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

–Software and Simulated Controls Documentation– ver. 0.1

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
AC	Alternating Current
ACE	Area Control Error
AGC	Automatic Generation Control
AMQP	Advanced Message Queue Protocol
API	Application Programming Interface
CTS	Classical Transient Stability
DACE	Distributed ACE
DTC	Definite Time Controller
EIA	United States Energy Information Administration
ERCOT	Electric Reliability Council of Texas
ERO	Electric Reliability Organization
FERC	Federal Energy Regulatory Commission
FTL	Frequency Trigger Limit
IACE	Integral of ACE
II	Inadvertent Interchange
IPC	Interprocess Communication
IPY	IronPython
LFC	Load Frequency Control
LTD	Long-Term Dynamic
NERC	North American Electric Reliability Corporation
ODE	Ordinary Differential Equation
PI	Proportional and Integral
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

Term	Definition
PY3	Python version 3.x
PyPI	Python Package Index
RACE	Reported ACE
SACE	Smoothed ACE
SS	Steady State
TLB	Tie-Line Bias
TSPF	Time-Sequenced Power Flow
WECC	Western Electricity Coordinating Council

1 Software Background

1.1 Classical Transient Stability Simulation

Classical transient stability (CTS) simulation is 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 model issues may lead to questionable stability conditions over the course of longer simulations.

General Electric's Positive Sequence Load Flow (PSLF) is an industry standard CTS simulation program that has a rather large library of dynamic components. Additionally, due to the longevity of PSLF, a wide variety of full system models have been created. For example, multiple full WECC base cases have been modeled in PSLF over the past 20 years and are continuously being updated to reflect the most current system. Newer verisons of PSLF have a Python 3 API and a .NET API. While both of these software packages are at various points of development and functionality, they 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 [26]. Guido Van Rossum first implemented a version of Python in December of 1989 [41]. Python is freely available and distributable for multiple computing platforms [11].

1.2.1 Python Specific Data Types

Python has various data types, most of which are common to other programing languages such as integer, string, list, and float, however 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. Dictionaries were found to be useful as 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 of lists or arrays as other programming languages may require and also allows for simple searching and iteration.

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.

1.2.2 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). PSLTDSim utilizes various 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 [7]. 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 for specific ODEs, namely system frequency. The Python implementation of the advanced message queuing protocol (AMQP) used for AMQP communication in this project was the Python version of Pika [13].

1.2.3 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# [12] and is able to use the common language runtime (CLR) package required for properly interacting with the GE PSLF .NET library. As such, IPY, and only 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 is only compatible python

packages compatible with 2.X. Python 3 (PY3) is 'the future of Python' and has many more useful community created packages. As of this writing, the current version of Python is 3.8 though most code was written using 3.7.

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 is to enable a Python 3 process to communicate with an IronPython 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 queues are then accessed by other processes that can then receive any queued messages.

It should be noted that Erlang was required to allow use of RabbitMQ and the PY3/IPY Pika package was required so that Python could use AMQP. While detailed descriptions of these softwares is beyond the scope of this paper, 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 PSLTDSim to function.

1.4 Agent Based Modeling

Agent Based Modeling (ABM) is oriented around the idea that any situation can be described by agents in an environment, and a definition of agent-agent and agent-environment interactions [39]. This ideology is applied to the coding of this project in that a power system acts as the environment, and all power system objects are treated as agents. The ABM coding style lends itself towards modular coding and natural expandability.

2 Software Tool

2.1 Time-Sequenced Power Flows

Differences between TSPF and CTS simulation techniques stem from the time frame each on focuses on. TSPF is focused on long-term events that take places over tens of minutes or longer, while CTS simulation focuses on events that are tens of seconds in duration. This difference in focus leads to a TSPF time-step of 0.5 to 1 seconds while transient simulation uses sub-cycle time-steps of approximately 4.667 ms in a 60 Hz system.

Additionally, the calculations involved with each method is different as well. While each method starts with a power-flow solution, CTS simulation then performs back calculations to set initial states of various dynamic models. Future states are then dictated by dynamic model interaction. TSPF also uses of dynamic models, but updated values are sent to a power-flow solver every step, and the power-flow solution dictates new system values.

2.2 Simulation Assumptions and Simplifications

Due to the large time steps involved with time-sequenced power flows, numerous assumptions and simplifications from transient stability methods were made. This section details such assumptions and simplifications.

2.2.1 General Assumptions and Simplifications

It is assumed that the system of study is stable and remains stable for the entirety of the simulation. The focus of this tool is on long-term operation of power systems, not transient stability. There are other software packages available better suited to study events where system stability is in question.

Power system stabilizers (PSS) are not modeled as it is assumed that the system will not be perturbed enough to require this action. Additionally, ideal exciters are assumed as modern exicters are typically fast enough to maintain reference voltage. Future work can be done to incorporate such modeling should the need arise.

The largest assumption is that useful results will still be produced that are capable of solving engineering problems.

2.2.2 Time Step Assumptions and Simplifications

With a larger time step, typically of 1 second, multiple assumptions can be made concerning power system behavior. Models used in transient stability simulations are greatly simplified for use in PSLTDSim.

Intermachine oscillations are ignored since subsynchronous resonances are sub-second and the time resolution is not great enough to capture these phenomena. Additionally, in the long-term, these effects are minor when the system is stable.

Machine models are greatly simplified. The only details collected from each machine model present in the `.dyd` file are model type, MW cap, machine MVA base, and machine inertia.

2.2.3 Combined System Frequency

Instead of a frequency being calculated for each bus, a single combined swing equation is used to model only one combined system frequency. As shown in Equation 2.1, accelerating power from the entire system, as well as total system inertia H_{sys}, is used to calculate $\dot{\omega_{sys}}$.

$$\dot{\omega}_{sys} = \frac{1}{2H_{sys}} \left(\frac{P_{acc,sys}}{\omega_{sys}} - D_{sys} \Delta\omega_{sys} \right) \tag{2.1}$$

After integration, $\dot{\omega_{sys}}$ leads to a single combined system frequency. For completeness, a damping term D_{sys} is included in Equation 2.1, but often set to zero.

2.2.4 Distribution of Accelerating Power

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

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

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

The system accelerating power is then distributed to all generators in the system according to machine inertia. Equation 2.3 details the method for this.

$$P_{e,i} = P_{e,i} - P_{acc,sys}\left(\frac{H_i}{H_{sys}}\right) \tag{2.3}$$

Once all accelerating power is distributed, the new value for each generators power output is used to solve a power flow. If the resulting power supplied by the slack generator is larger than the set slack tolerance, the difference is redistributed according to Equation 2.3 until slack tolerance is met, or a maximum number of iterations take place.

2.2.5 Governor models

Long-term dynamic models do not require the detail of a full transient simulation model. A simple governor model, *tgov1*, was created as it appears in the PSLF documentation to be used for validation purposes. The tgov1 model was later expanded to include an optional input deadband and configurable low pass filtered delay. The block diagram for the modified tgov1 governor is shown in Figure 2.1. Blocks with a * next to them indicate they are optional and only inserted if user defined.

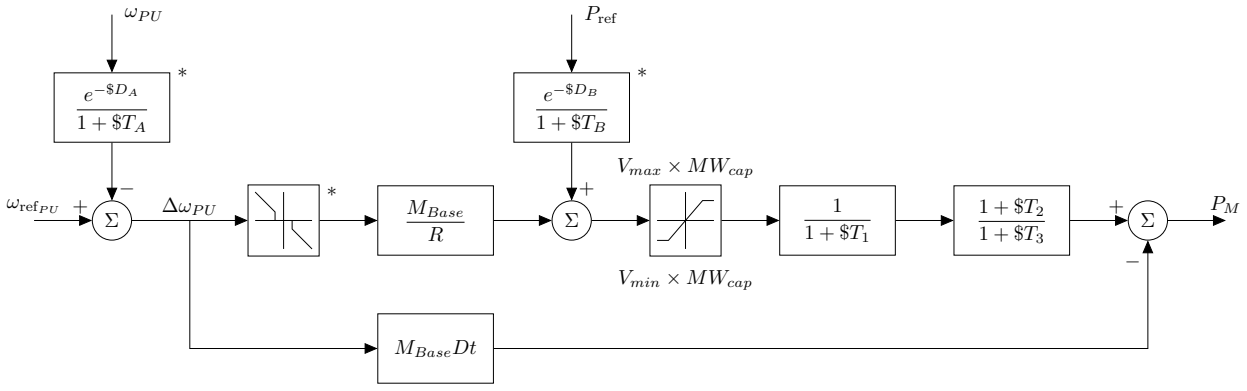

Figure 2.1: Block diagram of modified tgov1 model.

A 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 2.2. The genericGov uses the same time constant variables as a PST governor and adds the optional blocks added to the tgov| model.

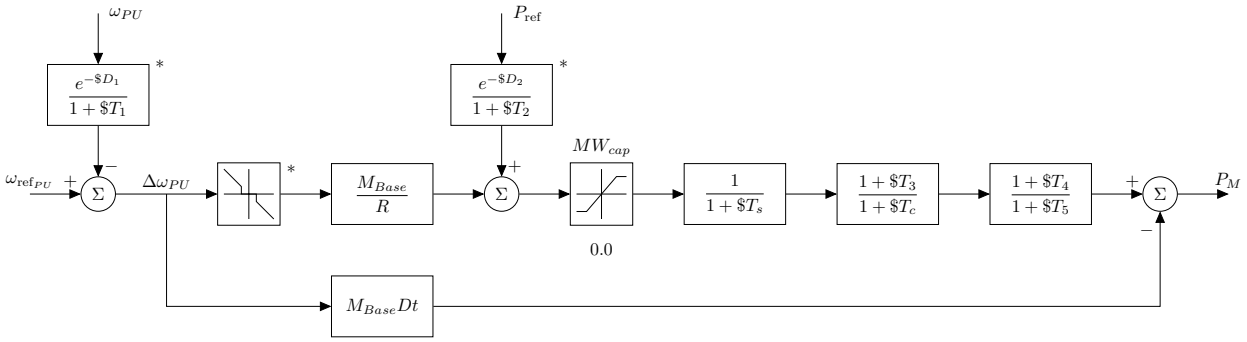

Figure 2.2: Block diagram of genericGov model.

2.2.5.1 Casting Process for genericGov

Since modeling all PSDS governors would be a rather large task, un-modeled governors were cast to a genericGov model according to prime mover model type with typical settings. The only data required from the dyd file is the droop setting and MW cap. Which PSDS model types were cast to what governor settings is shown in Table 2.1. Table 2.2 shows the generic time constants used for each cast type in the 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	

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 PSLTDSim. A flow chart showing a general overview of simulation action is shown in Figure 2.3. Any 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 performed 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.

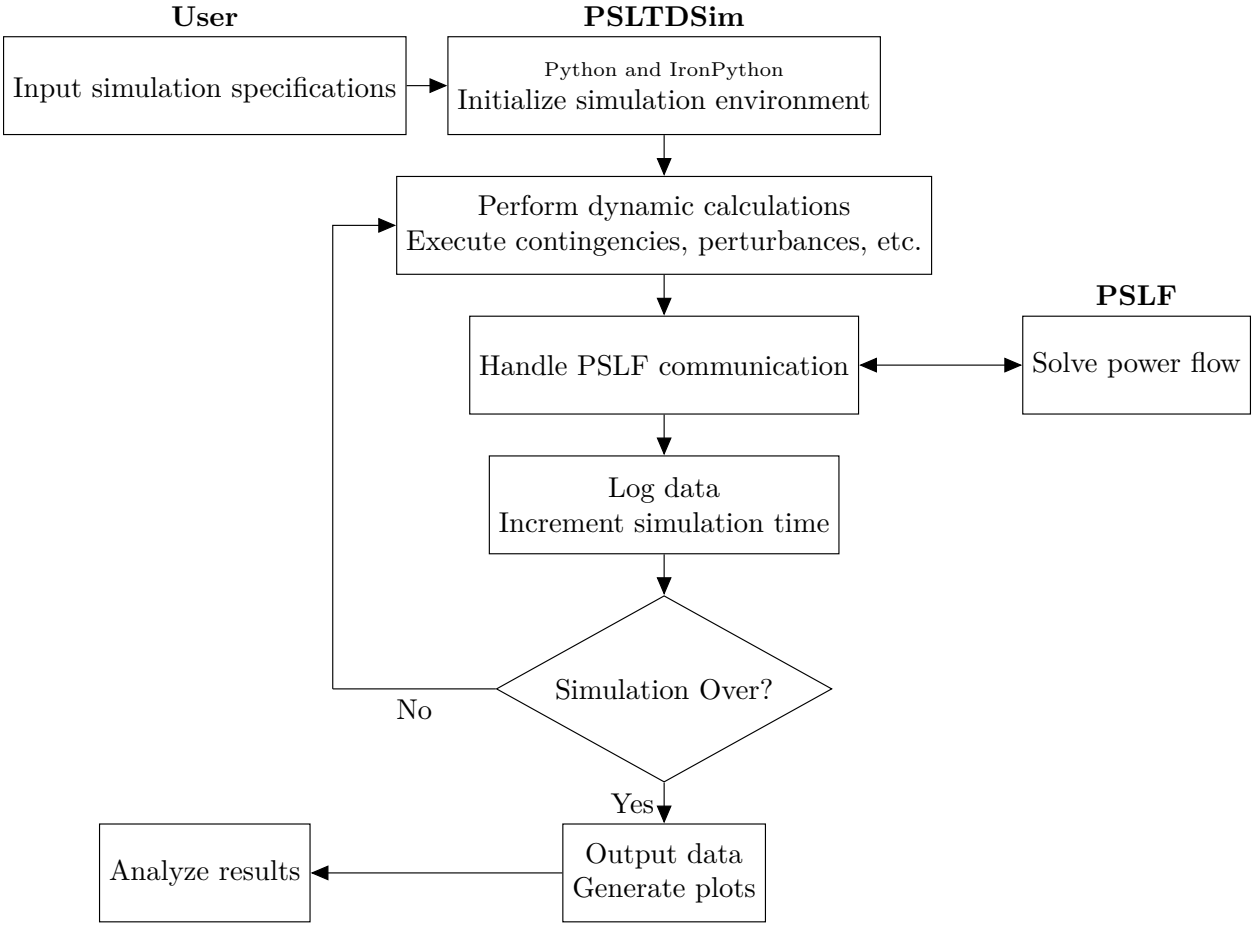

Figure 2.3: High level software flow chart.

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 what is happening in PSLTDSim. Since processes are independent from each other, they do not share a memory space [46]. 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 was required for functional PSLF software communication. Unfortunately, Ironpython is based on Python 2 and does not have packages necessary for numerical computation that Python 3 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 can communicate with PSLF and send data to PY3 which then handles the data and sends it back to IPY for reinsertion into PSLF. This cycle continues as along as simulation is required. While the AMQP solution has been shown to work, message handling accounts for about one half of all simulation time.

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. Additionally, 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 by consulting PSLF documentation and tutorials.

2.3.2.2 Simulation Parameter Input (.py)

Simulation parameter input is entered in a standard python `.py` file. Most input is collected in a Python dictionary named simParams. An example of the simParams dictionary defined inside the .py file is shown in Figure 2.4.

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 integration method, inclusion of frequency effects, and how system inertia is calculated. Information related to data export is also included in the simParams dictionary such as where the data will be placed and what it will be named.

```
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', # 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.
        'deleteInit' : 0, # Delete initialized mirror
        'assumedV' : 'Vsched', # assummed voltage - either Vsched or Vinit
        'logBranch' : True,
}
```

Figure 2.4: An example of a simParams dictionary.

In addition to the simParams dictionary, simulation notes, absolute file paths to the

desired `.sav` and `.ltd.py` file are also defined in this 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.

2.3.2.2.1 SimParams Dictionary

The `simParams` dictionary is used to set various required simulation parameters such as time step, end time, data output location, and integration method. 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.

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

The required `.ltd.py` file is used for inputting user defined perturbances and balancing authority settings. Furthermore, this file is the ideal place for adding additional user input if the need should arise in future software development. A description of how to add perturbances is presented in the following section. Balancing authority options are more complex and relate more to the engineering problem so only a light introduction is presented in this section.

2.3.2.3.1 Perturbance List

A list of single quoted strings describing system perturbances is defined in the `.ltd.py` file as *mirror.sysPerturbances*. Most 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 Descripction'. 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

Table 2.3: Perturbance Agent Identification options.

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

perturbance agent. 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. Figure 2.5 shows various examples

Table 2.4: Perturbance action options.

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

of valid perturbance agent definitions.

```
# 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 power of load up 400MW over 40 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 2.5: Perturbance agent examples.

As shown in Figure 2.5, 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 governred generator, both the mechanical power and power reference variables should be changed as shown in line 8 and 9 of Figure 2.5.

2.3.2.3.2 Balancing Authority Dictionary

A dictionary *mirror.sysBA* is defined in the `.ltd.py` file that sets all BA actions and parameters. Each individual BA is a nested dictionary inside the *mirror.sysBA* dictionary. The information entered in each nested dictionary describes how that particular BA calculates and its ACE. A description of each possible field is described in Appendix A Table A.1. Additional information on more in depth options is presented in Section 3.1.

2.3.2.3.3 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 *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 2.6.

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 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 2.6 was used in a simulation, a ramp would be created for each load in area 2 that increases load by 6.474 % between time 10 and 20. The relative percent is based off the value of each individual load at time 10. This method

```
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 2.6: Load control agent dictionary definition example.

of relative percent changing allows for other perturbances, such as noise, to be applied to the same load without issue. Alternative ramp types may be employed by changing the 'rampType' parameter but are untested 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.

2.3.2.3.4 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'. 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. Forcast data is again collected from the EIA website and parsed in the same manner as demand data. Figure 2.7 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 2.7: 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 2.7, 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 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.

2.3.2.3.5 Governor Delay Dictionary

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 2.8 shows an example of a valid delay dictionary.

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

Figure 2.8: Governor delay dictionary definition example.

Tuples are used to enter delay block parameters. Using Figure 2.1 as reference, the `wDelay` tuple contains settings for D_A and T_A respectively. Likewise, the `PrefDelay` tuple contains D_B and T_B. The block may be configured to use only the delay or the filtering without error. Additionally, while `genId` is optional, if set to None the first found generator on the specified bus is used.

2.3.2.3.6 Governor Deadband Dictionary

While a BA agent may be able to set area wide deadbands, it is also possible to specify a single deadband for any governed generator. Settings in the governor deadband dictionary will override any deadband settings specified by the BA dictionary. Figure 2.9 shows three examples of valid governor deadband definitions.

```
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 defined governor deadbands
}# end of govDelay dictionary
```

Figure 2.9: Governor deadband dictionary definition example.

2.3.2.3.7 Definite Time Controller Dictionary

During long simulations, system loading may change from initial values by ±20%. Such changes can cause voltage issues that require the setting or un-setting of reactive components contributing to MVAR supply. This can be accomplished by defining a definite time

controller (DTC) agent in the *mirror.DTCdict* dictionary. Figure 2.10 shows an examples 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 2.10: 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

System initialization begins 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 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 starts by also importing required packages and setting references to certain imported packages and PSLF 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 even conceived. IPY uses the received initialization information to load the GE Python API which is then used to load the .sav and .dyd into PSLF.

Once the PSLF specific files are loaded into the GE software, initialization of the Python model begins. 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. This idea of a single system object with a recursive data structure 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, the linking technique eliminated a need for global variables outside of imported packages and allowed for a single file containing **all** simulation data to be easily exported at the end of a simulation.

The mirror itself is .... a big deal of stuff that hasn't been completely explained in a written form yet, but soon, right?

2.3.4 Simulation Loop

Once simulation initialization is complete, and the system mirror is initialized, the simulation loop is executed. Figure 2.11 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 systems 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 condition.

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 is run. 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.

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 LTD mirror. The electric power output of the system is summed for use in calculating system accelerating power in the next time step. Any LTD logging agents

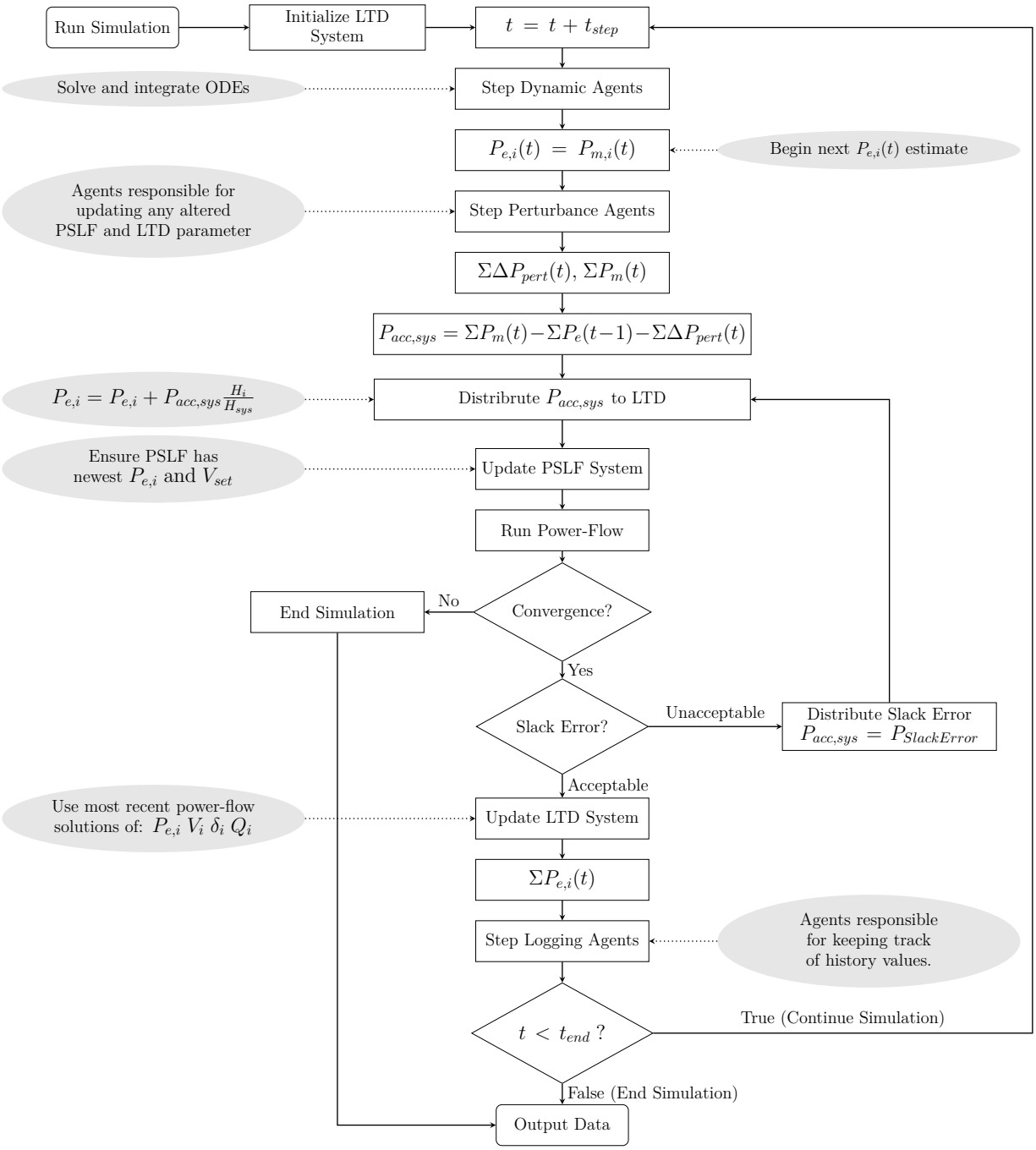

Figure 2.11: Flowchart showing overview of simulation time step actions.

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, 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 the python mirror data. The Python plot functions are located in the PSLTDSim package `plot` folder, while the MATLAB validation plots are location in the GitHub repository only.

3 Engineering Problem

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 Controls

The simulated controls used to explore the engineering problem of interest mainly deal with area wide settings. Specifically, area wide governor deadbands and droops, as well as AGC settings. A complete list of BA agent options is shown in Table A.1 in Appendix A and a detailed description of the more involved control is described in the following sections.

3.1.1 Governor Deadbands

The FERC maximum deadband is 36 mHz[10]. However, the execution of a governor deadband is not explicitly detailed and left to generator operators to configure. PSLTDSim offers a `deadBandAgent` that can apply various deadbands to the incoming $\Delta\omega$. Figure 3.1 graphically depicts how the various deadband options differ.

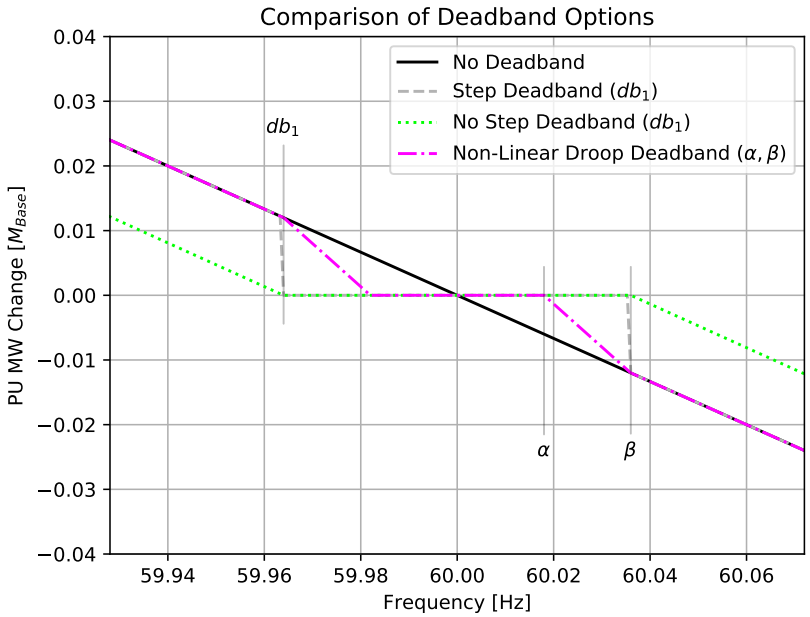

Figure 3.1: Examples of available deadband action.

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 and crosses the original droop line at the specified droop. For instance, if a droop is 5%, then a 5% deviation in frequency would request a 100% change in output power. While the no-step deadband does deliver a 100% increase at the given droop, the actual response will always be below the assumed droop response.

To eliminate this problem, a non-linear droop option was created. The non-linear droop requires two inputs. The first input,(α), specifies where the new droop starts, and the second input, (β), is when the response will return to the original droop setting.

Choosing and configuring a deadband is done via values in the BA parameter dictionary (*sysBA*) or the governor deadband dictionary (*govDeadBand*) in the `.ltd.py` file.

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.

3.1.2 Governor Input Delay and Filtering

The inputs to the modeled governors may be delayed and filtered using the Laplace domain block. Delay must be divisble by the timestep.

3.1.3 Area Wide Governor Droops

The typical FERC droop is 5%[10]. Area wide governor droops can be specified in the BA parameter dictionary that over write the droop setting read from a `.dyd`. All active governors in an area will use the specified `'AreaDroop'` value. This setting allows for fast and easy configuration of simulations aimed at exploring droop settings.

3.1.4 Automatic Generation Control

The general workings of AGC is shown in Figure 3.2. Simulation settings related to frequency bias, ACE integrating and filtering, and conditional and weighted summing are

explained in the following sections.

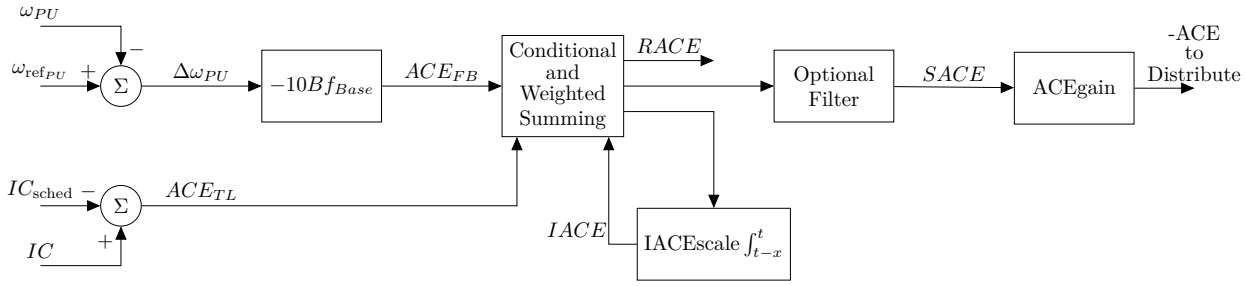

Figure 3.2: Block diagram of ACE calculation and manipulation.

3.1.4.1 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 BA parameter 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 `peramx` 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.

3.1.4.2 Integral of Area Control Error

As previously shown in Figure 3.2, ACE may be integrated and fed back into the weighted and conditional summing block. Settings related to this process are configured in the BA parameter dictionary. Settings related to the integral of ACE (IACE) are `'IACEwindow`, `'IACEscale`, and `'IACEdeadband`. As expected, the IACEwindow defines the length in seconds of the moving window integrator. If IACEwindow is set to zero, integration will be continuous. IACEscale acts as a gain of the output integral value. IACEdeadband specifies the frequency deviation in Hz below which integration values will stop being added back into the conditional summing.

3.1.4.3 Weighted and 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 3 types of conditional ACE calculation. All conditionals involve checking the sign of the calculated value to the sign of frequency deviation. Table 3.1 shows the various conditional summations.

Table 3.1: Tie-Line Bias AGC type ACE calculations.

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

If IACE is enabled, the value it calculates is summed with the previously calculated ACE in a weighted fashion. The particulars of which have yet to be hammered out...

3.1.4.4 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 PI. The selection and configuration of the 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. T The value passed in with an integrator filter is simply a gain. The first PI value is used as a proportional gain value and the second value describes the ratio between integral and proportional gain.

3.1.4.5 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 which dictates how much of the ACE signal is sent to each. The check done to ensure each BA

has a total participation factor of one will only issue a warning. 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.

4 Future Work

- Move away from reliance on GE software.

  This would require:

  - new system definitions and dynamic model input methods
  - new power-flow solver
  - integration into current mirror creation, 'solving system' update methods.

- Further refinement of definite time controller for voltage stability scenarios.
- Document altering effective system inertia for simulating loss of conventional generation scenarios.
- Addition of under load tap changing transformers
- Addition of exponential loads
- More dynamic models
- Improved dynamic model casting methods
- Variable frequency bias control (DTC controls gov pref according to $\Delta\omega$)
- Voltage scheduler Agent
- More defined Plant Controller Agent (sum and log P and Q gen)
- More efficient (performant) code

5 Bibliography

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

[2] 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.

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

[4] 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.

[5] 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.

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

[7] S. developers. (2019). About scipy, [Online]. Available: https://www.scipy.org/about.html.

[8] (2017). Ercot-internconnection_branded.jpg, ERCOT, [Online]. Available: http://www.ercot.com/news/mediakit/maps.

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

[10] 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.

[11] P. S. Foundataion. (2019). About python, [Online]. Available: https://www.python.org/about/.

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

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

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

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

[16] 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.

[17] (2020). Glossary of terms used in nerc reliability standards, NERC, [Online]. Available: `https://www.nerc.com/files/glossary_of_terms.pdf`.

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

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

[20] 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.

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

[22] 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`.

[23] (2019). July2019map.png, U.S. Energy Information Administration, [Online]. Available: `https://www.eia.gov/electricity/data/eia860m/`.

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

[25] 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.

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

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

[28] 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.

[29] (2004). Montana electric transmission grid: Operation, congestion, and issues, DEQ, [Online]. Available: https://leg.mt.gov/content/publications/Environmental/2004deq_energy_report/transmission.pdf.

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

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

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

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

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

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

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

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

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

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

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

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

[42] 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.

[43] 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.

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

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

[46] (2017). Thread in operating system, GeeksforGeeks, [Online]. Available: `https://www.geeksforgeeks.org/thread-in-operating-system/`.

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

[48] (2019). U.s. electric system operating data, U.S. Energy Information Administration, [Online]. Available: `https://www.eia.gov/realtime_grid/`.

[49] (2019). U.s. energy mapping system, U.S. Energy Information Administration, [Online]. Available: `https://www.eia.gov/state/maps.php?v=Electricity`.

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

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, ACE and IACe all 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.
IACEweight	Float	-	0.5	Weighting of IACE to ACE used during summation.
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.

```python
# 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.