PSLF .NET library. However, the most current stable IPY release is based off of Python 2.x and can only use Python packages compatible with 2.x. Python 3 (PY3) is 'the future of Python' and has many more maintained and useful community created packages. Some packages area available for both Python version 2 and 3, but many are not. As of this writing, the current version of Python is 3.8. A majority of project code was written using Python 3.7, but should continue to be compatible with future versions of PY3.

1.2.3 Python Specific Data Types

Python has various common data types found in other programing languages such as integer, string, list, and float. Python also has some unique data types. One unique data type is called a *dictionary* which is a collection of key value pairs. Dictionary keys are strings, and the value can be any other data type, including a dictionary. A benefit of using a dictionary is that it doesn't matter 'where' in an object a certain data point is located, only that is exists somewhere in the object. The key value pair eliminates the need to focus on indexing lists or arrays as other programming languages may require. Additionally, native python methods allow for simple searching and iteration of dictionaries.

Another somewhat unique python data type is a tuple. Tuples are essentially the same as lists, except defined using parenthesis instead of brackets and the data inside can not be changed in any way. While this may seem like a hindrance, it creates peace of mind when using tuples for important data references.

A powerful characteristic of Python is that everything is an object, and variables act as references, or pointers, to data. These Pythonic points were relied upon heavily during software development to make large collections of objects and references.

1.3 Advanced Message Queueing Protocol

Advanced message queueing protocol (AMQP) is a software messaging protocol that can be used for interprocess communication (IPC). The specific application of AMQP in this case was to enable a PY3 process to communicate with an IPY process on a Windows based machine. The idea behind AMQP is that of a virtual 'broker' which receives messages from various processes and places them into specific named queues. The named queues are accessed by other processes that check for, and receive, any queued messages.

It should be noted that Erlang was required to allow use of RabbitMQ, and the PY3 and IPY Pika packages were required so that Python could use AMQP. While detailed descriptions of these softwares is beyond the scope of this research: Erlang is an open source programming language and runtime environment, RabbitMQ is an open source AMQP broker software, and Pika is a Python package that works with RabbitMQ. The correct installation of these software packages is necessary for the created research software to function.

1.4 Agent Based Modeling

Agent Based Modeling (ABM), or Agent Based Simulation (ABS), is oriented around the idea that any situation can be described by agents in an environment, and a definition of agent-to-agent and agent-to-environment interactions [53]. The ABM coding style lends itself towards modular coding and natural expandability as each agent class may have inherited

methods and variable characteristics. As an example, [3] used an ABM approach to test the efficiency of differing airplane boarding methods. Passengers were treated as agents interacting with each other and the airplane environment. Each passenger agent may have had different characteristics, but performed a similar actions each simulation time step. ABM is applied to the coding of this project in that a power system acts as the environment, and all power system objects, such as buses, loads, and generators, are treated as agents.

Each time step, agents may perform their *step* method that executes code unique to the specific agent, but generally the same among agent types. For example, a timer agent step method may include checking a specified value and incrementing an accumulator according to a logic operation. If the accumulator becomes larger than a set threshold value, the timer may raise an activation flag. Agent action is meant to be direct and generic so that agents can be reused and nested inside other agents. Continuing the example, the timer agent may be nested inside another agent that checks the timers activation flags each step and takes action depending on the returned status. Actions can be arranged into a sequence of agent steps in a discrete time simulation, and then repeated ad infinitum. The idea of sequencing agent steps can be applied to systems of nearly any size and composition as long as agents have a step function and can reference other agents inside the environment.

2 Software Tool

2.1 Time-Sequenced Power Flows

The left side of Figure 2.1 is a visual representation of a single power-flow solution. The power-flow solution can be thought of as a snapshot of steady state system operation. A period of steady state system operation could be imagined by repeating this single power-flow solution in a time sequence. Dynamic system behavior could be realized if small changes are made to the power-flow problem inputs and the ensuing power-flow solution converges. The right side of Figure 2.1 illustrates the idea of time-sequenced power flow (TSPF) as a collection of slightly changing power-flow solutions. The TSPF method for dynamic simulation involves performing dynamic modeling calculations outside of, or inbetween, each power-flow solution.

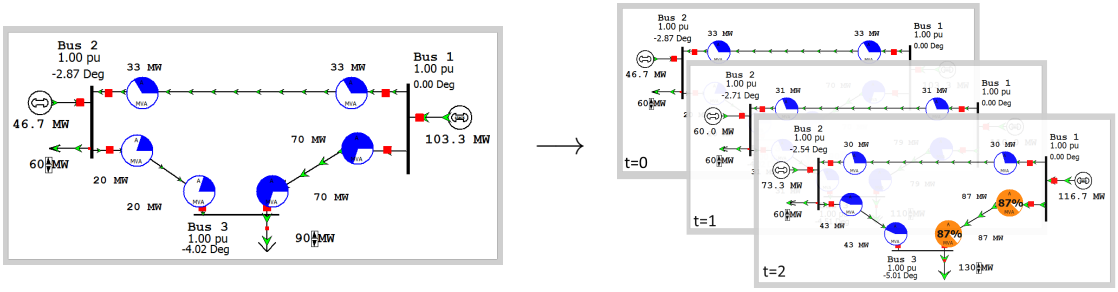

Figure 2.1: Time-sequenced power flows visualized.

Differences between TSPF and CTS simulation techniques stem from the time frame of simulation focus and differing dynamic calculations. TSPF is focused on long-term events that take place over the course of minutes to hours. CTS simulation focuses on events that are tens of seconds in duration. This difference in focus leads to each method utilizing a different appropriate time step. A TSPF time step may be 0.5 to 1 seconds while CTS uses sub-cycle time-steps in the millisecond range.

Calculations of CTS simulation include generator dynamics that are aggregated and simplified in TSPF. Each method starts with a power-flow solution, but CTS simulation performs back calculations to set initial states of various dynamic models. Future CTS

states, and resulting system behaviors, are dictated by dynamic model interactions which may go unstable. TSPF also uses dynamic models, but updated values are sent to a power-flow solver; and the power-flow solution provides new steady state system values. These differences create output data that is of different resolutions and captures slightly different system characteristics. Figure 2.2 shows a comparison of CTS and TSPF data.

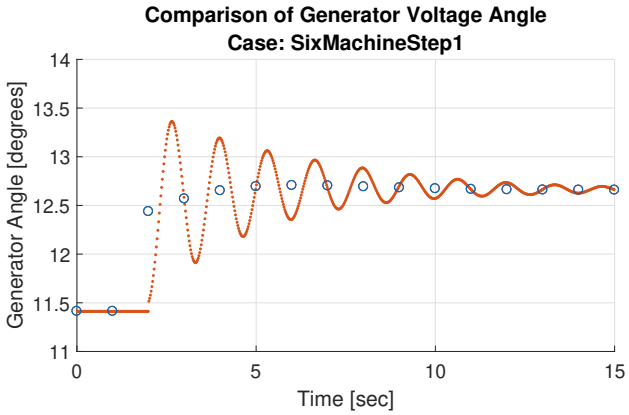

Figure 2.2: CTS and TSPF data output comparison.

The circles represent data from a TSPF simulation while dots show down-sampled data from PSLFs dynamic subsystem (PSDS) simulation. The two data sets are not the same during oscillatory behavior, but seem to match at steady state conditions. In the provided plot, TSPF data seems to follow the center of oscillation from the CTS simulation. This 'oscillation averaging' behavior was found to be common in TSPF results.

2.2 Simulation Assumptions and Simplifications

Numerous assumptions and simplifications were made due to fundamental differences between TSPF and CTS simulation. This section details such assumptions and simplifications and provides key TSPF equations.

2.2.1 General Assumptions and Simplifications

The focus of PSLTDSim is on the long-term operation of power systems. Since a power-flow solution is a stable steady state operating condition, it is assumed that the system of study is stable, and remains stable, for the entirety of the simulation. Other

software packages should be used if transient stability is in question.

In general, system responses to large step type contingencies are not the focus of PSLTDSim. Instead, a more suitable event type is in the form of ramps, or repeated small step perturbances. Because of the more 'gentle' system scenarios involved with ramps, power system stabilizers (PSS) are not modeled. It is assumed that the system will not be disturbed enough to require meaningful PSS action.

Further assumptions include ideal generator exciters that can always maintain a given generator reference voltage. Modern exciters are assumed to be fast enough to hold reference voltages to small perturbances in the long-term. Despite the ideal reference voltage assumption, machine VAR limits **are** enforced according to model parameters. Should the need arise, future work can be done to incorporate any additional, or alternative modeling.

2.2.2 Time Step Assumptions and Simplifications

Multiple assumptions can be made concerning power system behavior when a time step of one second is used. The order of magnitude time step difference between CTS and TSPF allows models created for CTS simulations to be simplified for use in TSPF. Intermachine oscillations were ignored since subsynchronous resonances are sub-second and the time resolution of TSPF is not great enough to capture these phenomena. Additionally, in the long-term, these effects are minor when the system is stable. Without the need for subsynchronous characteristics, generator modeling was greatly simplified. The only details required for a machine model are MW cap, machine MVA base, and machine inertia.

To further simplify generator modeling, all machines share a single system frequency that is calculated by a combined swing equation. The following two Sections explain how the combined swing equation is used in PSLTDSim. Governor models were also simplified. Section 2.2.5 explains the how PSLTDSim models governors.

2.2.3 Combined System Frequency

Instead of a frequency being calculated for each generator or bus, a single combined swing equation is used to model only one combined system frequency. This technique requires

a known total system inertia H_{sys} and the total system acceleration power $P_{acc,sys}$. In a system win N generators, H_{sys} is calculated from each individual machines inertia as

$$H_{sys} = \sum_{i=1}^{N} H_{PU,i} M_{base,i}. \tag{2.1}$$

In a system with N generators, total system accelerating power is calculated as

$$P_{acc,sys} = \sum_{i=1}^{N} P_{m,i} - \sum_{i=1}^{N} P_{e,i} - \sum \Delta P_{pert}, \tag{2.2}$$

where $P_{m,i}$ is mechanical power and $P_{e,i}$ is electrical power of generator i and any system power injections, or perturbances, are accounted for in the $\sum \Delta P_{pert}$ term.

The combined swing equation, shown in Equation 2.3, uses $P_{acc,sys}$ and H_{sys} to calculate $\dot{\omega}_{sys}$. After integration, $\dot{\omega}_{sys}$ leads to the single combined system frequency ω_{sys}.

$$\dot{\omega}_{sys} = \frac{1}{2H_{sys}} \left(\frac{P_{acc,sys}}{\omega_{sys}} - D_{sys} \Delta\omega_{sys} \right) \tag{2.3}$$

For completeness, a damping term $D_{sys}\Delta\omega$ is included in Equation 2.3, but as Equation **??**, D_{sys} is often set to zero while $\Delta\omega_{sys}$ is calculated using Equation **??** with ω_{sys} replacing ω.

2.2.4 Distribution of Accelerating Power

While system frequency can be calculated using total system accelerating power, to properly 'seed' the next power flow, each generator participating in the system inertial response must account for a portion of accelerating power absorption. The specific amount each generator absorbs is based on machine inertia. Equation 2.4 shows how the next electric power estimate $P_{e,EST,i}$ is created for generator i.

$$P_{e,EST,i} = P_{e,i} - P_{acc,sys} \left(\frac{H_i}{H_{sys}} \right) \tag{2.4}$$

Once all accelerating power is distributed to inertial responding generators, the new $P_{e,EST}$ value for each generator is used to seed a power flow. If the MW difference between

resulting power supplied by the slack generator and estimated power output is larger than the set slack tolerance, the difference is redistributed as $P_{acc,sys}$ according to Equation 2.4 until slack tolerance is met, or a maximum number of iterations take place. Once the slack tolerance is met, the power-flow solution is accepted as the current state of the system under study.

2.2.5 Governor models

Long-term dynamic models do not require the detail of a full transient simulation model. For software validation purposes, a *tgov1* governor model was created as it appears in the PSLF documentation. This particular governor model was selected due to simplicity, and was later expanded upon to include an optional deadband and filtered input delay. The block diagram for the modified tgov1 governor is shown in Figure 2.3. Blocks with a * next to them indicate they are optional, and only inserted into the model if user defined.

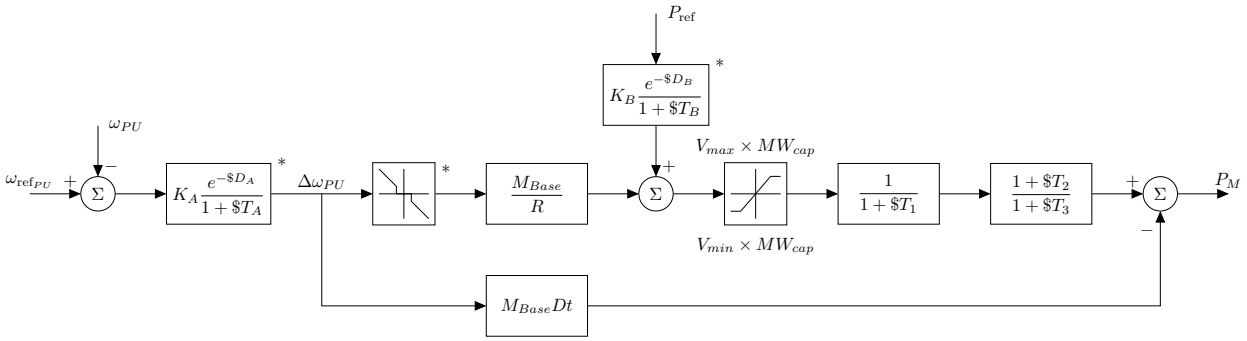

Figure 2.3: Block diagram of modified tgov1 model.

A slightly more generic governor was created based off the governor model used in Power System Toolbox (PST). This generic model, referred to as *genericGov*, is shown in Figure 2.4. The genericGov follows the time constant naming convention of the PST governor and includes the same optional blocks added to the modified tgov1 model.

2.2.5.1 Casting Process for genericGov

Modeling all PSDS governors would be a rather large task. Un-modeled governors encountered in a system dyd file were cast to a genericGov model based on assumed governor type. Table 2.1 shows the relation of PSDS model types to assumed governor setting type.

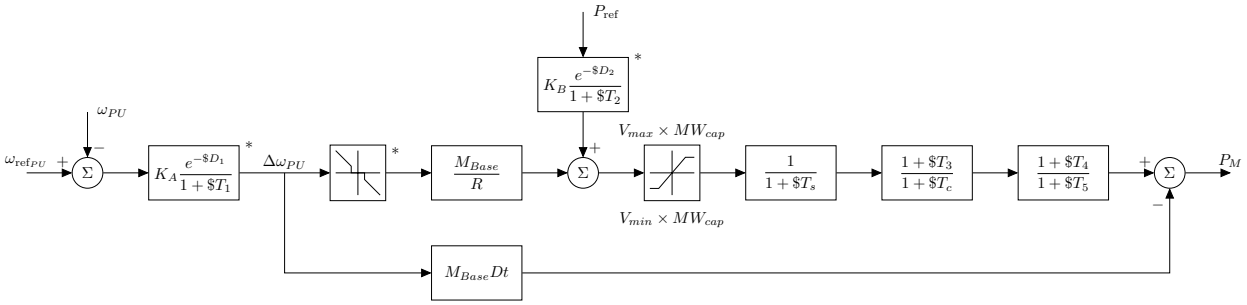

Figure 2.4: Block diagram of genericGov model.

Table 2.1: Generic governor model casting between LTD and PSDS.

genericGov	Steam	Hydro	Gas
PSDS	ccbt1	g2wscc	ggov1
	gast	hyg3	ggov3
	w2301	hygov4	gpwscc
	ieeeg3	hygov	
	ieeeg1	hygovr	
		pidgov	

Universal governor settings, such as permanent droop and MW cap values are collected from the dyd file and used as R and MW_{cap} respectively. The particular time constants for each model were selected according to typical settings associated with prime mover type that a PSDS model is assumed to represent. Table 2.2 lists the time constants used for each genericGov model type.

Table 2.2: Generic governor model parameters.

Parameter	Steam	Hydro	Gas
Ts	0.04	0.40	0.50
Tc	0.20	45.00	10.00
T3	0.00	5.00	4.00
T4	1.50	−1.00	0.00
T5	5.00	0.50	1.00

2.3 General Software Explanation

This section provides an explanation of the Power System Long-Term Dynamic Simulator (PSLTDSim). A flow chart showing a general overview of simulation action is shown in Figure 2.5. A simulation begins with user input of simulation specifications to PSLTDSim. The user input is then used to initialize the required simulation environment by PSLTDSim. The general simulation loop includes performing dynamic calculations and executing any perturbances which are then transferred to PSLF. A power-flow solution is then calculated by PSLF and relevant data sent back to PSLTDSim for logging. Various simulation variables are then checked to verify if the simulation loop is to continue or end. Once the simulation is complete, data is output and plots may be generated for the user to analyze.

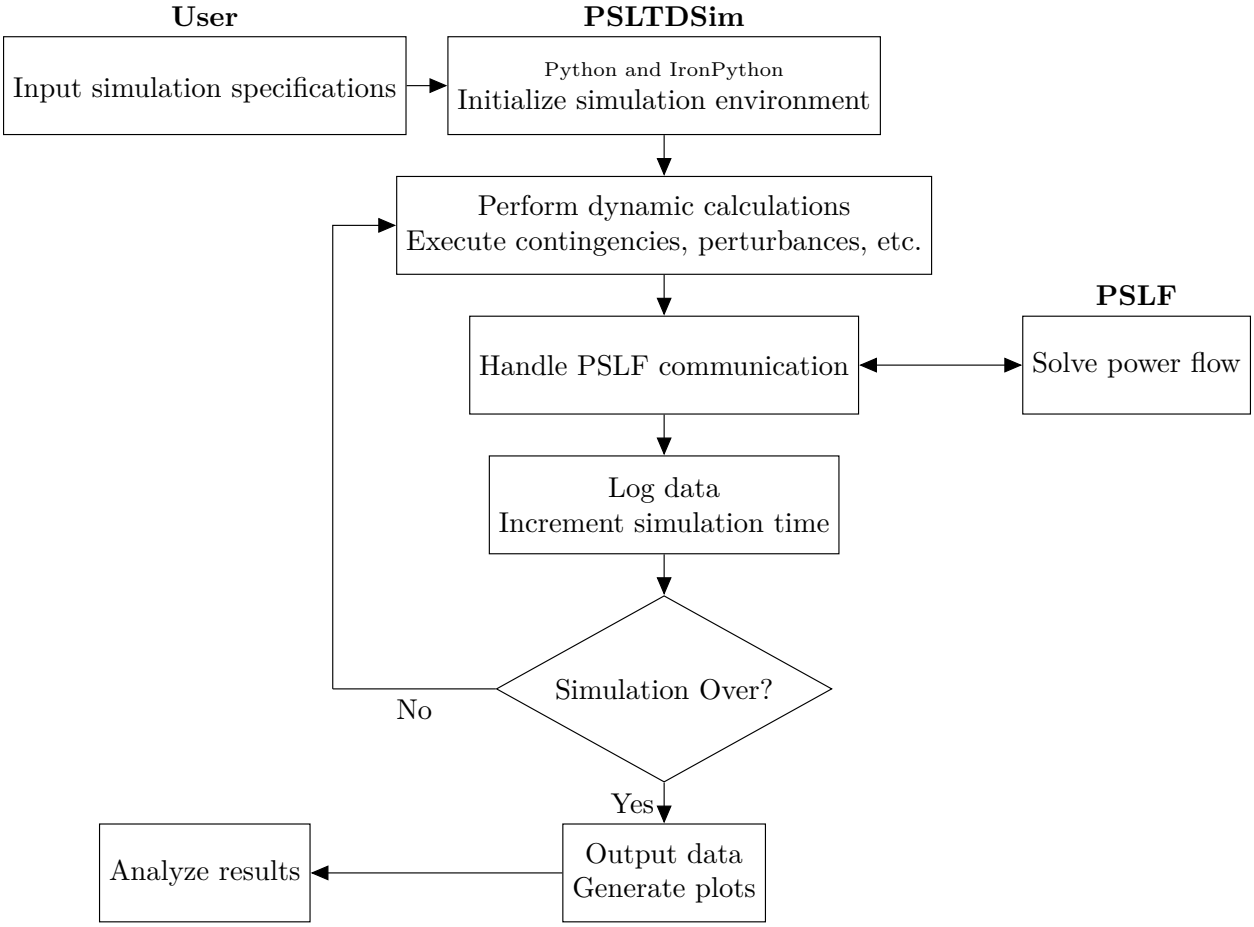

Figure 2.5: High level software flow chart.

2.3.1 Interprocess Communication

A process is another name for an instance of a running computer program. It is common for a computer program to run as a single process, however this is not always the case and not how PSLTDSim operates. Since processes are independent from each other, they do not share a memory space [22]. This effectively means that for two processes share data, some kind of interprocess communication (IPC) must be utilized.

Due to the current state of the GE Python 3 API, Ironpython (IPY) was required for functional PSLF software communication. Unfortunately, IPY is based on Python 2 and does not have packages necessary for numerical computation that Python 3 (PY3) possesses. The solution to these issues was to create a software that has an IPY process and PY3 process that communicate to each other via AMQP. The IPY process acts as a 'middle man' between PY3 and PSLF. Newly calculated dyanmive values from PY3 must be sent to PSLF via IPY, and after each new power-flow solution, PSLF values must be sent to PY3. This cycle continues as along as simulation is required. While the described AMQP solution has been shown to work, it is worth noting that message handling typicall accounts for about one half of all simulation time, and sometimes more in larger cases.

2.3.2 Simulation Inputs

As with any simulation software, PSLTDSim requires specific inputs to operate correctly. Some required input is the same as that used by PSLF, while other input is Python based. Both types of inputs are described in this subsection.

2.3.2.1 PSLF Compatible Input

The power system model input used by PSLTDSim is the same .sav binary file used by, and generated from, PSLF. This is due the reliance of PSLTDSim on the power-flow solver included with PSLF. Python system dynamics are created based on the .dyd text files also used by PSLF. Information on creating .sav or .dyd files is beyond the scope of this text, but may be found in [23].

2.3.2.2 Simulation Parameter Input (.py)

Simulation parameter input is entered in a standard python .py file. Most simulation parameter input is collected in a Python dictionary named simParams. An example of the simParams dictionary defined inside the .py file is shown in Figure 2.6.

```
simParams = {
        'timeStep': 1.0, # seconds
        'endTime': 60.0*8, # seconds
        'slackTol': 1, # MW
        'PY3msgGroup' : 3, # number of Agent msgs per AMQP msg
        'IPYmsgGroup' : 60, # number of Agent msgs per AMQP msg
        'Hinput' : 0.0, # MW*sec of entire system, if !> 0.0, will be calculated in code
        'Dsys' : 0.0, # Damping
        'fBase' : 60.0, # System F base in Hertz
        'freqEffects' : True, # w in swing equation will not be assumed 1 if true
        'assumedV' : 'Vsched', # assummed voltage - either Vsched or Vinit
        # Mathematical Options
        'integrationMethod' : 'rk45',
        # Data Export Parameters
        'fileDirectory' : "\\delme\\200109-delayScenario1\\", # relative path from cwd
        'fileName' : 'SixMachineDelayStep1', # Case name in plots
        'exportFinalMirror': 1, # Export mirror with all data
        'exportMat': 1, # if IPY: requies exportDict == 1 to work
        'exportDict' : 0, # when using python 3 no need to export dicts.
        'logBranch' : True,
}
```

Figure 2.6: An example of a simParams dictionary.

The simParams dictionary contains information required for the simulation to operate, such as time step, end time, base frequency, and slack tolerance. There are also parameters that alter how the simulation operates, such as integration method, inclusion of frequency effects, and how system inertia is calculated. Information related to data collection or export is also included in the simParams dictionary such as whether to log branch information or where the resultant data will be placed and what it will be named. Examples of valid dictionary keys, types of data, units of input, and a brief explanation is shown in Table A.2 located in Appendix A.

In addition to the simParams dictionary, simulation notes, absolute file paths to the desired .sav and .ltd.py file are also defined in this .py file. Dynamic input in the form of .dyd files are defined in a list to allow for using more than one `.dyd` file. The dynamic model overwriting that this feature was meant to incorporate has not been fully implemented as of this writing. An example of a valid simulation parameter input .py is shown in Appendix B as Figure B.1.

2.3.2.3 Long-Term Dynamic Input (.ltd.py)

The required .ltd.py file contains user defined objects related specifically to simulated events and control action. Further, this file is the ideal place for adding additional user input if the need should arise in future software development. Code in the .ltd.py file is standard Python and involves initializing objects attached to a *mirror* object. The mirror object is what PSLTDSim calls the total power system model. A description of the various objects that can be attached to the mirror is presented in the following sections. Most topics introduced are described completely, however, some options are more complex and described in later sections.

2.3.2.3.1 Perturbance List

The only list defined in the .ltd.py file is for entering simulation perturbances (often also referred to as contingencies or events). The list of single quoted strings describing system perturbances is defined as mirror.sysPerturbances. Common perturbances are changes in operating state and power, however, any value in the target agents current value dictionary may be changed. The format of the string is specific to agent type and perturbance, but strings follow the general format of 'Agent Identification : Perturbance Description'. Table 2.3 shows the format for the agent identification part of the perturbance string. Optional parameters are shown in brackets. If no `ID` is specified the first agent found with matching bus values will be chosen.

Table 2.4 describes the various parameters used to specify the action of the pertur-

Table 2.3: Perturbance agent identification options.

Agent Type	Identification Parameters		
`load`	Bus Number	[ID]	
`shunt`	Bus Number	[ID]	
`gen`	Bus Number	[ID]	
`branch`	From Bus Number	To Bus Number	[Circuit ID]

bance agent in the 'Pertrubance Description'. The three valid options for the 'Step Type' and 'Ramp Type' field are `abs`, `rel`, and `per`. To make an absolute change to the new value, the `abs` type should be selected. To alter the target parameter by a relative value, the `rel` option should be used. If a percent change is desired, the `per` type should be used.

Table 2.4: Perturbance agent action options.

Type	Settings				
`step`	Target Parameter	Action Time	New Value	Step Type	
`ramp`	Target Parameter	Start Time	Ramp Duration	New Value	Ramp Type

Figure 2.7 shows various examples of valid perturbance agent definitions. Double quoted strings may be used to clarify perturbance descriptions. It should be noted that the target parameter is case sensitive. Additionally, if stepping a governed generator, both the mechanical power and power reference variables should be changed as shown in line 8 and 9 of Figure 2.7.

```
# Perturbance Examples
mirror.sysPerturbances = [
    'gen 27 : step St 2 0',            # Set gen 27 status to 0 at t=2
    'branch 7 8 2 : step St 10 0 abs', # Trip branch between bus 7 and 8 with ckID=2
    'load 26 : ramp P 2 40 400 rel',   # ramp load 26 P up 400MW over t=2-42 seconds
    'load 9 : "Type" ramp "Target" P "startTime" 2 "RAtime" 40 "RAval" -5 "RAtype" per',
    'shunt 9 4 : step St 32 1',        # Step shunt id 4 on bus 9 on at t=32
    'gen 62 : step Pm 2 -1500 rel',    # Step gen Pm down 1500 MW at t=2
    'gen 62 : step Pref 2 -1500 rel',  # Step gen Pref down 1500 MW at t=2
    ]
```

Figure 2.7: Perturbance agent examples.

2.3.2.3.2 Noise Agent Attribute

A random noise agent was created to add random noise to all system loads. The noise agent is created in the .ltd.py file by defining the mirror.NoiseAgent attribute with the associated agent. Figure 2.8 shows an example of a noise agent being attached to a system mirror. The input arguments are: system mirror reference, percent noise to be added, a boolean value dictating random walk behavior, delay before noise is added, and the random number generator seed value.

```
# Noise Agent Creation
mirror.NoiseAgent = ltd.perturbance.LoadNoiseAgent(mirror, 0.05, walk=True, delay=0,
↪  damping=0, seed=11)
```

Figure 2.8: Noise agent creation example.

At every time step, noise is injected into each load $P_{L,i}$ in the system according to

$$P_{L,i}(t) = P_{L,i}(t-1)[1 \pm N_Z Rand_i + D_{nz}\Delta\omega] \tag{2.5}$$

where N_Z represents the maximum amount of random noise to inject as a percent, $Rand_i$ is a randomly generated number between 0 and 1 inclusive, D_{nz} is the user input damping value, and $\Delta\omega$ is calculated according to Equation **??**. The decision to add or subtract noise

is chosen by a unique randomly generated number. As described in [61], Equation 2.5 creates random walk behavior in load that is representative of real power systems. If random walk behavior is not desired, the walk input argument may be set to False and the scheduled load value is always used as $P_{L,i}(t-1)$. Only mirror reference and percent of desired noise is required for noise agent creation. If no additional arguments are provided, default values for walk, delay, damping and seed are set to False, 0, 0, and 42 respectively.

2.3.2.3.3 Balancing Authority Dictionary

The mirror.sysBA dictionary is defined in the .ltd.py file and is used to configure BA agents. Individual BA dictionaries are nested inside the mirror.sysBA dictionary. The information entered in each nested dictionary describes how that particular BA acts on the specified area. Additional information on BA AGC options and action is presented in Section 3.1.3. Examples of BA parameterization are shown in Figures B.3 and B.4 of Appendix B. A description of each possible field is described in Table A.1 located in Appendix A.

2.3.2.3.4 Load Control Dictionary

To simplify changing all area loads according to a known demand schedule, a load control agent may be defined in the .ltd.py file. Similar to the balancing authority dictionary, each agent is defined as a named dictionary inside a the mirror.sysLoadControl dictionary. The load control agent requires area number, start time, time scale, and a list of tuples for demand information. Specific BA demand data can easily be acquired via the United States Energy Information Administration (EIA) website and saved as a .csv file. A script was written to parse and display demand changes over time as a relative percent change so that real load patterns could be applied to test systems of differing scale. An example of a single area load control agent is shown in Figure 2.9.

```
mirror.sysLoadControl = {
    'testSystem' : {
        'Area': 2,
        'startTime' : 2,
        'timeScale' : 10,
        'rampType' : 'per', # relative percent change
        # Data from: 12/11/2019 PACE
        'demand' : [
            #(time , Precent change from previous value)
            (0, 0.000),
            (1, 3.675),
            (2, 6.474),
            (3, 3.770),
                    ] , # end of demand tuple list
                },# end of testSystem definition
}# end of sysLoadControl dictionary
```

Figure 2.9: Load control agent dictionary definition example.

The list of tuples named 'demand' defines the desired load change over time. The first value in each tuple is assumed to be a time value and the second number is a percent change value. The entered time value is scaled by the 'timeScale' value. It is assumed that both values in the first entry are always zero since there can be no relative change from negative time.

Relative percent ramps are used to alter load value between each entry. For example, if the load control agent shown in Figure 2.9 was used in a simulation, ramps would be created for each load in area 2 that increases load by 3.675% between time 2 and 10, 6.474% between time 10 and 20, and 3.770% between time 20 and 30. In total, a 100 MW load would be 114.548 MW after all ramps executed. The relative percent is based off the value of each load at start of each ramp. This method of relative percent changing allows for other perturbances, such as noise, to be applied to the same load without control conflicts. Alternative ramp types may be employed by changing the 'rampType' parameter, but are not created as of this writing. It should be noted that the first ramp starts at 'startTime', but all other ramps begin according to the calculated scaled time schedule. Additionally,

loads that are off at system initialization (status 0 at time 0) are ignored.

2.3.2.3.5 Generation Control Dictionary

To manipulate generation in the same way a load control agent manipulates load, a generation control agent may be defined in the .ltd.py file. The definition of a generation control agent is very similar to the definition of a load control agent by design, however, differences do exist. Generation control agents are defined in the dictionary named mirror.sysGenerationControl, have a list of strings detailing control generators, and have a time value tuple list named 'forcast'. While the forecast misspelling was unintentional, and can be changed, time has proven it to be a minor detail. Other parameters inside the generation control agent definition (such as area, start time, time scale, and ramp type) function exactly the same as the load control agent. Forecast data is again collected from the EIA website and parsed in the same manner as demand data. Figure 2.10 shows an example of a generation control agent definition.

```
mirror.sysGenerationControl = {
    'testSystem' : {
        'Area': 2,
        'startTime' : 2,
        'timeScale' : 10,
        'rampType' : 'per', # relative percent change
        'CtrlGens': [
            "gen 3 : 0.25",
            "gen 4 : 0.75",
            ],
        # Data from: 12/11/2019 PACE
        'forcast' : [
            #(time , Precent change from previous value)
            (0, 0.000),
            (1, 5.137),
            (2, 6.098),
            (3, 4.471),
                    ],# end of forcast tuple list
                }, #end of testSystem def
   }# end of sysLoadControl dictionary
```

Figure 2.10: Generation control agent dictionary definition example.

The main difference between load and generation control agents is the addition of the 'CtrlGens' list of strings. Each string inside the 'CtrlGens' list is of the form: "gen BusNumber ID : Participation Factor" where ID is optional. If ID is not defined, as shown in Figure 2.10, the first generator on the given bus will be controlled. The participation factor is used to distribute the total requested MW change in a more controlled fashion.

For example, if an area is generating 100 MW at time 0, the total requested area generation change by time 10 would be 5.137 MW. The generator on bus 3 would increase 1.28 MW while the generator on bus 4 would increase 3.85 MW. If a controlled generator has a governor, governor reference will be adjusted instead of mechanical power. Participation factor for all listed generators should always sum to 1.0 or improper distribution **will** occur. It should be noted that not all generation must be controlled for proper percent change of total area output power however, if a controlled machine hits a generation limit, excess changes are ignored.

Relative percent ramps are used to control generators in the same way as load so that a BA can also act on generators under generation control, however, this functionality is untested as of this writing.

2.3.2.3.6 Governor Input Delay and Filtering Dictionary

The inputs to modeled governors may be delayed, filtered, and gained using the Laplace domain block shown in Figure 2.11. If delay is not divisible by the simulation time step, rounding will occur.

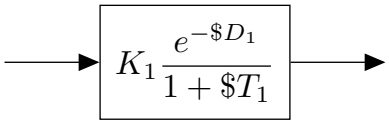

Figure 2.11: Block diagram of delay block.

To modify a governor model with a delay block, parameters may be entered in the

governor delay dictionary mirror.govDelay. Like previously described dictionaries, this is located in the .ltd.py file. Figure 2.12 shows an example of a valid delay dictionary that affects the governor of the generator on bus 3. Note that while genId is optional, if set to None the first found generator on the specified bus is used.

```
mirror.govDelay ={
        'delaygen3' : {
                'genBus' : 3,
                'genId' : None, # optional
                # (delay parameter, filter time constant, optional gain)
                'wDelay' : (40,30),
                'PrefDelay' : (10, 0),
        }, # end of 'delaygen3' definition
}# end of govDelay dictionary
```

Figure 2.12: Governor delay dictionary definition example.

Tuples are used to enter delay block parameters. Using Figure 2.3 as reference, the 'wDelay' tuple contains settings for D_A, T_A, and K_A respectively. Likewise, the 'PrefDelay' tuple contains D_B, T_B, and K_B. If the tuple contains three entries, the third is assigned to the optional gain associated with each block, otherwise K_x is one.

2.3.2.3.7 Governor Deadband Dictionary

A BA agent may be used to set area wide deadbands, but it is also possible to specify a single deadband for any governed generator. Individual governor deadband dictionaries are defined inside the mirror.govDeadBand dictionary in the .ltd.py file. Settings in each governor deadband dictionary will override any deadband settings specified by the BA dictionary. Figure 2.13 shows three examples of valid governor deadband definitions.

Entering 'step' or 'ramp' as a value for the 'GovDeadbandType' will create a step or ramp deadband at the given 'GovDeadband'. A non-linear droop governor deadband may be configured by setting the 'GovDeadbandType' to 'NLDroop' and entering desired 'GovAlpha' and 'GovBeta' values. An explanation of the different deadband types is presented in Section

```
mirror.govDeadBand ={
    'gen3DB' : {
        'genBus' : 3,
        'genId' : None, # optional
        'GovDeadbandType' : 'ramp', # step, ramp, nldroop
        'GovDeadband' : 0.036, # Hz
        },
    'gen1DB' : {
        'genBus' : 1,
        'genId' : None, # optional
        'GovDeadbandType' : 'nldroop', # step, ramp, nldroop
        'GovAlpha' : 0.016, # Hz, used for nldroop
        'GovBeta' : 0.036, # Hz, used for nldroop
        },
    'gen4DB' : {
        'genBus' : 4,
        'genId' : None, # optional
        'GovDeadbandType' : 'step', # step, ramp, nldroop
        'GovDeadband' : 0.036, # Hz
        'GovAlpha' : 0.016, # Hz, used for nldroop
        'GovBeta' : 0.036, # Hz, used for nldroop
        },
}# end of govDelay dictionary
```

Figure 2.13: Governor deadband dictionary definition example.

3.1.1.

2.3.2.3.8 Definite Time Controller Dictionary

During long simulations, system loading may change from initial values by more than ±20%. Such changes can cause voltage issues that require the setting or un-setting of components contributing to available system reactive power. This can be accomplished by defining a definite time controller (DTC) agent in the mirror.DTCdict dictionary. Other general programmable logic operations may also be accomplished using a DTC agent. Figure 2.14 shows an example of a valid DTC definition where a shunt is actuated by changes in bus voltage or branch MVAR flow. It should be noted that instead of using a specific 'tarX' in a timers 'act' field, operations on any off or on target can be accomplished by using

'anyOFFtar' or 'anyONtar' respectively.

```
# Definite Time Controller Definitions
mirror.DTCdict = {
        'ExampleDTC' : {
                'RefAgents' : {
                        'ra1' : 'bus 8 : Vm',
                        'ra2' : 'branch 8 9 1 : Qbr', # branches defined from, to, ckID
                },# end Reference Agents
                'TarAgents' : {
                        'tar1' : 'shunt 8 2 : St',
                        'tar2' : 'shunt 8 3 : St',
                }, # end Target Agents
                'Timers' : {
                        'set' :{
                                'logic' : "(ra1 < 1.0) or (ra2 < -15)",
                                'actTime' : 30, # seconds of true logic before act
                                'act' : "tar1 = 1",
                        },# end set
                        'reset' :{
                                'logic' : "(ra1 > 1.04) or (ra2 > 15)",
                                'actTime' : 30, # seconds of true logic before act
                                'act' : "tar1 = 0",
                        },# end reset
                        'hold' : 60, # minimum time between actions
                }, # end timers
        },# end ExampleDTC definition
}# end DTCdict
```

Figure 2.14: Definite time controller dictionary definition example.

Each DTC employs a set and a reset timer and may have a hold timer if hold time is set larger than zero. Multiple references and targets can be associated with a DTC, however, as of this writing only one action can be associated with each timer. Any logic string entered in a timer uses the given key names for each reference or target and is evaluated using standard Python logic conventions.

2.3.3 Simulation Initialization

To clarify the explanation of simulation initialization, the entire process can be broken into three parts: process creation, mirror initialization, and dynamic initialization pre-simulation loop. The first part (process creation) involves creating and configuring various software processes that enable the simulation to run. The majority of part two (mirror initialization) revolves around collecting data from PSLF to create a Python duplicate, or mirror, of the power system model. The last part of simulation initialization (dynamic initialization pre-simulation loop) handles user input and creates PY3 dynamics before entering the simulation loop.

2.3.3.1 Process Creation

System initialization begins in PY3 with package imports and creation of truly global variables before user input from the .py file is handled. The .py file includes debug flags, simulation notes, simulation parameters, and file locations of the .sav, .dyd, and .ltd.py files. If all file locations are valid, PY3 initializes AMQP queues, sends appropriate initialization information to the IPY queue, starts the IPY_PSLTDSim process, and then waits for an IPY response message.

The IPY process begins by also importing required packages and setting references to certain imported packages as truly global variables. An IPY AMQP agent is created and linked to the AMQP host generated by the PY3 process. This allows the IPY process to receive initialization information from the PY3 AMQP message sent before the IPY process was evoked. IPY uses the received initialization information to load the GE Python API which is then used to load the .sav and .dyd into PSLF. Upon successful file loading in PSLF, a global reference to the object is created.

2.3.3.2 Mirror Initialization

Once the PSLF specific files are loaded into the GE software, initialization of the Python environment, or system model, may begin. The system model, referred to as the system mirror, or just mirror, is a single object that almost all other Python objects are

created inside. The mirror is a single system object with a recursive data structure that allows any object the ability to reference any other object as long as they both share a reference to the mirror and are themselves referenced by the mirror. While the previous sentence may seem overly complicated, its not, and the use of such a linking technique eliminated the need for global variables outside of imported packages and also allowed for a single file containing **all** simulation data to be easily exported at the end of a simulation.

The system mirror begins its initialization by creating placeholder variables for simulation parameters, counters, and future agent collections. Specific case parameters, such as the number of buses or generators, are collected from PSLF to be used later in model verification processes.

Before any specific power system object data is collected, an initial power flow is performed to ensure that the loaded .sav is solvable, and to establish a steady state system operating point. System agents, representing power system objects like buses and loads, are then added to the mirror. The adding process queries PSLF for any buses in a specific area, checks each found bus for any connected system components, and creates Python agents for relevant objects. The querying and adding process continues until all system buses are accounted for. Each type of found object is added to a running tally so that it can be compared with the expected values collected earlier. Once all system buses have been found or accounted for, any created agents that are intended to log data are collected into a list for simpler group stepping.

Each area tally of found agents is checked for coherency with expected values. Any inconsistencies between the amount of found and expected objects will trigger warnings, but the simulation will not stop if this occurs. This choice is due to differences between what is counted in PSLF as a valid area object and PSLTDSim, which has the option to ignore islanded objects.

Dynamic model information from the specified .dyd file is then parsed. Collected machine or governor parameters are used to create PSLF model information objects (PMIOs)

inside the mirror. These PMIOs collect inertia H and MW cap for each machine as well as turbine type, governor MW cap, and permanent droop R from governors. Other information, such as MVA base, is also collected for both types of model. This process is required as the .dyd values for certain parameters overwrite pre-existing values that may be saved inside the .sav file.

Once the .dyd file is parsed and all PMIOs are created, the combined system inertia is calculated. For each found generator PMIO, the associated mirror agent is located and updated so that H and Mbase values match those found in the .dyd. The total system inertia is calculated according to Equation 2.1 and user input settings are interpreted so that any requested changes, such as scaling or alternative system inertia inputs, are handled correctly.

Mirror search dictionaries are then created to simplify and speed up agent searches and the global slack generator is identified. The global slack generator is important to locate as its variance from an expected value dictates accelerating power re-distribution and power-flow solution iterations during the simulation loop. Once search dictionaries are created, the IPY model is fully initialized. The mirror is then saved to disk and an AMQP message is sent to PY3 with the mirror location.

2.3.3.3 Dynamic Initialization Pre-Simulation Loop

After the handoff AMQP message is sent to PY3, IPY initializes values required for simulation and awaits an AMQP message from PY3 before entering its simulation loop.

PY3 uses the information received from IPY to load the newly created system mirror so that PY3 can perform further initializations such as creating any dynamic agents (governors), calculating area frequency characteristics and maximum capacities, executing any .ltd code, and creating the associated agents from the .ltd input.

PY3 dynamics are initialized to ensure R is on the correct PU base and any MW caps from dyd parsing is applied. Additionally, any limiting values for governor output are accounted for, and any deadbands or delays are created. Finally, before entering the PY3 simulation loop, agents that are designed to log values initialize blank lists for expected

values.

2.3.4 Simulation Loop

Once simulation initialization is complete, and the system mirror is initialized, the simulation loop is executed. Figure 2.15 shows the major actions that are processed each time step, but does not include details concerning AMQP communication. AMQP messages are sent and received during the 'Update' blocks and the system mirrors are checked for coherency at these times as well.

The simulation loop can be viewed as starting with the increment of simulation time followed by the stepping of any dynamic agents. These dynamic agents calculate the new system frequency and perform any required governor responses. Generator electrical power is then set equal to generator mechanical power. This step is the beginning of forming the next power-flow solution initial conditions.

Perturbance agents are then stepped. This means that any steps, ramps, or noise type events are performed, agents in both system mirrors are updated, and any related system value is changed accordingly. For example, the tripping of a generator requires the system inertia to change as well as the amount of power in the system. The variable ΔP_{pert} is used to keep track of system power changes. Additionally, BA action takes place at this time.

After all perturbance related actions are executed, accelerating power is calculated and distributed to the system according to generator inertia. The PSLF system is updated with any required values and a power-flow solution is attempted. If the solution diverges the simulation ends and any collected data is output. If the solution does not diverge, the magnitude of any slack error is checked against the slack error tolerance. If the slack error is larger than the slack tolerance, the error is redistributed to the system until the resulting error is within tolerance or the solution diverges. If the power-flow solution diverges ducing accelerating power redistribution the simulation ends and any collected data is output.

Once the system has converged to a point where the slack error is less than the slack tolerance, PSLF values for generator real and reactive power and bus voltage and angle are used to update the PY3 mirror. The electric power output of the system is summed for use

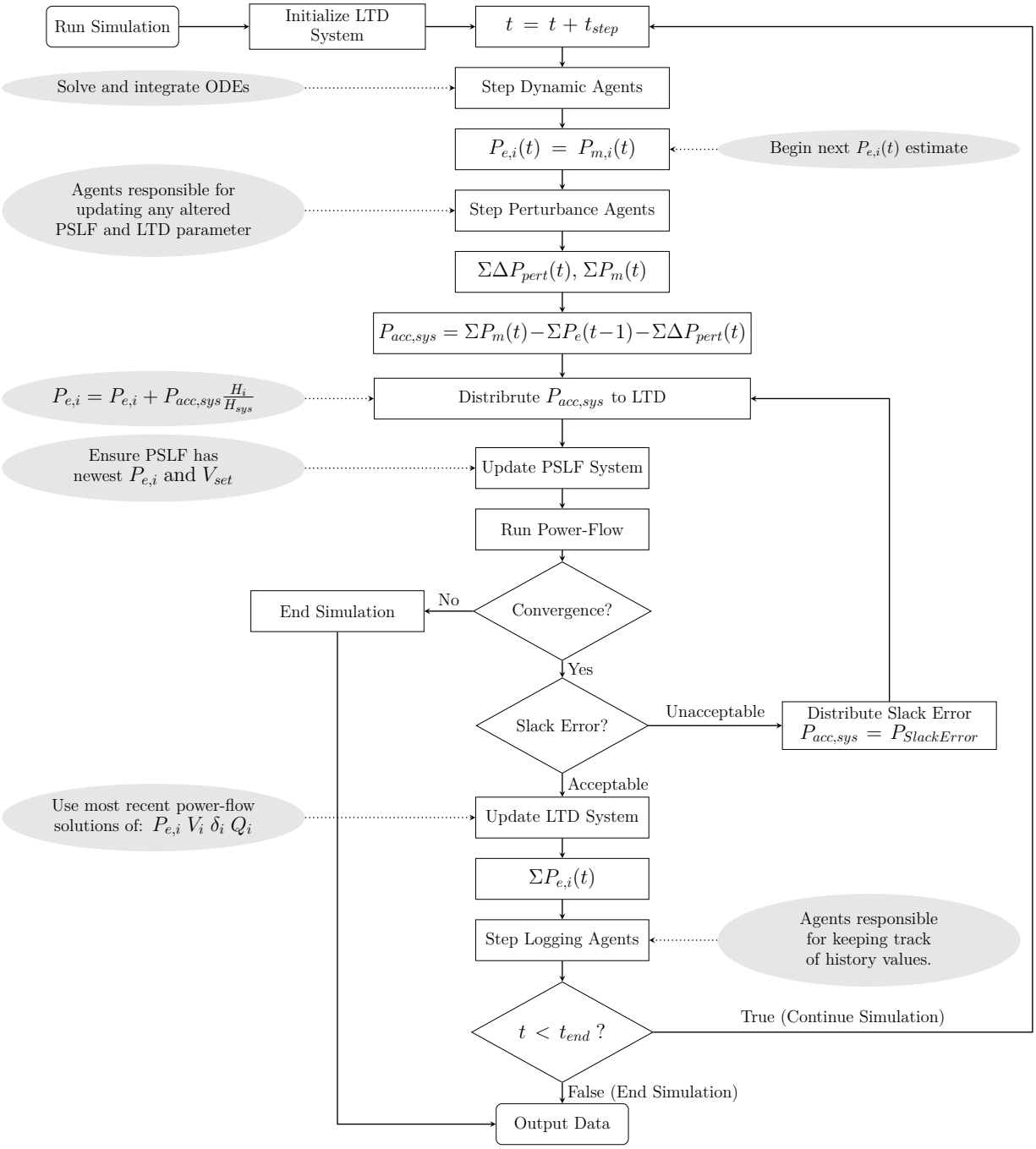

Figure 2.15: Simulation time step flowchart.

in calculating system accelerating power in the next time step. Any logging agents are then stepped and data for that particular simulation time step are recorded. Finally, system time is checked and if the simulation is complete, any collected data is output. If the simulation is not complete, the simulation time is incremented by the time step and the cycle repeats

itself again.

2.3.5 Simulation Outputs

Pre-defined data is collected by any agent with logging ability. Current agents with this ability are machines, loads, shunts, branches, transformers, areas, balancing authorities, and the system mirror itself. When a simulation is complete, the final system mirror is exported via the Python package `shelve` and options exist to export some data as a MATLAB .mat file. The .mat output is accomplished by combining various agent log dictionaries into a single dictionary. As such, only data deemed useful for validating the software is included.

It should be noted that numerous plot functions were created to easier visualize and validate python mirror data. Python plot functions are located in the PSLTDSim package plot folder, while the MATLAB validation plots are located in the GitHub repository only.

3 Engineering Applications

To further the investigation an engineering problem, the following simulated controls have been written and, when used in conjunction with the previously described software features, *may* prove useful.

3.1 Simulated Balancing Authority Controls

Simulated balancing authority controls affect governor response and AGC operation. These controls are defined in the sysBA dictionary in the .ltd.py file. Figure 3.1 shows an example of a BA parameter dictionary as it would appear in a .ltd.py file. A complete list of BA agent options is shown in Table A.1 in Appendix A. The following sections detail most sysBA dictionary keys.

```
# Balancing Authority Definition
mirror.sysBA = {
    'BA1':{
        'Area': 1,
        'B': "0.9 : permax", # MW/0.1 Hz
        'AGCActionTime': 30.00, # seconds
        'ACEgain' : 1.0,
        'AGCType':'TLB : 4', # Tie-Line Bias
        'UseAreaDroop' : False,
        'AreaDroop' : 0.05,
        'IncludeIACE' : True,
        'IACEconditional': True,
        'IACEwindow' : 30, # seconds - size of window - 0 for non window
        'IACEscale' : 1/5,
        'IACEdeadband' : 0, # Hz
        'ACEFiltering': 'PI : 0.04 0.0001',
        'AGCDeadband' : None, # MW? -> not implemented
        'GovDeadbandType' : 'nldroop', # step, None, ramp, nldroop
        'GovDeadband' : .036, # Hz
        'GovAlpha' : 0.016, # Hz - for nldroop
        'GovBeta' : 0.036, # Hz - for nldroop
        'CtrlGens': ['gen 1 : 0.5 : rampA',
                     'gen 2 1 : 0.5 : rampA',
                     ]
        }
```

```
26      }
```

Figure 3.1: Single sysBA dictionary definition.

3.1.1 Governor Deadbands

The FERC maximum deadband is 36 mHz[18]. However, the execution of a governor deadband is not explicitly detailed and left to generator operators to configure. PSLTDSim offers a deadband agent that can apply various deadbands to the incoming $\Delta\omega$. Figure 3.2 graphically depicts how the various available deadband options differ.

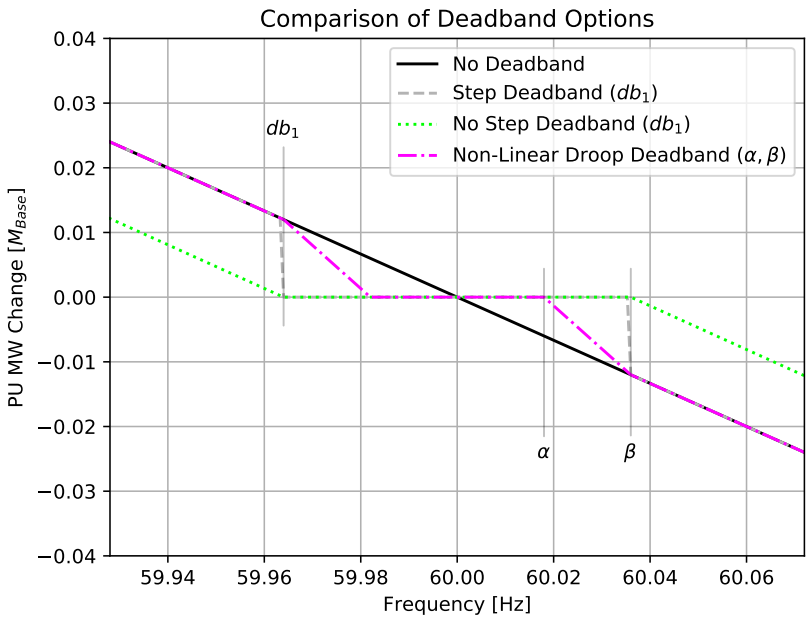

Figure 3.2: Examples of available deadband action.

A step deadband simply nullifies any $\Delta\omega$ whose absolute value is less than the prescribed deadband. A no-step, or ramp, deadband removes the step characteristic by essentially pushing the original droop curve out to deadband thresholds. While the no-step deadband has no abrupt changes, the actual governor response will always be below the ideal droop response. To eliminate this oversight, a non-linear droop option was created that ramps from a set Hz deadband to the ideal droop curve.

Configuring a deadband is done via the sysBA parameter dictionary or the governor deadband dictionary in the .ltd.py file. The dictionary keys are the same in the sysBA dictionary as the govDeadBand dictionary. Details about creating deadbands using a governor deadband dictionary is presented in Section 2.3.2.3.7.

3.1.2 Area Wide Governor Droops

The typically recommended FERC droop is 5%[18]. Area wide governor droops can be specified in the sysBA dictionary that overwrite any droop setting previously read from a .dyd. All active governors in an area will use the specified 'AreaDroop' value if 'UseAreaDroop' is True. This setting allows for fast and easy configuration of simulations aimed at exploring droop settings.

3.1.3 Automatic Generation Control

A block diagram of the AGC model employed by PSLTDSim is shown in Figure 3.3. Distributed ACE (DACE) may vary quite a lot depending on control parameters. Simulation settings related to frequency bias, ACE integrating and filtering, and conditional summing are explained in the following sections.

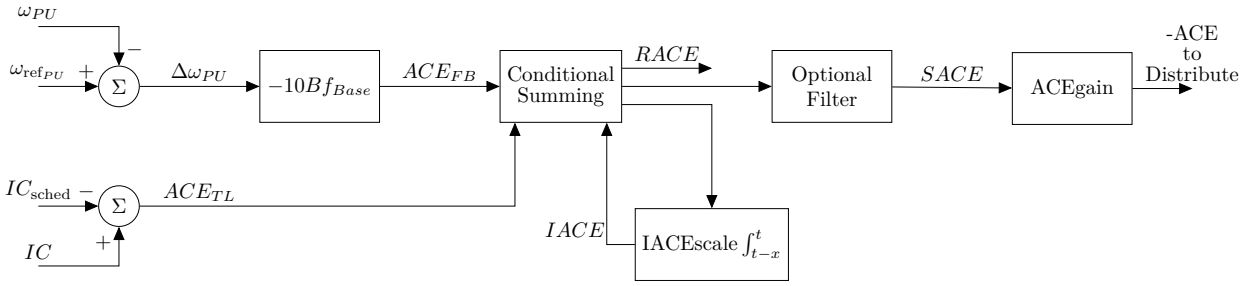

Figure 3.3: Block diagram of ACE calculation and manipulation.

3.1.3.1 Frequency Bias

Choosing a desired frequency bias B can be accomplished in a number of different ways. All methods are configured by the string entered as the 'B' value in the sysBA dictionary. The format of the 'B' string is "Float Value : B type". The available B types are scalebeta, perload, permax, and abs.

The scalebeta type will scale the automatically calculated area frequency response characteristic β by the given float value. The perload type will set B equal to the current area load times the given float value. The permax type will set B equal to the maximum area capacity times the given float value. The abs type will set B equal to the given float value. Note that the units on B are MW/0.1 Hz and, despite B being a negative number, is entered as a positive value.

A variable frequency bias may be used in distributed ACE signals. If the 'BVgain' dictionary entry is a non-zero value, a variable frequency bias B_V is calculated as

$$B_V = B_F \left(1 + K_B|\Delta\omega|\right) \tag{3.1}$$

where B_F is the fixed bias value, and K_B is the user entered value for 'BVgain'. It should be noted that $\Delta\omega$ is calculated according to Equation **??** and is a PU value. This leads to seemingly large required values of 'BVgain' before effects are noticeable. To clarify, B_V is only used in ACE calculations that are meant to be distributed to the generation fleet, RACE is always calculated using B_F.

3.1.3.2 Integral of Area Control Error

As previously shown in Figure 3.3, ACE may be integrated and fed back into the conditional summing block. The default signal sent to the integrator is RACE as AGC is meant to drive RACE to zero. Settings related to this process are configured in the sysBA dictionary. Integral of ACE (IACE) parameters are 'IACEwindow', 'IACEscale', and 'IACEdeadband'. IACEwindow defines the length in seconds of the moving window integrator. If IACEwindow is set to zero, integration will be continuous (i.e. for all time). IACEscale acts as a gain of the output integral value. IACEdeadband specifies the frequency deviation in Hz between which integration values will stop being added into the conditional summing block.

3.1.3.3 Conditional Area Control Error Summing

Depending on the type of AGC agent chosen, ACE is calculated in different ways. The Tie-Line Bias (TLB) agent has multiple types of conditional ACE calculations. All conditionals involve comparing the sign of a value to the sign of frequency deviation. The main idea behind this conditional summing is to ensure that only events occurring inside an area will receive an AGC response. Options are available to allow for continued frequency response as well. Table 3.1 shows the conditional summations and associated TLB type.

Table 3.1: Tie-line bias AGC type ACE calculations.

TLB Type	ACE Calculation
0	$ACE_{FB} + ACE_{TL}$
1	$ACE_{FB} + ACE_{TL} * [\text{sgn}(\Delta\omega) == \text{sgn}(ACE_{TL})]$
2	$ACE_{FB} + ACE_{TL} * [\text{sgn}(\Delta\omega) == \text{sgn}(ACE_{FB} + ACE_{TL})]$
3	$ACE_{FB} * [\text{sgn}(\Delta\omega) == \text{sgn}(ACE_{FB})] + ACE_{TL} * [\text{sgn}(\Delta\omega) == \text{sgn}(ACE_{TL})]$
4	$[ACE_{FB} + ACE_{TL}] * [\text{sgn}(\Delta\omega) == \text{sgn}(ACE_{FB} + ACE_{TL})]$

3.1.3.4 Area Control Error Filtering

The calculated ACE can be put through a filter, or smoothed, to become smoothed ACE (SACE). The three basic filters created were low pass, integral, and proportional and integral (PI). Block diagrams of these filters are shown in Figure 3.4.

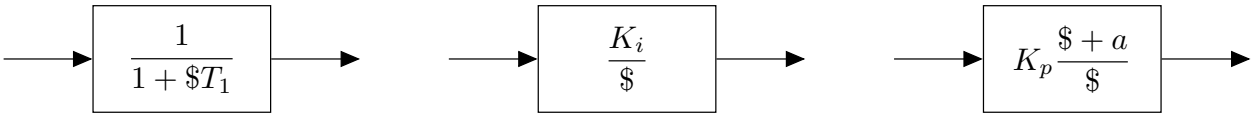

Figure 3.4: Block diagrams of optional ACE filters.

The selection and configuration of a filter is done in the BA parameter dictionary via the 'ACEFiltering' key value. The format of the string input to the 'ACEFiltering' key is "type : val1 val2". Valid filter types are lowpass, integrator, and pi. The low pass and integrator take only one value while the PI filter takes two. In the case of the low pass filter, the passed in value is set as the low pass time constant. The value passed in with an

integrator filter is simply a gain. The first PI value is used as a proportional gain value K_p and the second value describes the ratio between integral and proportional gain a.

3.1.3.5 Controlled Generators and Participation Factors

Each BA is configured with a list of controlled generators that receive AGC signals. These generators, or power plants, are given a participation factor between 1 and 0. The participation factor dictates the percent of ACE signal each agent receives. A check is done to ensure each BA has a total participation factor of one, however, if the sum of participation factors is not one, only a warming is issued. It is up to the user to enter reasonable values. Additionally, each list value of the 'CtrlGens' key describes if the signal should be applied as a step or a ramp. Ramps are best when single units are receiving ACE, while steps are useful for power plants that are intended to handle distribution of ACE independently.

4 Future Work

As with any software project, future work revolves around expansion and refinement. The number of dynamic agents, or governor models, could be expanded without bound. Alternatively, a more refined way of casting un-modeled governors could be devised. Something involving automated one machine infinite bus scenarios, step analysis, and 2nd order approximation has been suggested.

Exponential load models and under-load transformer tap changers are some possible agents that should be accounted for. Powerplant agents that act as plant controllers have been conceptually modeled, but not implemented to their full extent. To better capture voltage changes, and thus reactive power, some form of exciter modeling may be desired, although with the simplification of machine models, this task may prove difficult. Voltage scheduling of generator buses via perturbance agents should be possible, but is untested as of this writing.

PSLTDSim is open-source code that relies on proprietary software for essential functions. To move away from this reliance, a method for creating system models and solving power flows would have to be created. Any changes would have to be incorporated into the way PSLTDSim creates a system mirror, solves a power flow, and updates the mirror. A semi-clear point to perform this break would be when AMQP messages are sent. If PSLTDSim did not rely on PSLF, there would also be no need for AMQP messages to be sent as all code would be PY3. This would not only speed up simulation, but create a more clear flow of events and result in a fully open-source simulation tool.

To accommodate for transient, or oscillatory events, PSLTDSim could be coupled with the ideas presented in [60]. This would essentially involve a variable time step and some set threshold that would automatically switch between TSPF and CTS simulation.

5 Bibliography

[1] .NET Foundation. (2018). Ironpython overview, [Online]. Available: https://ironpython.net/.

[2] P. M. Anderson and A. A. Fouad, *Power System Control and Stability*, Second Edition. Wiley-Interscience, 2003.

[3] J. Audenaert, K. Verbeeck, and G. V. Berghe. (2009). Mulit-agent based simulation for boarding, CODeS Research Group, [Online]. Available: https://www.semanticscholar.org/paper/Multi-Agent-Based-Simulation-for-Boarding-Audenaert-Verbeeck/24ba2c3190de5b7162c37e81581b062cda3e4d54.

[4] A. Aziz, A. Mto, and A. Stojsevski, "Automatic generation control of multigeneration power system," Journal of Power and Energy Engineering, 2014.

[5] J. Carpentier, "'To be or not to be modern' that is the question for automatic generation control (point of view of a utility engineer)," International Journal of Electrical Power & Energy Systems, 1985.

[6] R. W. Cummings, W. Herbsleb, and S. Niemeyer. (2010). Generator governor and information settings webinar, North American Electric Reliability Corporation, [Online]. Available: https://www.nerc.com/files/gen-governor-info-093010.pdf.

[7] F. P. deMello and R. Mills, "Automatic generation control part II - digital control techniques," IEEE PES Summer Meeting, 1972.

[8] DEQ. (2004). Montana electric transmission grid: Operation, congestion, and issues, DEQ, [Online]. Available: https://leg.mt.gov/content/publications/Environmental/2004deq_energy_report/transmission.pdf.

[9] EIA. (2016). U.s. electric system is made up of interconnections and balancing authorities, [Online]. Available: https://www.eia.gov/todayinenergy/detail.php?id=27152.

[10] EIA. (2019). July2019map.png, U.S. Energy Information Administration, [Online]. Available: https://www.eia.gov/electricity/data/eia860m/.

[11] EIA. (2019). U.s. electric system operating data, U.S. Energy Information Administration, [Online]. Available: https://www.eia.gov/realtime_grid/.

[12] EIA. (2019). U.s. energy mapping system, U.S. Energy Information Administration, [Online]. Available: https://www.eia.gov/state/maps.php?v=Electricity.

[13] E. Ela and J. Kemper, "Wind plant ramping behavior," National Renewable Energy Laboratory, 2009.

[14] M. D. of Environmental Quality. (2020). Colstrip statistics, DEQ, [Online]. Available: http://deq.mt.gov/DEQAdmin/mfs/AllColstrip/DEQAdmin/mfs.

[15] ERCOT. (2017). Ercot-internconnection_branded.jpg, ERCOT, [Online]. Available: http://www.ercot.com/news/mediakit/maps.

[16] J. H. Eto, J. Undrill, C. Roberts, P. Mackin, and J. Ellis, "Frequency control requirements for reliable interconnection frequency response," Energy Analysis and environmental Impacts Division Lawrence Berkeley National Laboratory, 2018.

[17] D. Fabozzi and T. Van Cutsem, "Simplified time-domain simulation of detailed long-term dynamic models," IEEE Xplore, 2009.

[18] FERC, "Essential reliability services and the evolving bulk-power system–primary frequency response," Federal Energy Regulatory Commission, Docket No. RM16-6-000 Order No. 842, Feb. 2018.

[19] FERC. (2019). About ferc, [Online]. Available: https://www.ferc.gov/about/about.asp.

[20] T. Garnock-Jones and G. M. Roy. (2017). Introduction to pika, [Online]. Available: https://pika.readthedocs.io/en/stable/.

[21] GE Energy, *Mechanics of running pslf dynamics*, 2015.

[22] GeeksforGeeks. (2017). Thread in operating system, GeeksforGeeks, [Online]. Available: https://www.geeksforgeeks.org/thread-in-operating-system/.

[23] General Electric International, Inc, *PSLF User's Manual*, 2016.

[24] W. B. Gish, "Automatic generation control algorithm - general concepts and application to the watertown energy control center," Bureau of Reclamation Engineering and Research Center, 1980.

[25] J. D. Glover, M. S. Sarma, and T. J. Overbye, *Power System Analysis & Design*, 5e. Cengage Learning, 2012.

[26] R. Gonzales. (2019). Pg&e transmission lines caused california's deadliest wildfire, state officials say, NPR, [Online]. Available: https://www.npr.org/2019/05/15/723753237/pg-e-transmission-lines-caused-californias-deadliest-wildfire-state-officials-sa.

[27] M. Goossens, F. Mittelbach, and A. Samarin, *The LaTeX Companion*. Addison-Wesley, 1993.

[28] R. Hallett, "Improving a transient stability control scheme with wide-area synchrophasors and the microwecc, a reduced-order model of the western interconnect," Master's thesis, Montana Tech, 2018.

[29] E. Heredia, D. Kosterev, and M. Donnelly, "Wind hub reactive resource coordination and voltage control study by sequence power flow," IEEE, 2013.

[30] J. JMesserly. (2008). Electricity_grid_simple-_north_america.svg, United States Department of Energy, [Online]. Available: `https://commons.wikimedia.org/wiki/File:Electricity_grid_simple-_North_America.svg`.

[31] T. Kennedy, S. M. Hoyt, and C. F. Abell, "Variable, non-linear tie-line frequency bias for interconnected systems control," IEEE Transactions on Power Systems, 1988.

[32] Y. G. Kim, H. Song, and B. Lee, "Governor-response power flow (grpf) based long-term voltage stability simulation," IEEE T&D Asia, 2009.

[33] G. Kou, P. Markham, S. Hadley, T. King, and Y. Liu, "Impact of governor deadband on frequency response of u.s. eastern interconnection," IEEE Transactions on Smart Grid, 2016.

[34] D. Kuhlman, *A Python Book: Beginning Python, Advanced Python, and Python Exercises*. 2009.

[35] P. Kundur, *Power System Stability and Control*. McGraw-Hill, 1994.

[36] K. H. LaCommare and J. H. Eto, "Understanding the cost of power interruptions to u.s. electricity consumers," Ernest Orlando Lawrence Berkeley National Laboratory, 2004.

[37] M. Liedtke. (2020). Court approves pg&e's $23b bankruptcy financing package, AP News, [Online]. Available: `https://apnews.com/b70582ee8d4bb7f781553215612da993`.

[38] P. Maloney. (2018). What is the value of electric reliability for your operation? [Online]. Available: `https://microgridknowledge.com/power-outage-costs-electric-reliability/`.

[39] Y. Mobarak, "Effects of the droop speed governor and automatic generation control agc on generator load sharing of power system," International Journal of Applied Power Engineering, 2015.

[40] NERC, "Frequency response initiative report," North American Electric Reliability Corporation, 2012.

[41] NERC, "Procedure for ero support of frequency response and frequency bias setting standard," North American Electric Reliability Corporation, 2012.

[42] NERC, "Standard bal-003-1.1 — frequency response and frequency bias setting," North American Electric Reliability Corporation, 2015.

[43] NERC, "Standard bal-001-2 – real power balancing control performance," North American Electric Reliability Corporation, 2016.

[44] NERC. (2017). About nerc, [Online]. Available: https://www.nerc.com/AboutNERC/Pages/default.aspx.

[45] NERC. (2017). Nerc interconnections map, [Online]. Available: https://www.nerc.com/AboutNERC/keyplayers/PublishingImages/NERC%20Interconnections.pdf.

[46] NERC, "Bal-002-3 – disturbance control standard – contingency reserve for recovery from a balancing contingency event," North American Electric Reliability Corporation, 2018.

[47] NERC, "Frequency response annual analysis," North American Electric Reliability Corporation, 2018.

[48] NERC. (2020). Glossary of terms used in nerc reliability standards, NERC, [Online]. Available: https://www.nerc.com/files/glossary_of_terms.pdf.

[49] NERC Resources Subcommittee, "Balancing and frequency control," North American Electric Reliability Corporation, 2011.

[50] NERC Resources Subcommittee, "Bal-001-tre-1 — primary frequency response in the ercot region," North American Electric Reliability Corporation, 2016.

[51] P. W. Parfomak, "Physical security of the u.s. power grid: High-voltage transformer substations," Congressional Research Service, 2014.

[52] Python Software Foundataion. (2019). About python, [Online]. Available: https://www.python.org/about/.

[53] B. Rand. (2018). Agent-based modeling: What is agent-based modeling? Youtube, [Online]. Available: https://www.youtube.com/watch?v=FVmQbfsOkGc.

[54] C. W. Ross, "Error adaptive control computer for interconnected power systems," IEEE Transactions on Power Apparatus and Systems, 1966.

[55] G. van Rossum. (2009). A brief timeline of python, [Online]. Available: https://python-history.blogspot.com/2009/01/brief-timeline-of-python.html.

[56] J. Sanchez-Gasca, M. Donnelly, R. Concepcion, A. Ellis, and R. Elliott, "Dynamic simulation over long time periods with 100% solar generation," Sandia National Laboratories, SAND2015-11084R, 2015.

[57] P. W. Sauer, M. A. Pai, and J. H. Chow, *Power System Dynamics and Stability With Synchrophasor Measurement and Power System Toolbox*, Second Edition. John Wiley & Sons Ltd, 2018.

[58] SciPy developers. (2019). About scipy, [Online]. Available: `https://www.scipy.org/about.html`.

[59] K. Siegel. (2012). The true cost of power outages, Yale Environment Review, [Online]. Available: `https://environment-review.yale.edu/true-cost-power-outages-0`.

[60] M. Stajcar, "Power system simulation using an adaptive modeling framework," Master's thesis, Montana Tech, 2016.

[61] C. W. Taylor and R. L. Cresap, "Real-time power system simulation for automatic generation control," IEEE Transactions on Power Apparatus and Systems, 1976.

[62] D. Trudnowski, "Properties of the dominant inter-area modes in the wecc interconnect," Montana Tech, 2012.

[63] T. Van Cutsem and C. Vournas, *Voltage Stability of Electric Power Systems*, 1st ed. Springer US, 1998.

[64] WECC. (2015). About wecc, [Online]. Available: `https://www.wecc.org/Pages/AboutWECC.aspx`.

A Large Tables

This appendix is used to present large tables too distracting for inclusion in their respective sections.

Table A.1: Balancing authority dictionary input information.

Key	Type	Units	Example	Description
B	String	MW/0.1Hz	"1.0 : permax"	Describes the frequency bias scaling factor B used in the ACE calculation. Various Options exist.
AGCActionTime	Float	Seconds	5	Time between AGC dispatch messages.
AGCType	String	-	"TLB : 2"	Dictates which AGC routine to use and type specific options.
UseAreaDroop	Boolean	-	FALSE	If True, all governed generators under BA control will use the area droop.
AreaDroop	Float	Hz/MW	0.05	Droop value to use if 'UseAreaDroop' is True.
IncludeIACE	Boolean	-	TRUE	If True, include IACE in ACE calculation
IACEconditional	Boolean	-	FALSE	Adds IACE to ACE if signs of deltaw and IACE match.
IACEwindow	Integer	Seconds	60	Defines the length of moving integration window to use in IACE. If set to 0, integration takes place for all time.
IACEscale	Float	-	0.0167	Value used to scale IACE.
IACEdeadband	Float	Hz	0.036	Absolute value of system frequency where IACE will not be calculated below.
ACEFiltering	String	-	PI : 0.03 0.001'	String used to dictate which filter agent is created and filter specific parameters.
AGCDeadband	Float	MW	1.5	Value of ACE to ignore sending in AGC dispatch. Not implemented as of this writing.
GovDeadbandType	String	-	step'	Type of deadband to be applied to area governors.
GovDeadband	Float	Hz	0.036	Absolute value of system frequency that governors will not respond below.
GovAlpha	Float	Hz	0.016	Specific to 'NLDroop' type of deadband. Specifies lower bound of non-linear droop.
GovBeta	Float	Hz	0.036	Specific to 'NLDroop' type of deadband. Specifies upper bound of non-linear droop.
CtrlGens	List of Strings	-	-	List of generators, participation factor, and dispatch signal type.

Table A.2: Simulation parameters dictionary input information.

Key	Type	Units	Example	Description
timeStep	float	Seconds	1	Simulated time between power-flow solutions
endTime	float	Seconds	1800	Number of seconds simulation is to run for.
slackTol	float	MW	0.5	MW Value that slack error must be below for returned solution to be accepted.
PY3msgGroup	integer	-	3	Number of messages to combine into one AMQP message for PY3 to IPY communication.
IPYmsgGroup	integer	-	60	Number of messages to combine into one AMQP message for IPY to PY3 communication.
Hinput	float	MW sec	0	Value to use for total system inertia. Units are MW*sec. If set to 0.0, system inertia will be calculated from the given .sav information.
Dsys	float	PU	0	Value of system damping used in swing equation and governor models. While this option is available, it is untested and typically set to 0.0.
fBase	float	Hz	60	Value of base system frequency.
freqEffects	boolean	-	True	If True, the ω used in the swing equation will be the current system frequency. If this is set to False then ω will be set equal to 1 for the swing equation calculation
integrationMethod	string	-	'rk45'	This option defines how the swing equation is integrated to find current frequency. Valid options are 'rk45', 'ab', and 'euler'. The default is the 'euler' method which is a simple forward Euler integration. The 'ab' option uses a two step Adams-Bashforth method and the 'rk45' options uses the scipy `solve_ivp` function that utilizes an explicit Runge-Kutta 4(5) method.
fileDirectory	string	-	"\\delme\\"	This is a relative path location from the folder where PSLTDSim is exectued in which the output files are saved to.
fileName	string	-	"SimTest"	This is that name used to save files.
exportFinalMirror	int	-	1	If this value is 1 a final system mirror will be exported. If this value is 0 no final mirror will be exported.
exportMat	int	-	1	If this value is 1 a MATLAB .mat file will be exported. If this value is 0 no MATLAB .mat file will be exported.

B Code Examples

This appendix is used to present code examples too large for inclusion in the body of the text. Some examples span multiple pages. To adhere to document format requirements, the description of each code example is presented after the corresponding code figure.

```
# Format of required info for batch runs.
debug = 0
AMQPdebug = 0
debugTimer = 0

simNotes = """
AGC TUNING (no delay)
Delay over response test
Loss of generation in area 1 at t=2
Delayed action by area 2
AGC in both areas
"""

# Simulation Parameters Dictionary
simParams = {
        'timeStep': 1.0, # seconds
        'endTime': 60.0*8, # seconds
        'slackTol': 1, # MW
        'PY3msgGroup' : 3, # number of Agent msgs per AMQP msg
        'IPYmsgGroup' : 60, # number of Agent msgs per AMQP msg
        'Hinput' : 0.0, # MW*sec of entire system, if !> 0.0, will be calculated in code
        'Dsys' : 0.0, # Damping
        'fBase' : 60.0, # System F base in Hertz
        'freqEffects' : True, # w in swing equation will not be assumed 1 if true
        # Mathematical Options
        'integrationMethod' : 'rk45',
        # Data Export Parameters
        'fileDirectory' : "\\delme\\200109-delayScenario1\\", # relative path from cwd
        'fileName' : 'SixMachineDelayStep1',
        'exportFinalMirror': 1, # Export mirror with all data
        'exportMat': 1, # if IPY: requies exportDict == 1 to work
        'exportDict' : 0, # when using python 3 no need to export dicts.
        'deleteInit' : 0, # Delete initialized mirror
        'assumedV' : 'Vsched', # assummed voltage - either Vsched or Vinit
        'logBranch' : True,
}
```

```

savPath = r"C:\LTD\pslf_systems\sixMachine\sixMachineTrips.sav"
dydPath = [r"C:\LTD\pslf_systems\sixMachine\sixMachineDelay.dyd"]
ltdPath = r".\testCases\200109-delayScenario1\sixMachineDelayStep1.ltd.py"
```

Figure B.1: An example of a full .py simulation file.

```
# Format of required info for batch runs.
debug = 0
AMQPdebug = 0
debugTimer = 0

simNotes = """
agc with deadband and nz, area 2 perturbance TLB 4
"""

# Simulation Parameters Dictionary
simParams = {
    'timeStep': 1.0,
    'endTime': 60.0*10,
    'slackTol': 1,
    'PY3msgGroup' : 3,
    'IPYmsgGroup' : 60,
    'Hinput' : 0.0, # MW*sec of entire system, if !> 0.0, will be calculated in code
    'Dsys' : 0.0, # Untested
    'fBase' : 60.0, # System F base in Hertz
    'freqEffects' : True, # w in swing equation will not be assumed 1 if true
    # Mathematical Options
    'integrationMethod' : 'rk45',
    # Data Export Parameters
    'fileDirectory' : "\\delme\\200325-smFinal\\", # relative path from cwd
    'fileName' : 'smAGCt4Ex1',
    'exportFinalMirror': 1, # Export mirror with all data
    'exportMat': 1, # if IPY: requies exportDict == 1 to work
    'exportDict' : 0, # when using python 3 no need to export dicts.
    'deleteInit' : 0, # Delete initialized mirror
    'assumedV' : 'Vsched', # assummed voltage - either Vsched or Vinit
    'logBranch' : True,
    }

savPath = r"C:\LTD\pslf_systems\sixMachine\sixMachineLTD.sav"
```

```
dydPath = [r"C:\LTD\pslf_systems\sixMachine\sixMachineLTD.dyd"]
ltdPath = r".\testCases\200325-smFinals\smAGCt4Ex1.ltd.py"
```

Figure B.2: Required .py file for external AGC event with conditional ACE.

```
# Perturbances
mirror.sysPerturbances = [
    'gen 5 : step Pm 2 -150 rel',
    ]

mirror.NoiseAgent = ltd.perturbance.LoadNoiseAgent(mirror, 0.05, walk=True, delay=0,
↪  damping=0, seed=11)

# Balancing Authorities
mirror.sysBA = {
    'BA1':{
        'Area':1,
        'B': "0.9 : permax", # MW/0.1 Hz
        'AGCActionTime': 30.00, # seconds
        'ACEgain' : 1.0,
        'AGCType':'TLB : 4', # Tie-Line Bias
        'UseAreaDroop' : False,
        'AreaDroop' : 0.05,
        'IncludeIACE' : True,
        'IACEconditional': True,
        'IACEwindow' : 30, # seconds - size of window - 0 for non window
        'IACEscale' : 1/5,
        'IACEdeadband' : 0, # Hz
        'ACEFiltering': 'PI : 0.04 0.0001',
        'AGCDeadband' : None, # MW? -> not implemented
        'GovDeadbandType' : 'nldroop', # step, None, ramp, nldroop
        'GovDeadband' : .036, # Hz
        'GovAlpha' : 0.016, # Hz - for nldroop
        'GovBeta' : 0.036, # Hz - for nldroop
        'CtrlGens': ['gen 1 : 0.5 : rampA',
                     'gen 2 1 : 0.5 : rampA',
                     ]
        },
    'BA2':{
        'Area':2,
        'B': "0.9 : permax", # MW/0.1 Hz
        'AGCActionTime': 45.00, # seconds
```

```
        'ACEgain' : 1.0,
        'AGCType':'TLB : 4', # Tie-Line Bias
        'UseAreaDroop' : False,
        'AreaDroop' : 0.05,
        'IncludeIACE' : True,
        'IACEconditional': True,
        'IACEwindow' : 45, # seconds - size of window - 0 for non window
        'IACEscale' : 1/3,
        'IACEdeadband' : 0, # Hz
        'ACEFiltering': 'PI : 0.04 0.0001',
        'AGCDeadband' : None, # MW? -> not implemented
        'GovDeadbandType' : 'nldroop', # step, None, ramp, nldroop
        'GovDeadband' : .036, # Hz
        'GovAlpha' : 0.016, # Hz - for nldroop
        'GovBeta' : 0.036, # Hz - for nldroop
        'CtrlGens': ['gen 3 : 1.0 : rampA',]
        },
    }
```

Figure B.3: Required .ltd file for external AGC event with conditional ACE.

```
mirror.NoiseAgent = ltd.perturbance.LoadNoiseAgent(mirror, 0.05, walk=True, delay=0,
↪  damping=0, seed=11)

# Balancing Authorities
mirror.sysBA = {
    'BA1':{
        'Area':1,
        'B': "0.9 : permax", # MW/0.1 Hz
        'AGCActionTime': 30.00, # seconds
        'ACEgain' : 1.0,
        'AGCType':'TLB : 4', # Tie-Line Bias
        'UseAreaDroop' : False,
        'AreaDroop' : 0.05,
        'IncludeIACE' : True,
        'IACEconditional': True,
        'IACEwindow' : 30, # seconds - size of window - 0 for non window
        'IACEscale' : 1/5,
        'IACEdeadband' : 0, # Hz
        'ACEFiltering': 'PI : 0.04 0.0001',
        'AGCDeadband' : None, # MW? -> not implemented
        'GovDeadbandType' : 'nldroop', # step, None, ramp, nldroop
```

```
        'GovDeadband' : .036, # Hz
        'GovAlpha' : 0.016, # Hz - for nldroop
        'GovBeta' : 0.036, # Hz - for nldroop
        'CtrlGens': ['gen 1 : 0.5 : rampA',
                     'gen 2 1 : 0.5 : rampA',
                     ]
        },
    'BA2':{
        'Area':2,
        'B': "0.9 : permax", # MW/0.1 Hz
        'AGCActionTime': 45.00, # seconds
        'ACEgain' : 1.0,
        'AGCType':'TLB : 4', # Tie-Line Bias
        'UseAreaDroop' : False,
        'AreaDroop' : 0.05,
        'IncludeIACE' : True,
        'IACEconditional': True,
        'IACEwindow' : 45, # seconds - size of window - 0 for non window
        'IACEscale' : 1/3,
        'IACEdeadband' : 0, # Hz
        'ACEFiltering': 'PI : 0.04 0.0001',
        'AGCDeadband' : None, # MW? -> not implemented
        'GovDeadbandType' : 'nldroop', # step, None, ramp, nldroop
        'GovDeadband' : .036, # Hz
        'GovAlpha' : 0.016, # Hz - for nldroop
        'GovBeta' : 0.036, # Hz - for nldroop
        'CtrlGens': ['gen 3 : 1.0 : rampA',]
        },
    }

# Load and Generation Cycle Agents
mirror.sysGenerationControl = {
    'BPATDispatch' : {
        'Area': 1,
        'startTime' : 2,
        'timeScale' : CTRLtimeScale,
        'rampType' : 'per', # relative percent change
        'CtrlGens': [
            "gen 1 : 0.5",
            "gen 2 1 : 0.5",
            ],
        # Data from: 12/11/2019 PACE
        'forcast' : [
            #(time , Precent change from previous value)
```

```
            (0, 0.0),
            (1, 5.8),
            (2, 8.8),
            (3, 9.9),
            (4, 4.0),
            ],
        }, #end of generation controller def
    'CAISODispatch' : {
        'Area': 2,
        'startTime' : 2,
        'timeScale' : CTRLtimeScale,
        'rampType' : 'per', # relative percent change
        'CtrlGens': [
            "gen 4 : 1.0",
            ],
        # Data from: 12/11/2019 PACE
        'forcast' : [
            #(time , Precent change from previous value)
            (0, 0.0),
            (1, 0.7),
            (2, 7.5),
            (3, 11.2),
            (4, 4.4),
            ],
        }, #end of generation controller def
    }

mirror.sysLoadControl = {
    'BPATDemand' : {
        'Area': 1,
        'startTime' : 2,
        'timeScale' : CTRLtimeScale,
        'rampType' : 'per', # relative percent change
        # Data from: 12/11/2019 BPAT
        'demand' : [
            #(time , Precent change from previous value)
            (0, 0.000),
            (1, 3.2),
            (2, 8.2),
            (3, 9.3),
            (4, 3.8),
        ] ,
    }, # end of demand agent def
    'CAISODemand' : {
```

```
        'Area': 2,
        'startTime' : 2,
        'timeScale' : CTRLtimeScale,
        'rampType' : 'per', # relative percent change
        # Data from: 12/11/2019 CAISO
        'demand' : [
            #(time , Precent change from previous value)
            (0, 0.000),
            (1, 3.0),
            (2, 7.0),
            (3, 10.5),
            (4, 4.4),
        ] ,
    },# end of demand load control definition
}# end of loac control definitions

# Definite Time Controller Definitions
mirror.DTCdict = {
    'bus8caps' : {
        'RefAgents' : {
            'ra1' : 'bus 8 : Vm',
            'ra2' : 'branch 8 9 1 : Qbr', # branches defined from, to, ckID
            },# end Referenc Agents
        'TarAgents' : {
            'tar1' : 'shunt 8 2 : St',
            'tar2' : 'shunt 8 3 : St',
            'tar3' : 'shunt 8 4 : St',
            'tar4' : 'shunt 8 5 : St',
            'tar5' : 'shunt 8 6 : St',
            }, # end Target Agents
        'Timers' : {
            'set' :{ # set shunts
                'logic' : "(ra1 < 1.0)", # or (ra2 < -26)",
                'actTime' : 30, # seconds of true logic before act
                'act' : "anyOFFTar = 1", # set any target off target = 1
            },# end set
            'reset' :{ # reset shunts
                'logic' : "(ra1 > 1.04)",# or (ra2 > 26)",
                'actTime' : 30, # seconds of true logic before act
                'act' : "anyONTar = 0", # set any target On target = 0
            },# end reset
            'hold' : 90, # minimum time between actions
            }, # end timers
        },# end bus8caps
```

```
'bus9caps' : {
    'RefAgents' : {
        'ra1' : 'bus 9 : Vm',
        'ra2' : 'branch 8 9 1 : Qbr', # branches defined from, to, ckID
        },# end Referenc Agents
    'TarAgents' : {
        'tar1' : 'shunt 9 2 : St',
        'tar2' : 'shunt 9 3 : St',
        'tar3' : 'shunt 9 4 : St',
        'tar4' : 'shunt 9 5 : St',
        'tar5' : 'shunt 9 6 : St',
        }, # end Target Agents
    'Timers' : {
        'set' :{ # set shunts
            'logic' : "(ra1 < 1.0)",
            'actTime' : 45, # seconds of true logic before act
            'act' : "anyOFFTar = 1", # set any target off target = 1
        },# end set
        'reset' :{ # reset shunts
            'logic' : "(ra1 > 1.04)",
            'actTime' : 45, # seconds of true logic before act
            'act' : "anyONTar = 0", # set any target On target = 0
        },# end reset
        'hold' : 120, # minimum time between actions
        }, # end timers
    },# end bus8caps
}# end DTCdict
```

Figure B.4: Required .ltd file for forecast demand scenario with noise and deadbands.