LONG-TERM DYNAMIC SIMULATION OF POWER SYSTEMS USING PYTHON, AGENT BASED MODELING, AND TIME-SEQUENCED POWER FLOWS

by
Thad Haines

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science in Electrical Engineering

Montana Technological University
2020

Abstract

Abstract will go here once thesis is finished.

Keywords: Power system simulation, Time-sequenced power flow, Long-term dynamics, Automatic generator control, AGC, Agent based modeling, ABM, Python, IronPython, AMQP

Dedication

Forthcoming...

Acknowledgments

Forthcoming...

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
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
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
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
RACE	Reported ACE
SACE	Smoothed ACE
SS	Steady State

Term	Definition
TLB	Tie-Line Bias
TSPF	Time-Sequenced Power Flow
WECC	Western Electricity Coordinating Council

1. Introduction

The increasing demand for electric power has forced power systems to evolve. From the earliest power generating stations designed to meet only local needs, to many remote units serving a wide area of distributed customers, power systems continuously get larger and more complicated. As most things do, these changes came about quiet naturally. To meet increasing electric demand, more generating stations were built and connected together. Then, as areas of demand became more dispersed, the use of high voltage transmission lines were used to link nearby regional power systems together.

The gradual connection of many remote power systems not only allowed for a pooling of resources, but with proper control, could also increase system stability and reliability. However, the independent development of local power systems led to a non-uniform and complex network that requires detailed control. To further muddle matters, the aging North American grid is often pushed to its operating limits. Adding more or updating infrastructure to relieve system stress may be a long and difficult process due to permitting or financial reasons. Additions that are approved must be properly planned to not only handle a variety of environmental, economic, and social concerns, but to also contribute long-term benefit to power system operation.

Fortunately, we live in the future and can use computers to design, simulate, monitor, and control the Bulk Electric System (BES) in ways, and at speeds, never before possible.

. . .

This is where the computer simulation to engineering problem link is introduced and backed up.

The engineering goal of this work is to simulate frequency and voltage controls to solve a ‘long-term’ engineering problem, such as. . . A long-term dynamic simulation framework based on a time-sequenced power technique is believed to be capable of accomplishing such a task. Furthermore, the resulting open-source software may be expanded upon to allow future research.

Since such a simulation tool does not yet exist, to accomplish the desired engineering goal, a time-sequenced power flow simulation program must be created and validated. Certain elements such as system model format and power-flow solver can be reused from previously ex-

isting software solutions, but features like automatic generation control, dynamic governor models, and plant controllers must be recreated. However, due to time limitations and engineering intuition, several modeling simplifications and assumptions are expected be employed.

General code structure must allow for modular expansion, custom procedure insertion, and large system scaling. The software must also be able to run full base case scenarios involving thousands of buses and generators. The finished program must be able to output data that can be validated against industry standard software. Output data should provide enough detail so that analysis of system control validity and efficiency can be performed.

This thesis provides a brief explanation of basic concepts involved with the electrical engineering aspects of this work as well as the computer science, or software, aspects. The created software is then explained and validated before the engineering problem is introduced. Simulation results related to the engineering problem are presented and analyzed. Finally, conclusions are stated and future work suggested.

2. Electrical Engineering Background

2.1. Power System Basics

Before covering the physical structure of power systems, it's worth providing a general introcution to the regulatory and operational structure of the North American bulk electric system (BES), or more commonly 'the grid', which is regulated by multiple tiers of organizations. At the 'top' is the federally empowered Federal Energy Regulatory Commission (FERC) which regulates the interstate transmission of natural gas, oil, and electricity as well as regulating natural gas and hydro electric power projects [1]. The North American Electric Reliability Corporation (NERC), granted it's authority by FERC, aims to assure the effective and efficient reduction of risks to the reliability and security of the BES by developing and enforcing reliability standards and providing education and training to industry personnel [2]. Under the top two there are a collection of regional transmission operators (RTOs), independent service operators (ISOs), balancing authorities (BAs) and utilities. While each identified entity has a specific purpose, this research focuses on BA action, which is to balance supply and demand in certain areas of a power system while also adhering to required NERC operational mandates [50].

Physically, the North American grid is separated into interconnections which can be thought of as relative frequency zones. The three main interconnections in the United States (east, west and Texas) separated from each other by AC-DC-AC ties. These AC-DC-AC ties allow for a separation of frequency between zones because rectifying any AC signal to DC removes any frequency content, then through the inverting process, any frequency and phase may be created. This rectification and inversion allows power to be transported between the relative frequency zones (interconnections) with acceptable losses.

While power system loads are generally placed wherever people think they want to use electricity, generation is located in areas that have convenient resources nearby. This often means generation is built many miles away from large load centers. The use of AC signals, transformers, and high-voltage transmission lines allow remotely generated power to be efficiently sent to distribution systems. These distribution systems then, unsurprisingly, handle the distribution of

electricity to customers.

In a well designed power system, electric demand is always met. However, demand is always changing. Therefore, automatic controls have been developed to better manage certain aspects of the grid. The automatic control types this thesis is concerned with are separated by the time frames in which they operate. The first automatic control, referred to as primary control, responds immediately and operates for a number of seconds after an event or pertubance. Primary control consists of turbine speed governors that respond to deviations in system frequency from the nominal operating condition. Secondary control, which takes place after primary control and acts over multiple minutes, is referred to as automatic generation control (AGC). AGC is a configured by a BA to manage its area control error (ACE) which is a combination of interchange a frequency error. AGC is often referred to as load frequency control (LFC) and the two terms are often used interchangeably in literature. This thesis will use AGC to refer to secondary control.

Simulation of the grid is required for planning of grid expansion as well as testing new operational procedures, such as AGC settings or shunt switching schemes. Additionally, power system simulation can be used to explore events that may not be fully understood or occur rarely in reality.

2.2. Power Flow

The power flow equation is a classic power engineering formula. It is a solution to an array of non-linear equations. Given a system admittance matrix (Y), and certain powers and voltages at every bus, a steady state of the system can be found via iteration. The steady state solution involves bus real and reactive power and voltage magnitude and angle. While there are many methods to solve a power flow, it is beyond the scope of this document. An explanation of the power-flow solution is presented in [20].

To calculate power flowing on a line, the line impedance along with voltage magnitude and angle of two connected buses is required. For example, if a system has any two buses that can be equated to Figure 1, the equations to calculate current, real power, and reactive power flow between them is shown in Equations 1-3 respectively.

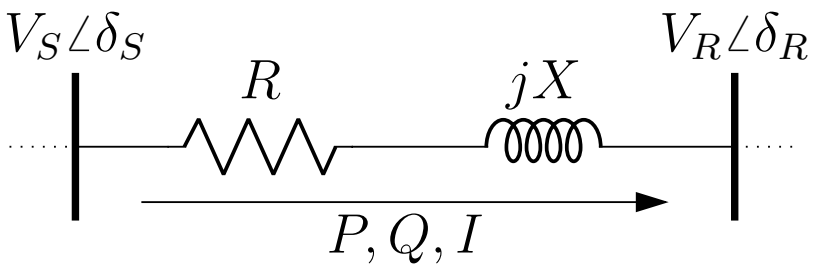

Figure 1: Two buses with power flow between them.

$$I = \frac{V_S e^{j\delta_S} - V_R e^{j\delta_R}}{\sqrt{3}(R + jX)} \tag{1}$$

$$P = \sqrt{3} V_S |I| \cos(\delta_S - \angle I) \tag{2}$$

$$Q = \sqrt{3} V_S |I| \sin(\delta_S - \angle I) \tag{3}$$

It should be noted that the above equations are per-unit (PU) and the square root of three is used to handle per phase and line-to-line value transforms. All equations may be useful because mega Watts (MW) on a line is a common way to describe power flow, but lines are rated in current, and reactive power flow may be used in shunt switching schemes to control bus voltage.

2.3. Swing Equation

The swing equation is the basis of dynamic generator speed and is based on Newton's second law. In the case of electric generators, frequency is directly related to turbine speed and either word is often used to refer to the same thing. The per-unit swing equation, Equation 4, shows the acceleration of generator frequency $\dot{\omega}$ is directly related to the balance between mechanical power P_{mech} supplied by the generator, and electrical power P_{elec} required by the load.

$$\dot{\omega} = \frac{1}{2H}\left(\frac{P_{mech} - P_{elec}}{\omega} - D\Delta\omega\right) \tag{4}$$

Additionally, it can be seen that machine inertia H has an inverse relationship to the magnitude of $\dot{\omega}$. The D term may be used to represent damping forces any time a generator deviates

from synchronous speed [20] however, it is often assumed to be zero. Per-unit speed deviation,

$$\Delta\omega = \omega_{rated} - \omega, \tag{5}$$

is a simple calculation used to scale the damping term. Note that the nature of PU equations dictate that rated speed is equal to one.

2.4. Turbine Speed Governors

Turbine speed governors, often just governors, act to arrest frequency change by adjusting a generator's power reference setting. Governors are classified as primary control because of their typically fast response. While many detailed models of governors exist, almost all include a permanent droop or regulation constant R. The droop constant dictates the resulting change in mechanical power to any deviation in turbine speed. R is often written in percent or a decimal representing a percent. For instance, if a generator has a droop of 0.05, or 5%, then a 5% change in frequency would cause a 100% change in mechanical power output. While R is typically shown as a positive value, the practical use of R, which is to define the slope relation between change in power to change in frequency, is negative. Governor settings are dictated by FERC while NERC offers interconnection suggestions. The most commonn R is 5% for nearly all types of generators[39].

2.5. Automatic Generation Control

Automatic generation control (AGC), also referred to as load frequency control (LFC), is a form of secondary control that works to correct system frequency and area interchange error. AGC does this by calculating an area control error (ACE) that is then distributed to generation units under AGC control. The received AGC signals are then used to vary specified generators mechanical reference point. How AGC is actually distributed varies according to operator discretion so that various operating or economic requirements are met. NERC requires ACE to be reported in a specific way [19]. As shown in Equation 6, reporting ACE (RACE) is the difference between actual and scheduled net interchange (NI) and scaled frequency deviation. The I_{ME} and

I_{ATEC} are interchange meter error and area time error correction respectively.

$$RACE = (NI_A - NI_S) - 10B(F_A - F_S) - I_{ME} + I_{ATEC} \tag{6}$$

By convention, positive $RACE$ indicates a general over generation condition and negative $RACE$ is representative of an under generation situation. More detailed AGC action is covered in more in Section 5.3.4.

2.6. Reactive Power and Voltage Control

While AGC is concerned with frequency and area interchange control, system voltage must also be controlled. To maintain a reliable power system, bus voltage must be held at, or relatively near, a known scheduled value so that various system components operate in a predictable and safe way. Depending on loading conditions and generator VAR output, bus voltages can vary from these known acceptable values. Unchecked, this phenomena may lead to voltage collapse caused by a lack of available reactive power in a system.

Voltage, unlike system frequency, is a local issue and must be resolved locally by the addition of capacitive VARs for low voltage situations, and a removal of capacitive VARS for high voltage situations. Alternatively, Inductive VARs may be used in the opposite manner. Due to the nature of some renewable energies, areas with large amounts of renewable generation may not be able to produce adequate VARs. Thus, to maintain a scheduled voltage, shunt control becomes an important consideration in automatic control configurations.

3. Software Background

3.1. 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. 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.

3.2. Python

Python is an interpreted high-level general purpose programming language that utilizes object oriented techniques and emphasizes code readability [28]. Guido Van Rossum first implemented a version of Python in December of 1989 [43]. Python is freely available and distributable for multiple computing platforms [13].

3.2.1. 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# [14] 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 and only use Python packages compatible

with 2.X. Python 3 (PY3) is 'the future of Python' and has many more useful community created packages. While some packages area available for both Python version 2 and 3, many are not. As of this writing, the current version of Python is 3.8 though most code was written using 3.7.

3.2.2. 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. 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 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.

Non basic data is passed by reference.

3.2.3. 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 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 [9]. 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 proto-

col (AMQP) used for AMQP communication in this project was Pika [15].

3.3. 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 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.

3.4. 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 [41]. This ideology 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. The ABM coding style lends itself towards modular coding and natural expandability.

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 instance, a timer agents step 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. While this action is generic, it is meant to be 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. These actions can be simplified into a sequence of agent steps and then repeated ad infinitum. The simple 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 somehow reference other agents inside the environment.

4. Software Tool

This chapter describes what time-sequenced power flows are and what the simulation tool assumes and simplifies. The software is also explained and validated.

4.1. 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 the course 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.

4.2. 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.

4.2.1. 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 in-

corporate such modeling should the need arise.

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

4.2.2. 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.

4.2.3. 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. 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

$$H_{sys} = \sum_{i=1}^{N} H_{PU,i} M_{base,i}. \tag{7}$$

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

$$P_{acc,sys} = \sum_{i=1}^{N} P_{m,i} - \sum_{i=1}^{N} P_{e,i} - \sum \Delta P_{pert}, \tag{8}$$

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 9, 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}.

$$\dot{\omega}_{sys} = \frac{1}{2H_{sys}} \left(\frac{P_{acc,sys}}{\omega_{sys}} - D_{sys}\Delta\omega_{sys} \right) \tag{9}$$

For completeness, a damping term $D_{sys}\Delta\omega$ is included in Equation 9, but as Equation 4, D_{sys} is often set to zero while $\Delta\omega_{sys}$ calculated using Equation 5 with ω_{sys} as ω.

4.2.4. 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 10 shows how the next electric power estimate is created for each generator i.

$$P_{e,EST,i} = P_{e,i} - P_{acc,sys} \left(\frac{H_i}{H_{sys}} \right) \tag{10}$$

Once all accelerating power is distributed, the new $P_{e,EST}$ value for each generator is used to solve 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 10 until slack tolerance is met, or a maximum number of iterations take place.

4.2.5. 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. Blocks with a * next to them indicate they are optional, and only inserted into the model if user defined.

A slightly more generic governor was created based off the governor model used in Power

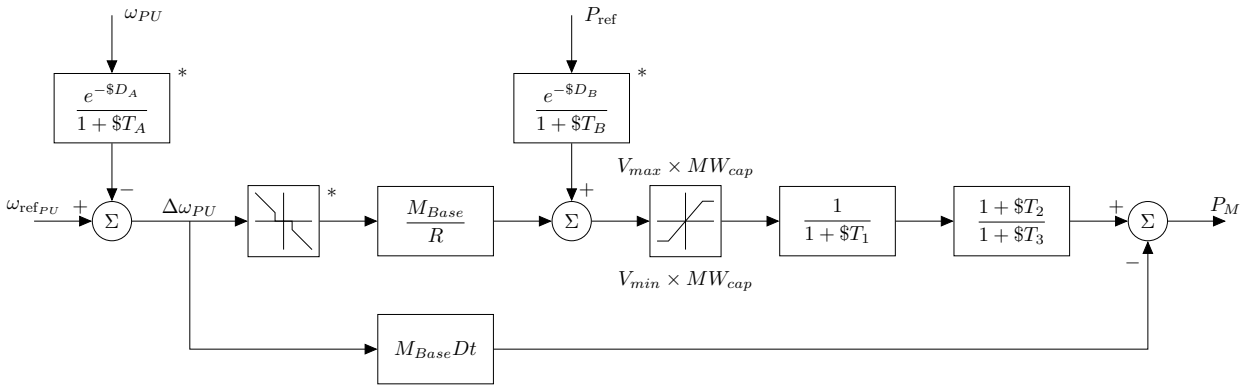

Figure 2: Block diagram of modified tgov1 model.

System Toolbox (PST). This generic model, referred to as *genericGov*, is shown in Figure 3. The genericGov follows the time constant naming convention of the PST governor and includes the same optional blocks added to the tgov1 model.

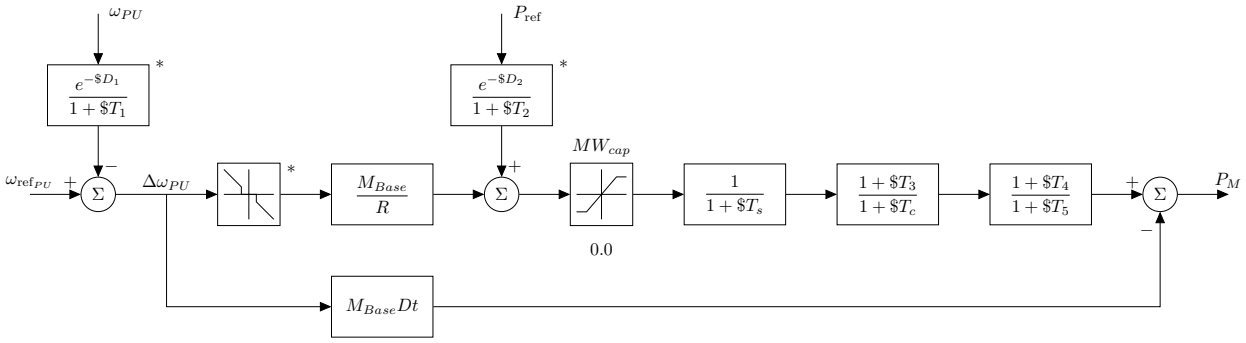

Figure 3: Block diagram of genericGov model.

4.2.5.1. Casting Process for genericGov

Since 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. Universal governor settings, such as permanent droop and MW cap values collected are 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 meant to represent. Table I shows the relation of PSDS model types to assumed governor setting type, and Table II lists the time constants used for each genericGov model type.

Table I: 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 II: 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

4.3. 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 4. 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.

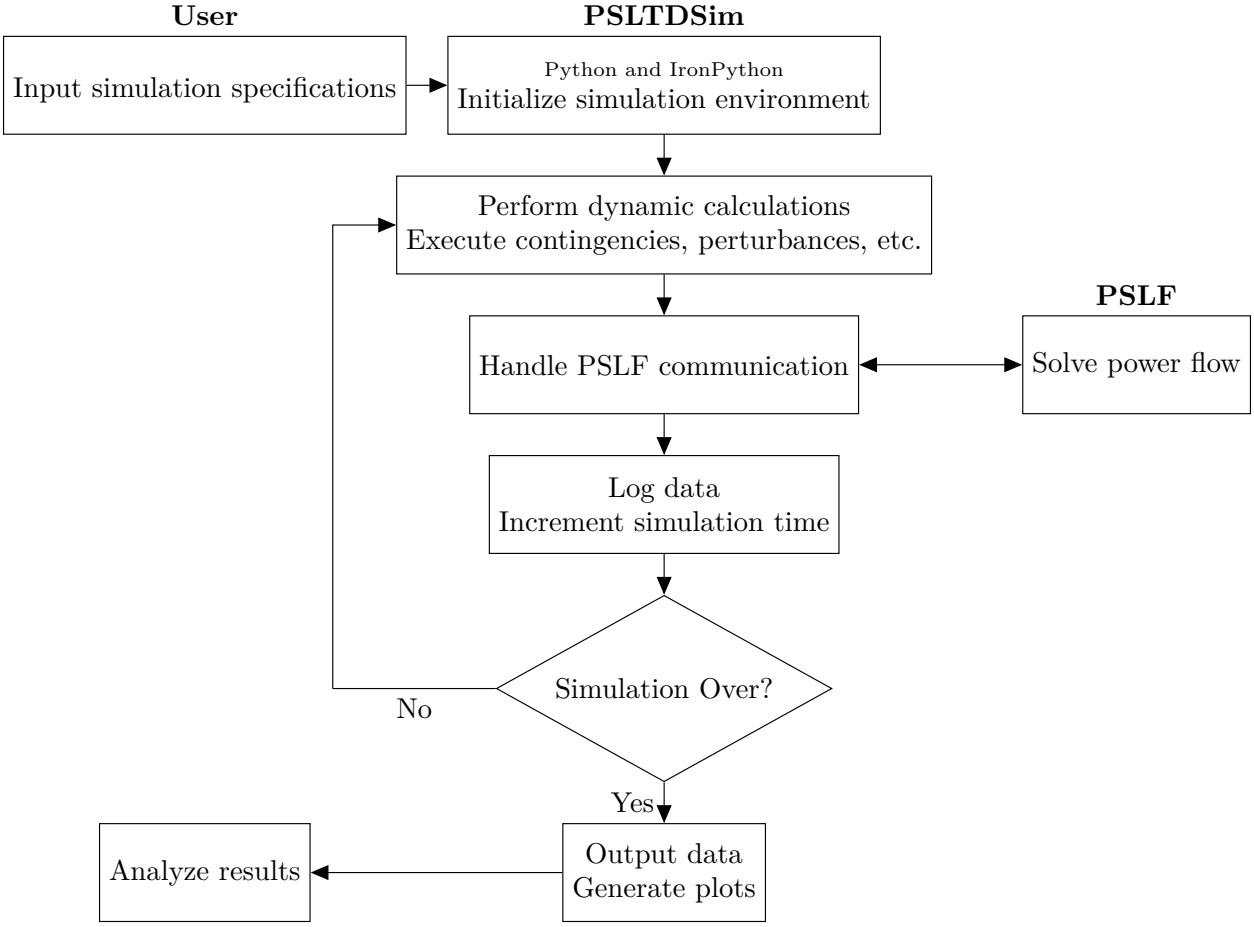

Figure 4: High level software flow chart.

4.3.1. 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 [48]. 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 values in PY3 must be sent to PSLF via IPY, and after each new power-flow solution, PSLF values must be sent to PY3 the same way. 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 accounts for about one half of all simulation time.

4.3.2. 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.

4.3.2.1. 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.

4.3.2.2. 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 5.

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 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.

```
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.
        'deleteInit' : 0, # Delete initialized mirror
        'logBranch' : True,
}
```

Figure 5: An example of a simParams dictionary.

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

sired `.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 10 as Figure 105.

4.3.2.2.1. 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 XIII located in Appendix 11.

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

The required `.ltd.py` file contains user defined objects related specifically to simulated events and control action. Furthermore, this file is the ideal place for adding additional user input if the need should arise in future software development. A description of the perturbance list and various dictionaries is presented in the following section. Most topics covered are described completely, however, balancing authority options are more complex and relate more to the engineering problem so only a lite introduction is presented in this section.

4.3.2.3.1. 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*. 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 III 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 IV describes the various parameters used to specify the action of the perturbance agent. The

Table III: 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]

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 6 shows various examples of valid perturbance agent definitions.

Table IV: 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

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

Figure 6: Perturbance agent examples.

As shown in Figure 6, 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 6.

4.3.2.3.2. 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 acts on the specified area. Additional information on BA options and action is presented in Section 5.3. Additionally, a description of each possible field is described in Appendix 11 Table XII.

4.3.2.3.3. 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 7.

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 7 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 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

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

begin according to the calculated scaled time schedule. Additionally, loads that are off at system initialization (status 0 at time 0), are ignored.

4.3.2.3.4. 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 8 shows an example of a generation control agent definition.

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

```
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 8: Generation control agent dictionary definition example.

ID : Participation Factor" where ID is optional. If ID is not defined, as shown in Figure 8, 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.

4.3.2.3.5. 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 9 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 9: Governor delay dictionary definition example.

Tuples are used to enter delay block parameters. Using Figure 2 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.

4.3.2.3.6. 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 10 shows three examples of valid governor deadband definitions. Information about deadband action may be found in Subsection 5.3.1.

```
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 10: Governor deadband dictionary definition example.

4.3.2.3.7. 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. Figure 11 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.

Each DTC employs a set and a reset timer and may have a hold timer if hold time is set

```
# 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 11: Definite time controller dictionary definition example.

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.

4.3.3. 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 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.

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. 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, 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 starting 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 querring 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 col-

lected 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 simulation 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 for each machine, and turbine type and permanent droop R from governors. Other information, such as MWcap and MVA base, are 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 7 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 distribution and power-flow solution iterations during simulation loop.

With the IPY model fully initialized, the mirror is saved to disk and an AMQP message is sent to PY3 with the mirror location. After the handoff message is sent to PY3, IPY enters its simulation loop which is described in Section XXX (TBA).

PY3 uses the information received from IPY to load the newly created system mirror to perform PY3 specific initializations such as creating any dynamic agents (governors), calculat-

ing area frequency characteristics and maximum capacities, executing any .ltd code and creating associated agents.

PY3 dynamics are initialized to ensure R is on the correct PU base and any MWcaps from dyd paring 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.

4.3.4. Simulation Loop

Once simulation initialization is complete, and the system mirror is initialized, the simulation loop is executed. Figure 12 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 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

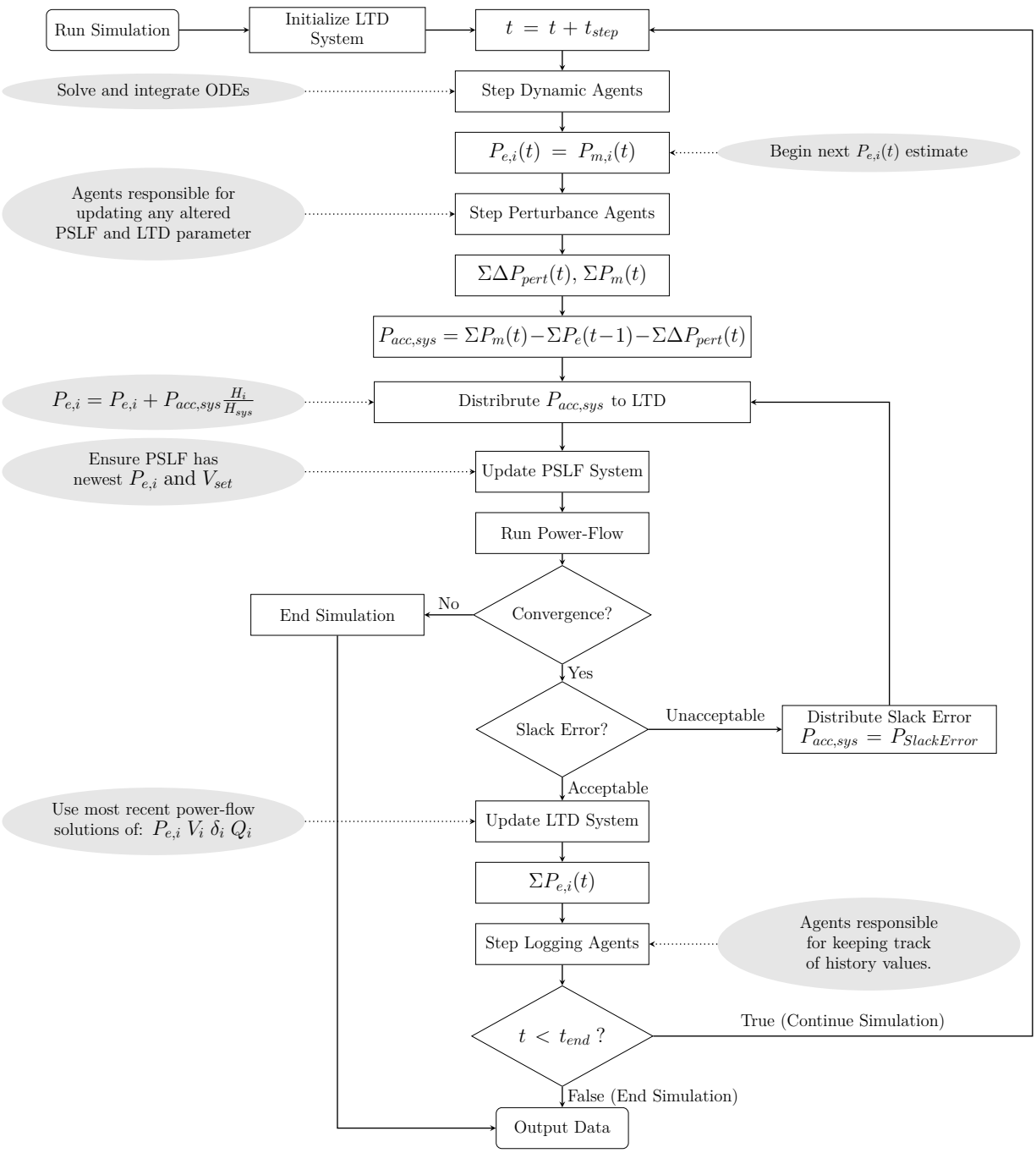

Figure 12: Flowchart showing overview of simulation time step actions.

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 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.

4.3.5. 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.

4.4. Software Validation

PSLTDSim was validated by comparing result data from identical simulation scenarios performed in PSDS and PSLTDSim. Compared values of interest include: system frequency, generator mechanical power, generator real power, bus voltage magnitude, generator voltage angle, generator reactive power, branch current, branch real power flow, and branch reactive power flow. The following sections describe the plots used for validation, present results from various scenarios, and provide a validation summary.

4.4.1. Validation Plots Explained

Plots were used to present simulation validation data. Due to the volume of information, various types of plots were created. Figure 13 shows the three types of plots used for the validation process. Because of the different time steps used in PSDS and LTD, when computing differ-

ence data between the two simulations, the same LTD value was held for multiple comparisons. For example, any PSDS($t = 2.x$) data was compared to the corresponding LTD($t = 2$) data point.

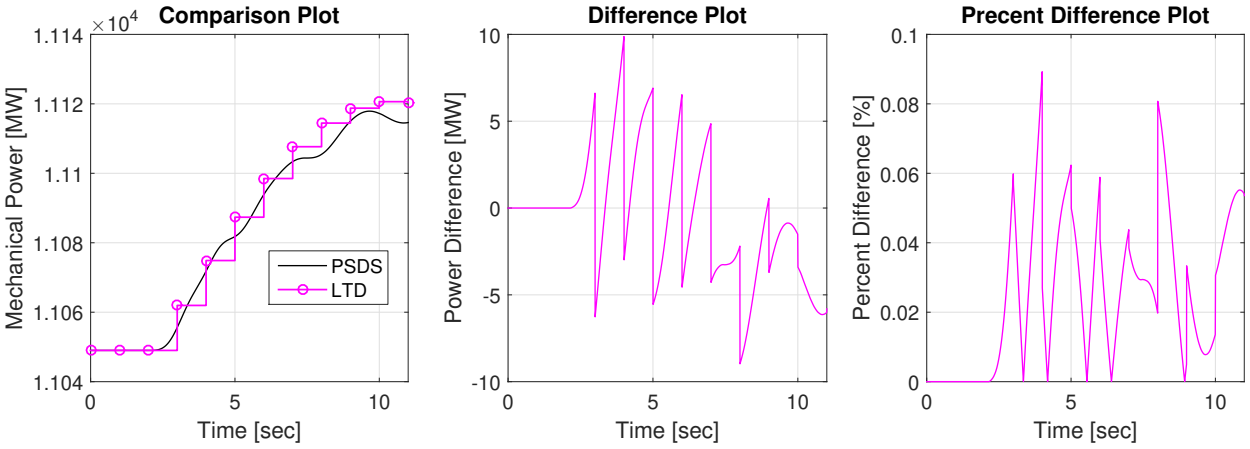

Figure 13: Validation plot examples.

4.4.1.1. Comparison Plot

The most basic plot is a comparison plot that simply puts data from one simulation environment on top of another. An example of a comparison plot is shown in Figure 14 where LTD data is plotted on top of PSDS data.

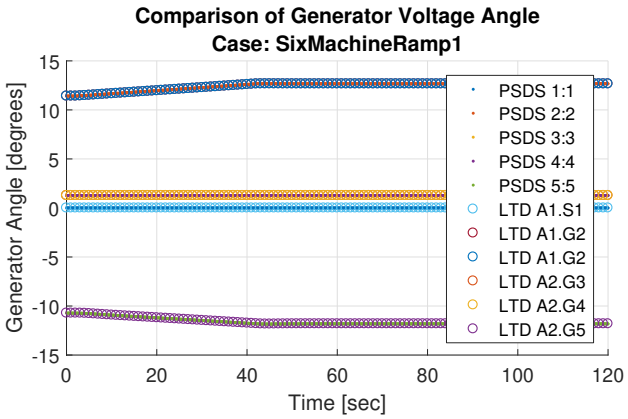

Figure 14: Comparison plot example.

While comparison plots are useful for quick validation, they do not provide quantitative data. Additionally, when comparing many values, plots become difficult to interpret. As a result, comparison plots are used only for showing frequency.

4.4.1.2. Difference Plot

To allow for more easily compared values and provide quantitative data, a difference plot was created. Figure 15 provides an example of a difference plot. Since individual comparisons seemed less important, all data comparisons were plotted in grey with an absolute average plotted in black.

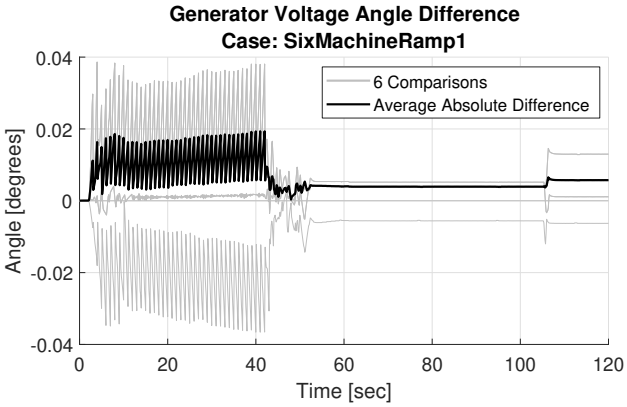

Figure 15: Difference plot example.

Equation 11 describes the difference calculation between PSDS and LTD variables.

$$\text{PSDS}_{data} - \text{LTD}_{data} = \text{Difference}_{data} \tag{11}$$

The absolute average was calculated using Equation 12 where $data_i$ represents a time series of difference data for a particular value of interest and n is the total number of comparisons made.

$$\text{Average}_{abs} = \frac{\sum_i^n |data_i|}{n} \tag{12}$$

While difference plots are more quantitative than comparison plots, there is no sense of scale when comparing two values. As power systems can greatly vary in size, a subjectively insignificant difference in one system may be very significant in another system.

4.4.1.3. Percent Difference Plot

To account for the wide variety of data magnitudes being compared, a percent difference plot was created. Figure 16 shows two examples of a percent difference plot.

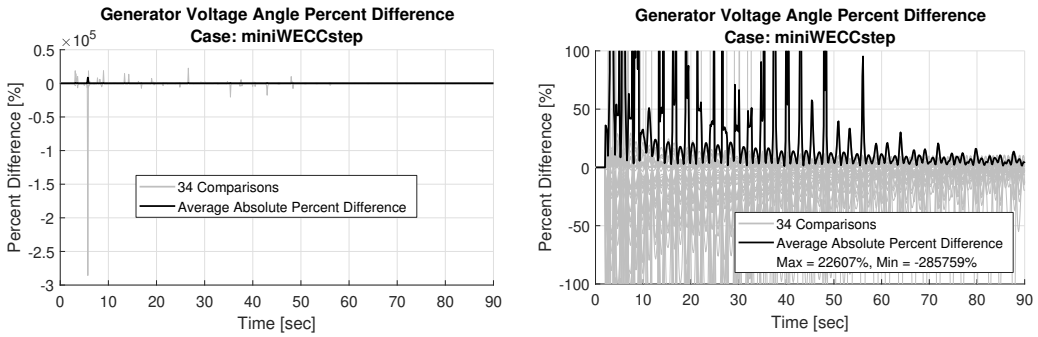

Figure 16: Percent difference plot examples.

Similar to the difference plot, individual comparisons were not deemed as important a cumulative comparison, so the same plotting scheme is repeated. The percent difference was calculated according to Equation 13.

$$\%_{diff} = \frac{|\text{PSDS}_{data} - \text{LTD}_{data}|}{\frac{\text{PSDS}_{data} + \text{LTD}_{data}}{2}} \times 100\% \tag{13}$$

When using percent difference, it should be noted that if one value is positive, and the other negative, results may be misleading. More specifically, as the average of the two numbers being compared approaches zero, so does the denominator of Equation 13. Such a situation may lead to a divide by zero error or, more likely, a very large percent difference. To alleviate such data obfuscation, the y-axis was scaled if the average absolute percent difference was over 150% and the maximum and minimum percent differences are listed in the legend. This is shown in the right hand plot of Figure 16.

4.4.1.4. Weighted Frequency Plot

Frequency comparisons were used to validate the single system frequency assumption calculated by PSLTDSim. Comparison of frequency data from PSDS to LTD was simplified by calculating a single weighted PSDS frequency f_w based on generator inertia. In a system with N generators

$$f_w = \sum_{i=1}^{N} f_i \frac{H_{PU,i} M_{base,i}}{H_{sys}} \tag{14}$$

where f_i is a full time series of machine i's frequency, and H_{sys} is calculated according to Equation 7.

Figure 17 shows all bus frequencies plotted in grey, the calculated weighted frequency in black, and the single system frequency calculated by PSLTDSim in magenta. It is worth pointing out that non-generator bus frequencies are ignored in the weighted system frequency (because a they lack inertia), and that the weighted frequency was used for difference calculations between PSDS and PSLTDSim.

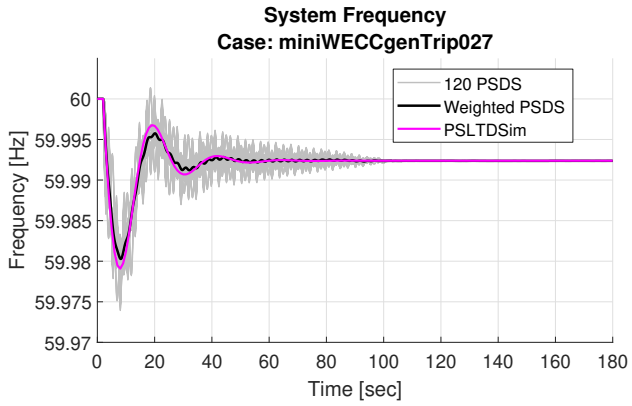

Figure 17: Frequency comparison plot example.

4.4.2. Six Machine System

The six machine system used for validation is shown in Figure 18. The system is based off the Kundur four machine system presented in [29] with a some modifications.

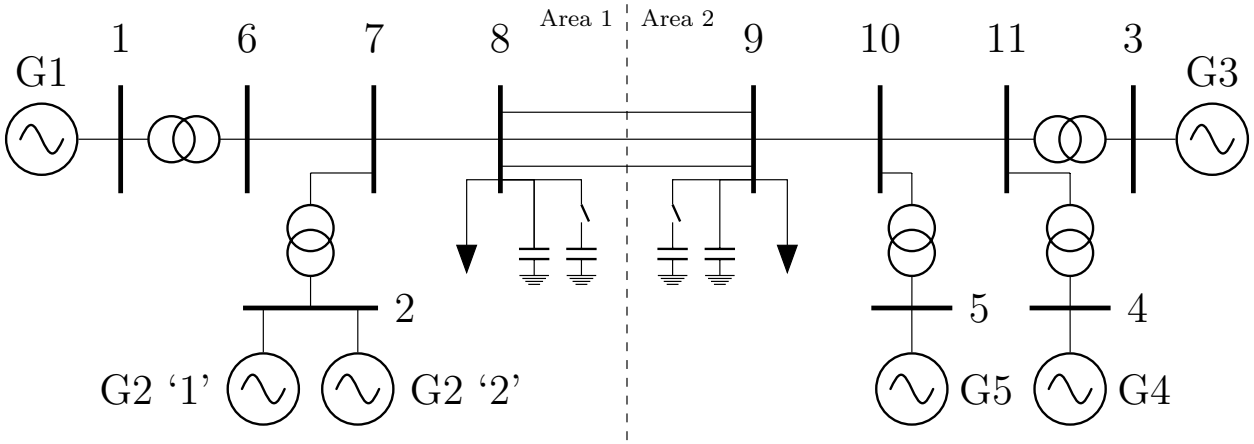

Figure 18: Six machine system.

The most noticeable difference is the addition of two generators and switchable shunts on bus 8 and 9. The generators were added to test power plant style AGC dispatching and multiple generators per bus. While placing multiple generators per bus is not a standard or recommended modeling technique, it is present in the full WECC PSLF model and as such, should be validated in PSLTDSim. Detailed six machine system model specifications are presented in Appendix 9.

4.4.2.1. Simulated Scenario Descriptions

A load step, a load ramp, and a generator trip were simulated using the Six Machine Model. The simplest validation test, a stepping of load, occurred at $t = 2$ when the load on bus 9 was increased by 75 MW. To test longer perturbances, a 40 second 75 MW load ramp of the load on bus 9 was simulated. Due to the LTD one second time step, the ramp is equivalent to forty 1.875 MW steps taking place every second from $t = 2$ to $t = 42$. Finally, to test the handling of inertia, generator 5, which was initially generating 90 MW, was tripped at $t = 2$.

4.4.2.2. Frequency Results

System frequency comparison results are presented for each simulated scenario in Figures 19, 20, and 21. The load step results show a maximum 17 mHz difference that is reduced below 2.5 mHz by $t = 25$. The system frequency during the ramp test never exceeds more than 1.4 mHz with peaks at the start and end of the ramp event. Similar to the load step result, the generator trip event had a maximum difference of 30 mHz that was reduced below 2.5 mHz by $t = 25$. This larger frequency difference is due to the generator trip perturbance being 15 MW larger than the load step perturbance.

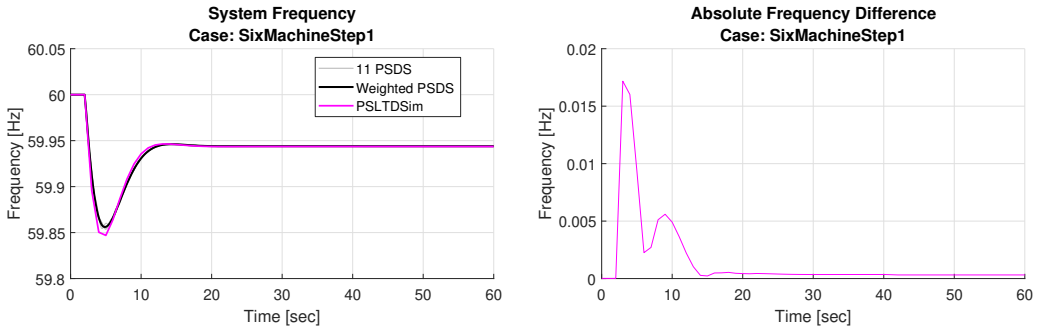

Figure 19: Six machine load step system frequency comparison.

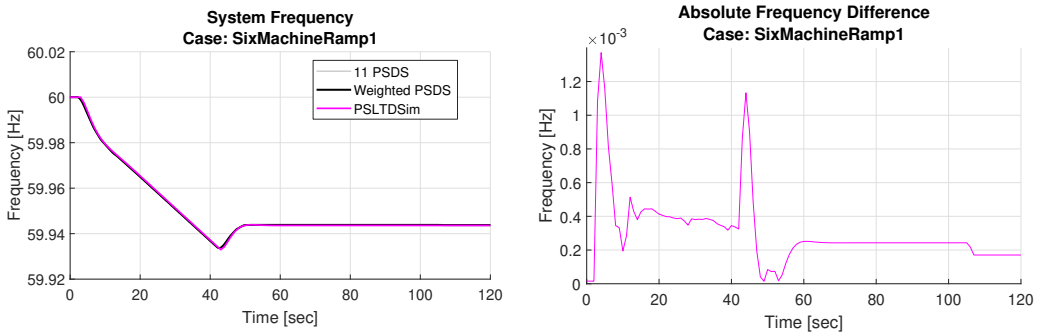

Figure 20: Six machine load ramp system frequency comparison.

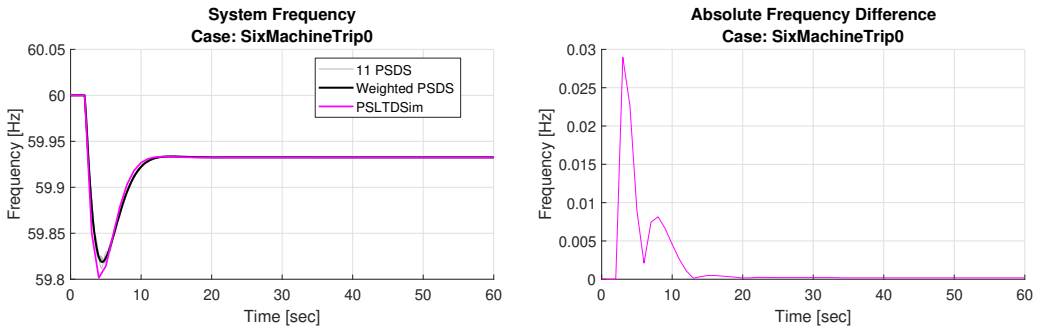

Figure 21: Six machine generator trip system frequency comparison.

4.4.2.3. Generator Mechanical Power Results

Generator mechanical power comparison results are presented for each simulated scenario in Figures 22, 23, and 24. It should be noted that only generators equipped with governors are compared in this section as machines without governors do not have any changes in mechanical power. Similar to frequency results, the ramp event has the smallest difference from PSDS never exceeding 0.5 MW, or 0.2% difference. Unlike the frequency results, most ramp mechanical power differences happen during the ramp with no large peaks at the beginning or end of the event. Further, there is a small steady state error present after the ramp is complete. Both step type events have an immediate peak difference of 5 to 7 MW (2.5 to 3.5 percent difference) that is reduced to roughly zero after 20 seconds.

4.4.2.4. Generator Real Power Results

Generator real power comparison results are presented for each simulated scenario in Figures 25, 26, and 27. All six generators are compared in this section as real power is calculated

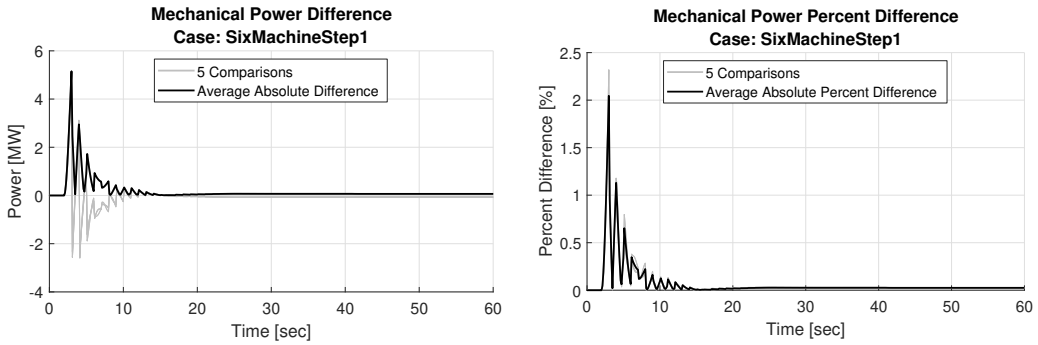

Figure 22: Six machine load step mechanical power comparison.

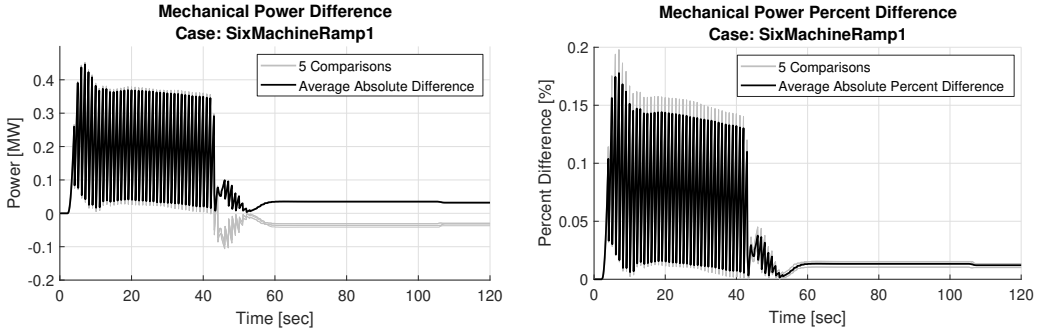

Figure 23: Six machine load ramp mechanical power comparison.

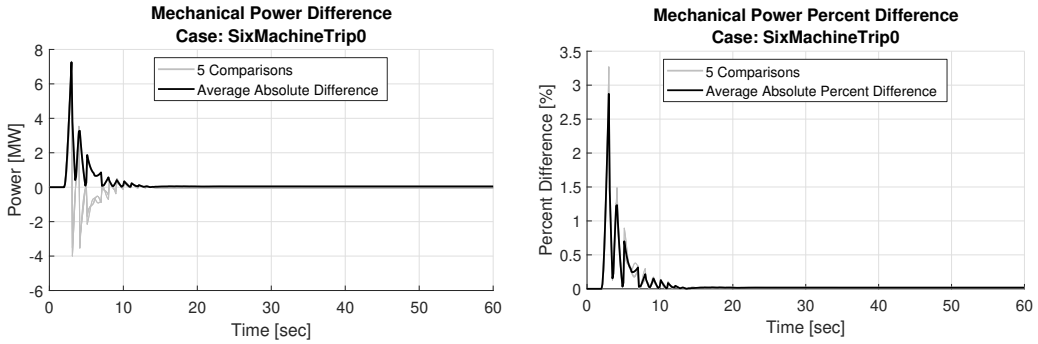

Figure 24: Six machine generator trip mechanical power comparison.

via the power-flow solution and can change every step. Averaged results are very similar to the mechanical power results though slightly more transient. In the step type events, individual difference peaks are nearly double the mechanical power maximums. In all cases, the difference approaches zero approximately 20 seconds after each perturbance.

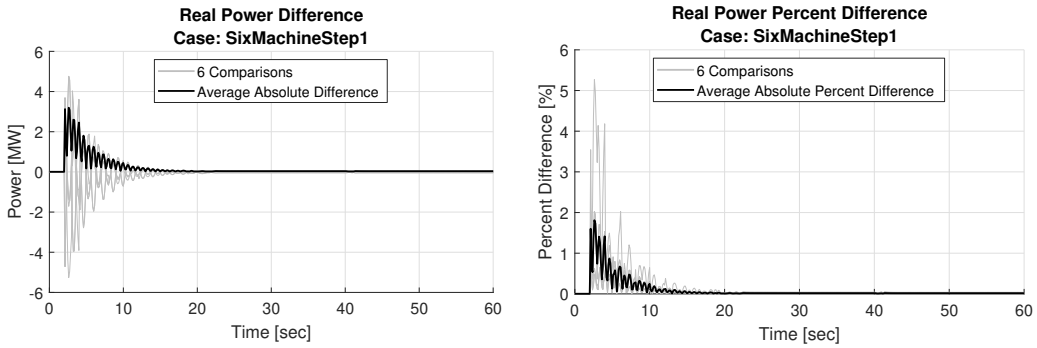

Figure 25: Six machine load step real power comparison.

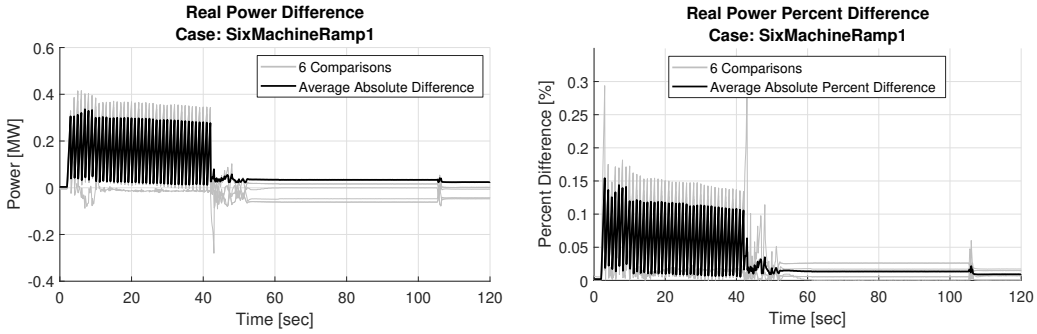

Figure 26: Six machine load ramp real power comparison.

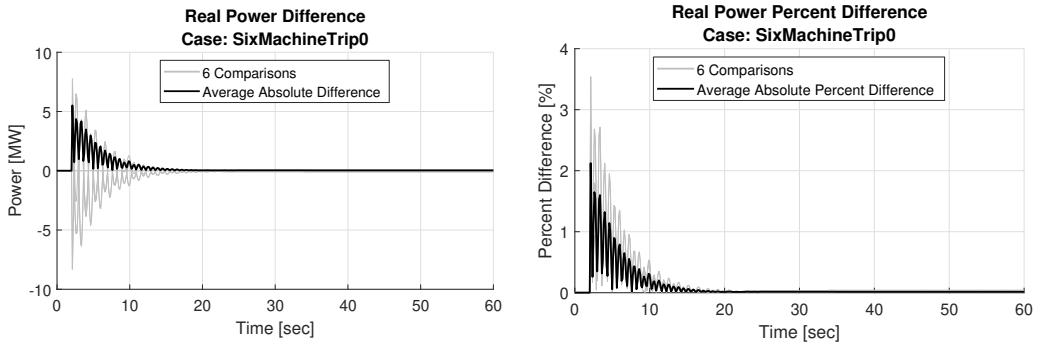

Figure 27: Six machine generator trip real power comparison.

4.4.2.5. Voltage Magnitude Results

Bus voltage magnitude comparison results are presented for each simulated scenario in Figures 28, 29, and 30. Since voltage is presented in PU, percent difference plots

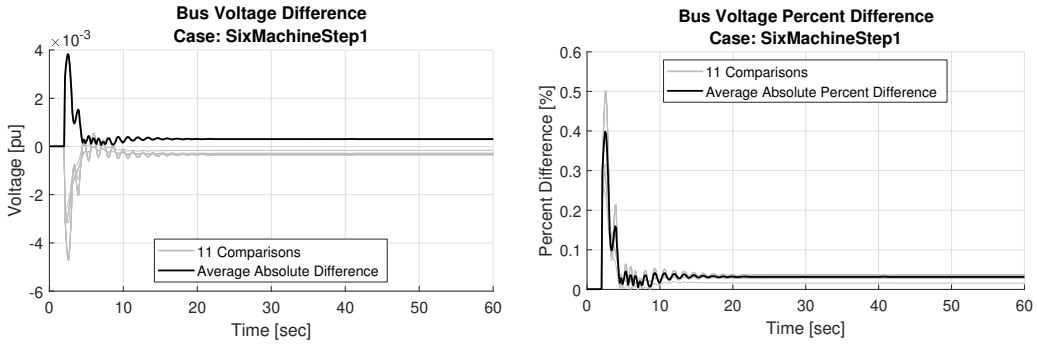

Figure 28: Six machine load step voltage comparison.

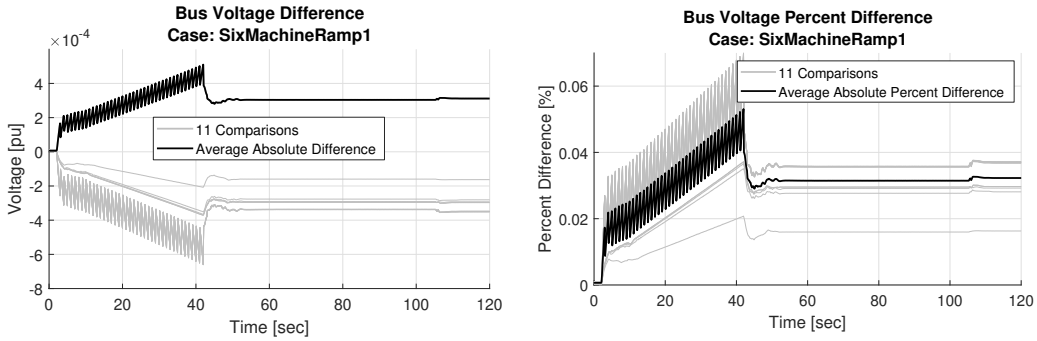

Figure 29: Six machine load ramp voltage comparison.

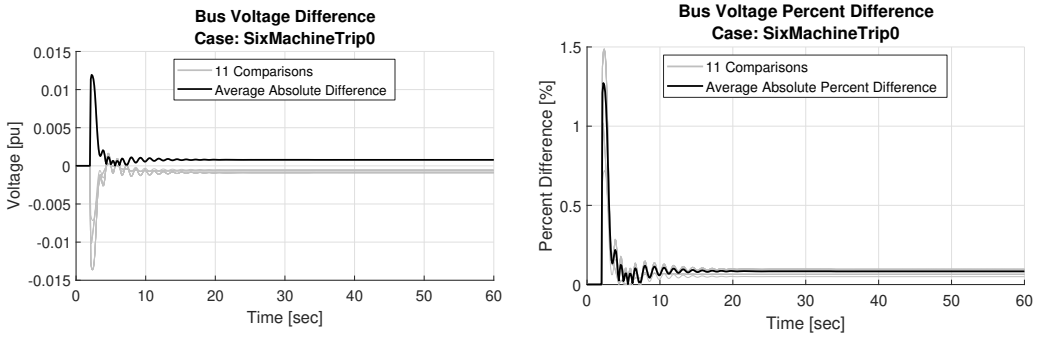

Figure 30: Six machine generator trip voltage comparison.

4.4.2.6. Voltage Angle Results

Bus voltage angle comparison results are presented for each simulated scenario in Figures 31 32 and 33.

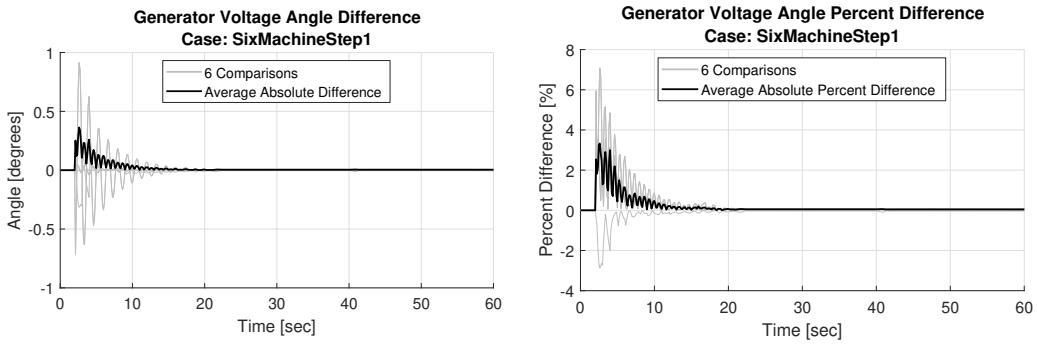

Figure 31: Six machine load step voltage angle comparison.

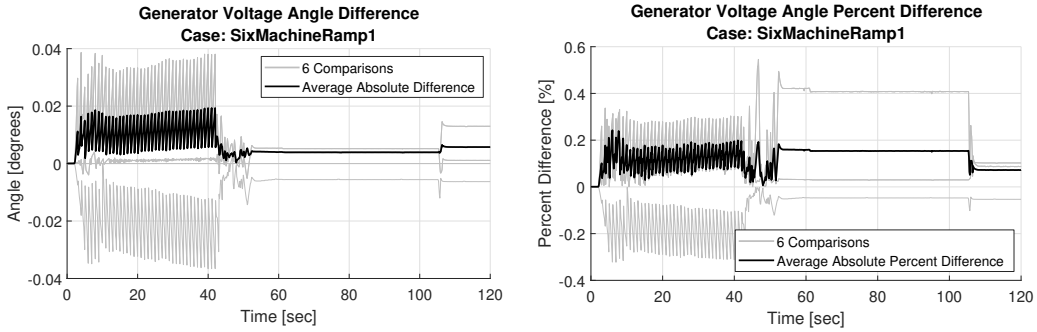

Figure 32: Six machine load ramp voltage angle comparison.

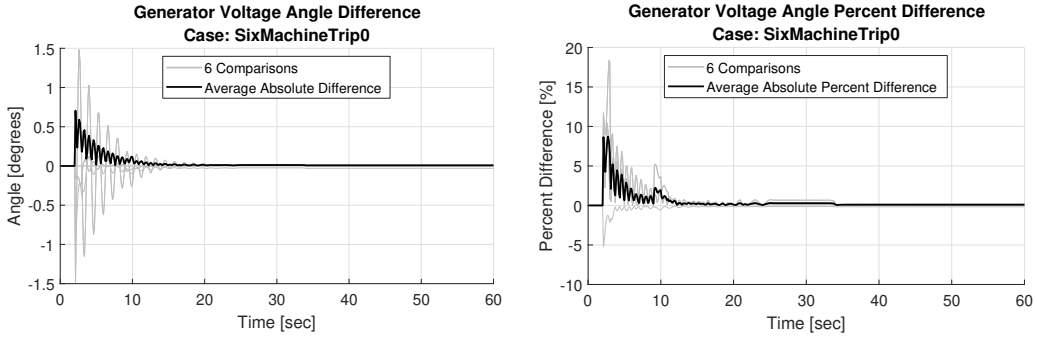

Figure 33: Six machine generator trip voltage angle comparison.

4.4.2.7. Generator Reactive Power Results

Generator reactive power comparison results are presented for each simulated scenario in Figures 34 35 and 36.

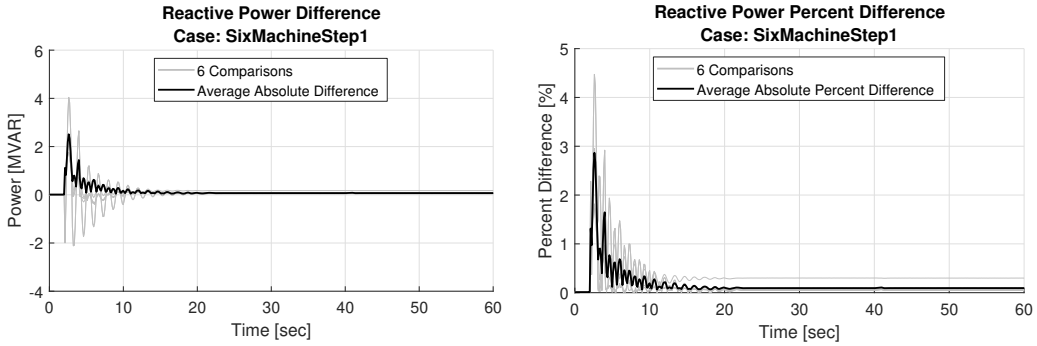

Figure 34: Six machine load step reactive power comparison.

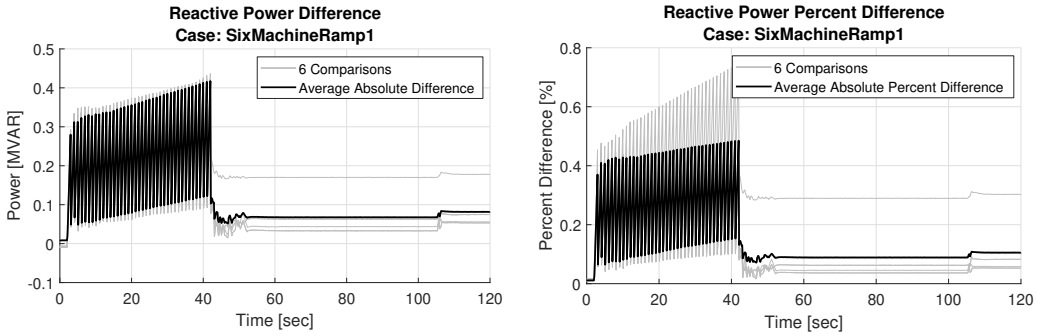

Figure 35: Six machine load ramp reactive power comparison.

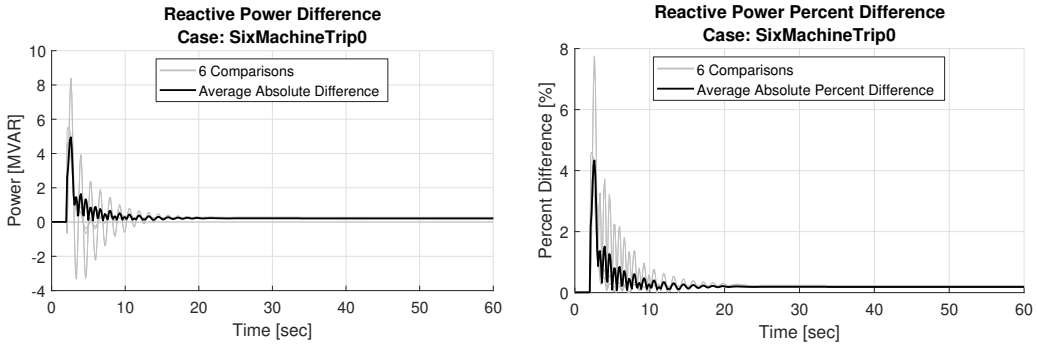

Figure 36: Six machine generator trip reactive power comparison.

4.4.2.8. Branch Current Results

Branch current comparison results are presented for each simulated scenario in Figures 37 38 and 39.

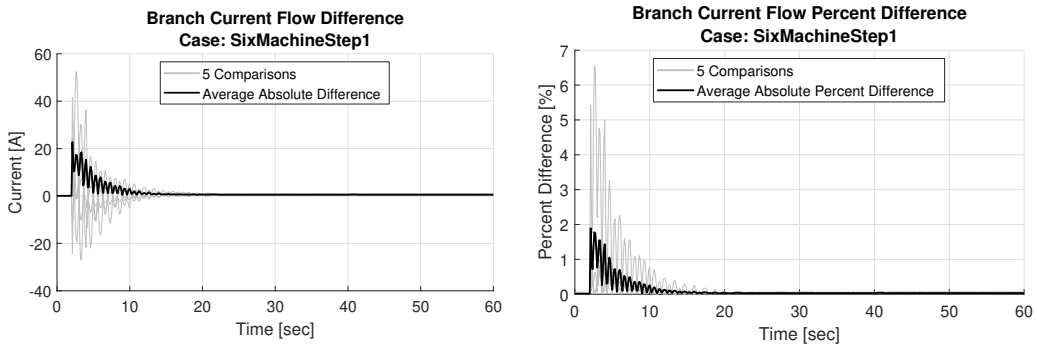

Figure 37: Six machine load step branch current flow comparison.

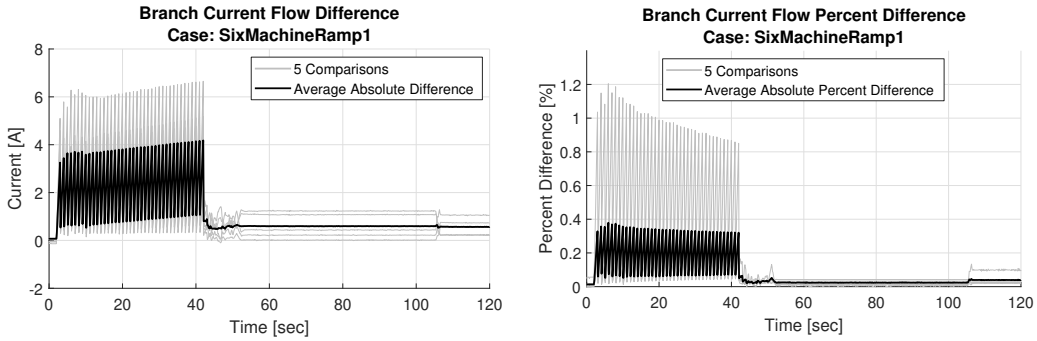

Figure 38: Six machine load ramp branch current flow comparison.

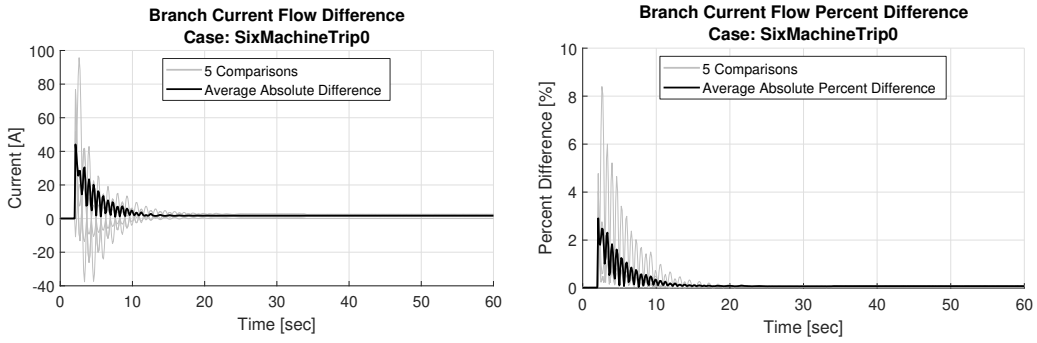

Figure 39: Six machine generator trip branch current flow comparison.

4.4.2.9. Branch Real Power Flow Results

Branch real power flow comparison results are presented for each simulated scenario in Figures 40 41 and 42.

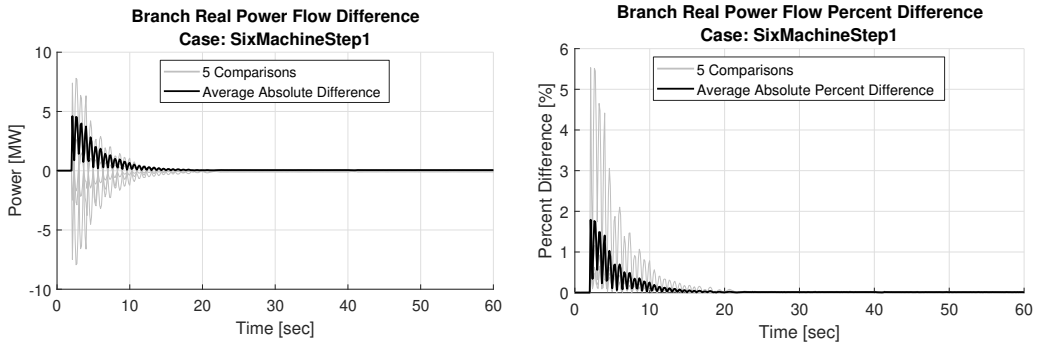

Figure 40: Six machine load step branch real power flow comparison.

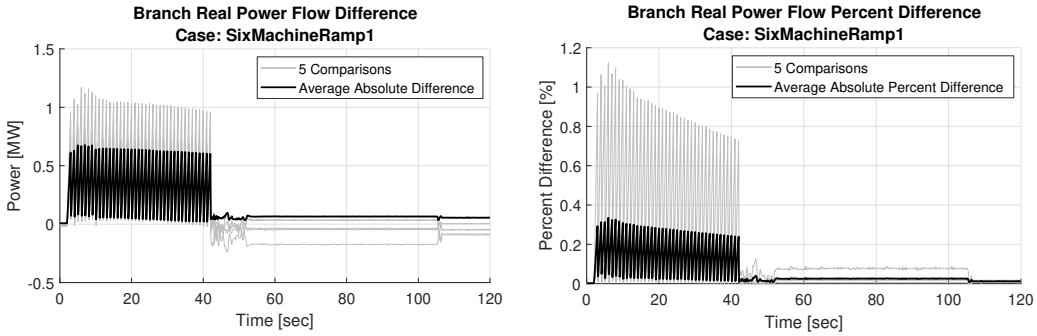

Figure 41: Six machine load ramp branch real power flow comparison.

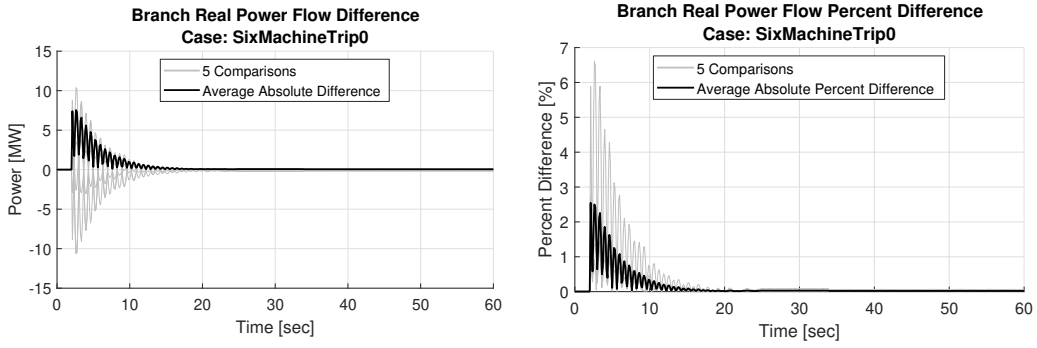

Figure 42: Six machine generator trip branch real power flow comparison.

4.4.2.10. Branch Reactive Power Flow Results

Branch reactive power flow comparison results are presented for each simulated scenario in Figures 43 44 and 45.

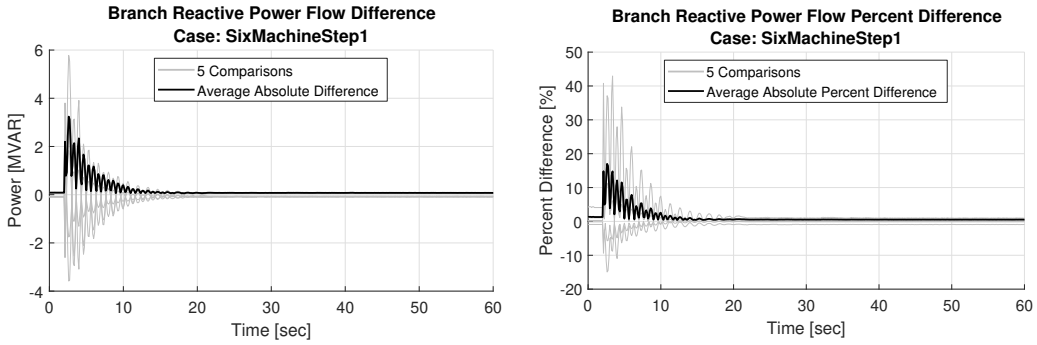

Figure 43: Six machine load step branch reactive power flow comparison.

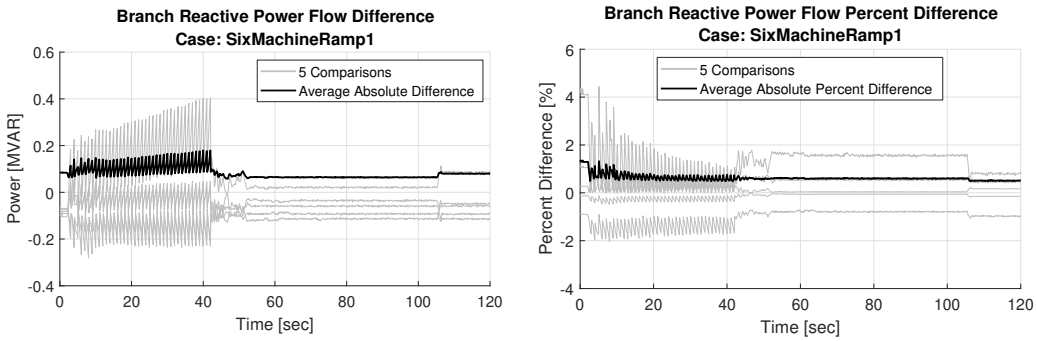

Figure 44: Six machine load ramp branch reactive power flow comparison.

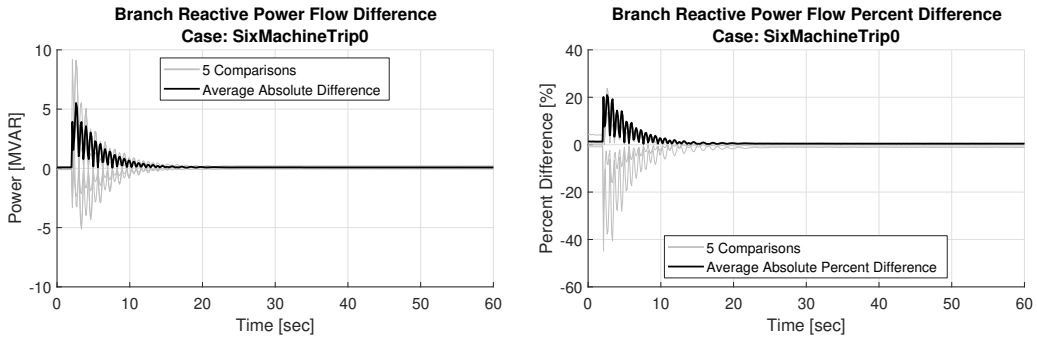

Figure 45: Six machine generator trip branch reactive power flow comparison.

4.4.3. Mini WECC System

A larger system model was used To test the scaling ability of PSLTDSim . The ‘mini WECC’ shown in Figure 46, is a 120 bus 34 generator system created in PSLF. All governors in the miniWECC were modeled with the tgov1 which enabled easier validation. Further details about the creation and use of the miniWECC may be found in [22], [44], [49].

The mini WECC was modified from the original configuration to include three areas. Governors were removed from generators so that each area had only roughly 20% of generation capacity under governor control. Additionally, since the mini WECC was designed to test heavy loading and transient type events, all loads were reduced by 5%. Further, to keep the voltage profile similar to the original design, shunts were also reduced by 5%.

4.4.3.1. Simulated Scenario Descriptions

Similar to the six machine tests, a load step, load ramp, and generator trip were simulated. At $t = 2$ the loads on buses 16, 21, and 26 were each increased by 400 MW. The ramp perturbance also affects the same loads 400 MW, but over 40 seconds. The ramp is equivalent to forty 30 MW steps taking place every second from $t = 2$ to $t = 42$. Generator 27 was tripped at $t = 2$ and was initially generating ≈200 MW.

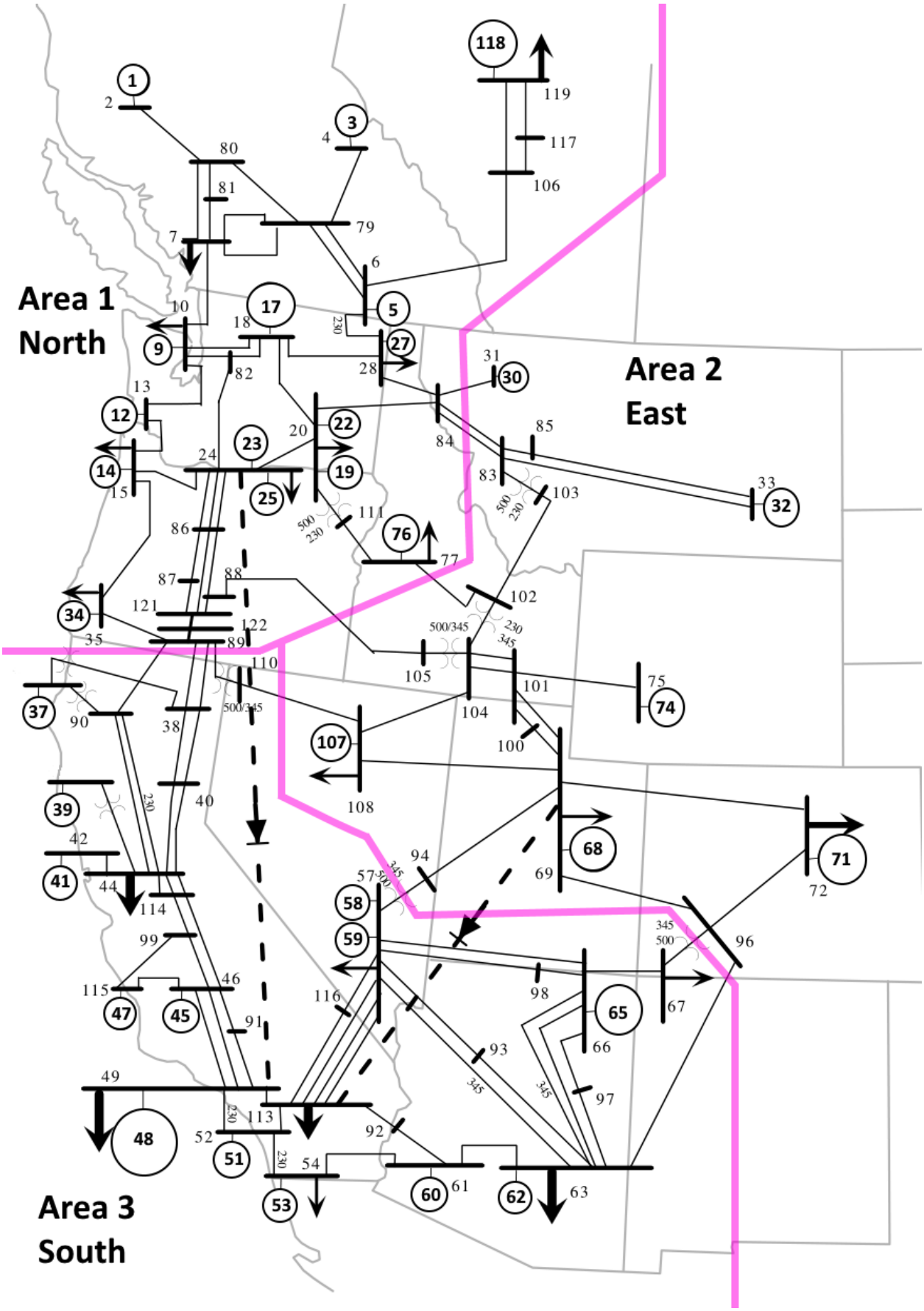

Figure 46: Mini WECC system adapted from [22].

4.4.3.2. Frequency Results

System frequency comparison results are presented for each simulated scenario in Figures 47 48 and 49.

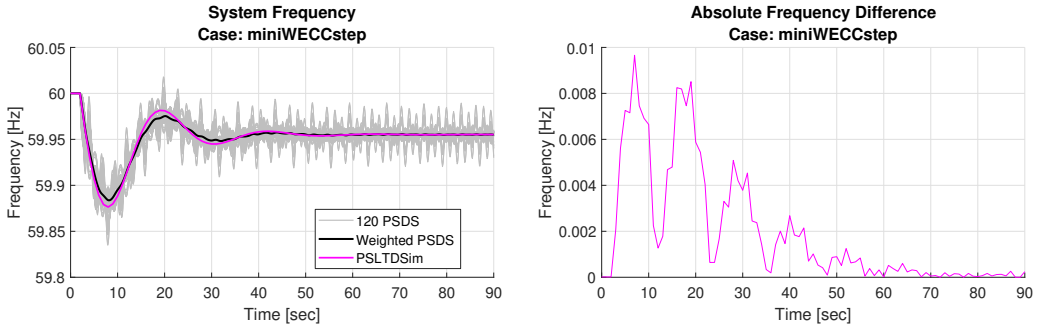

Figure 47: Mini WECC load step system frequency comparison.

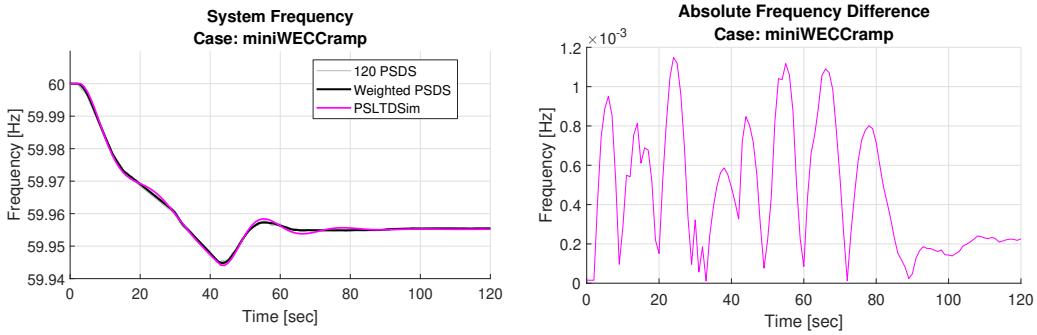

Figure 48: Mini WECC load ramp system frequency comparison.

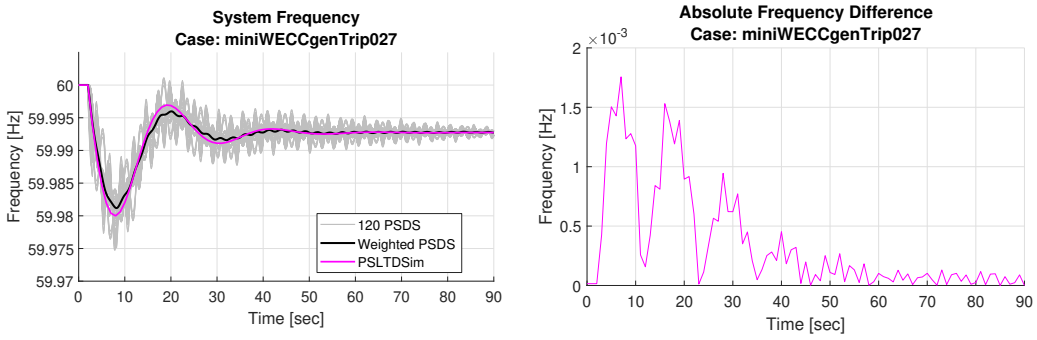

Figure 49: Mini WECC generator trip system frequency comparison.

4.4.3.3. Generator Mechanical Power Results

Generator mechanical power comparison results are presented for each simulated scenario in Figures 50 51 and 52.

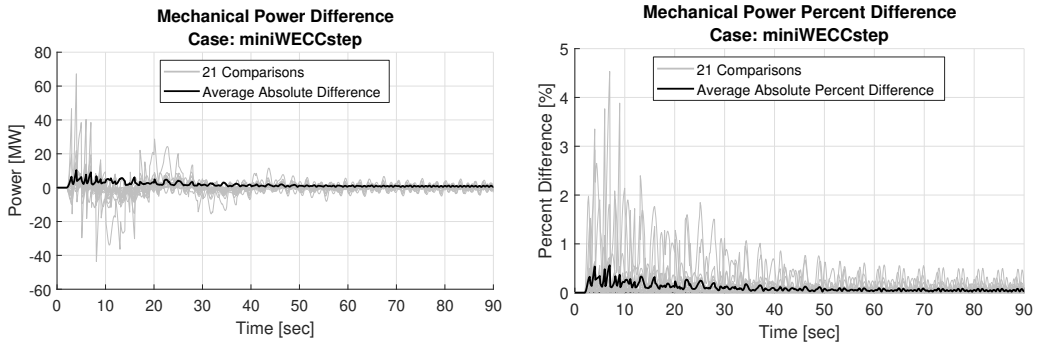

Figure 50: Mini WECC load step mechanical power comparison.

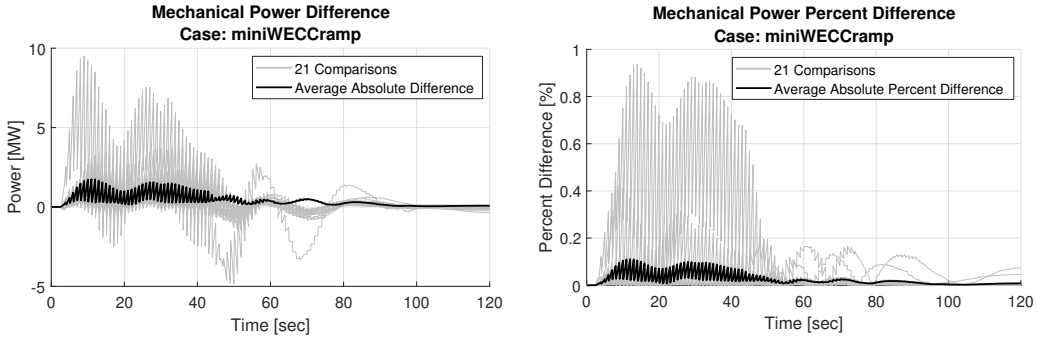

Figure 51: Mini WECC load ramp mechanical power comparison.

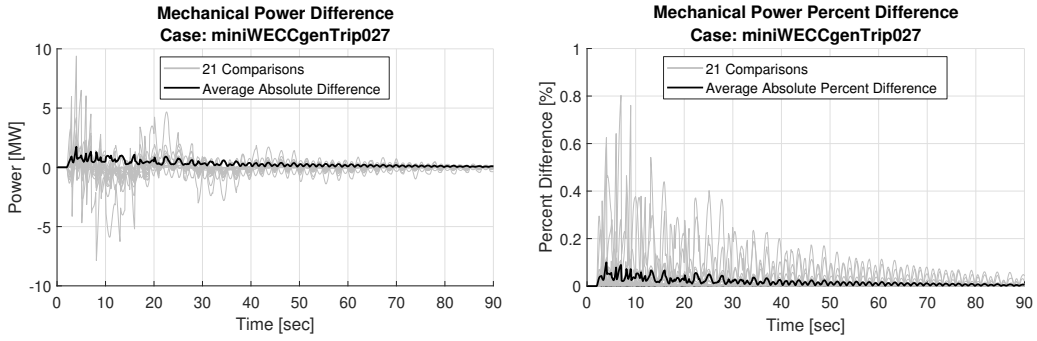

Figure 52: Mini WECC generator trip mechanical power comparison.

4.4.3.4. Generator Real Power Results

Generator real power comparison results are presented for each simulated scenario in Figures 53 54 and 55.

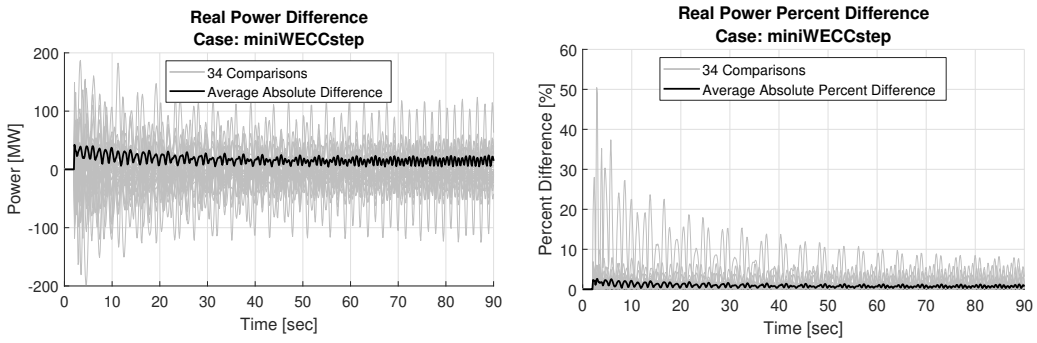

Figure 53: Mini WECC load step real power comparison.

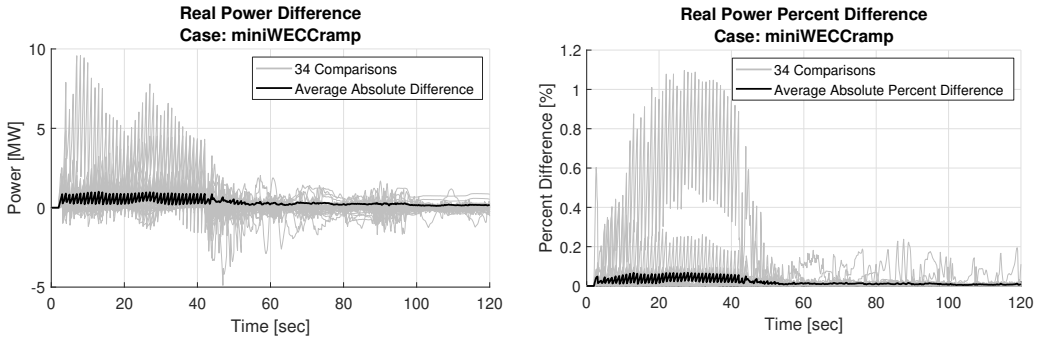

Figure 54: Mini WECC load ramp real power comparison.

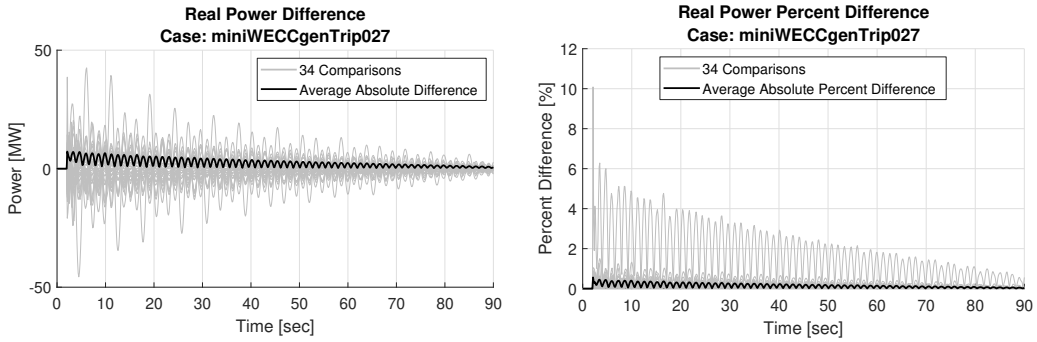

Figure 55: Mini WECC generator trip real power comparison.

4.4.3.5. Voltage Magnitude Results

Bus voltage magnitude comparison results are presented for each simulated scenario in Figures 56 57 and 58.

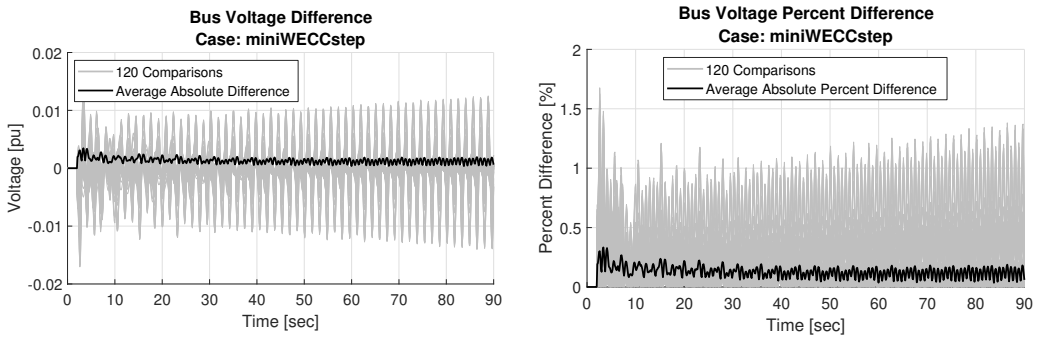

Figure 56: Mini WECC load step voltage comparison.

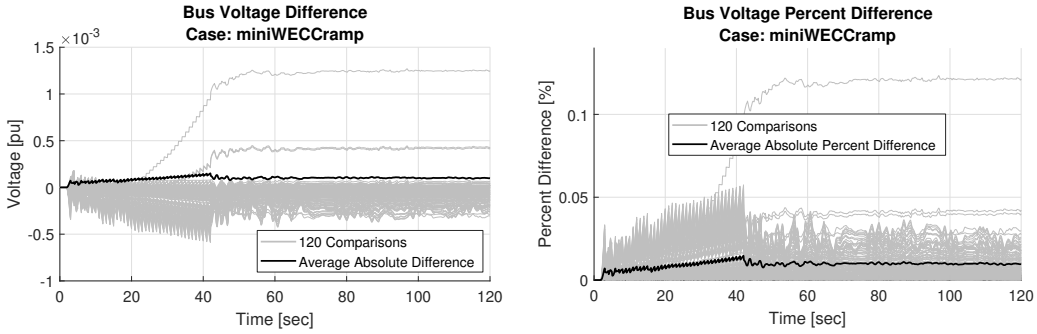

Figure 57: Mini WECC load ramp voltage comparison.

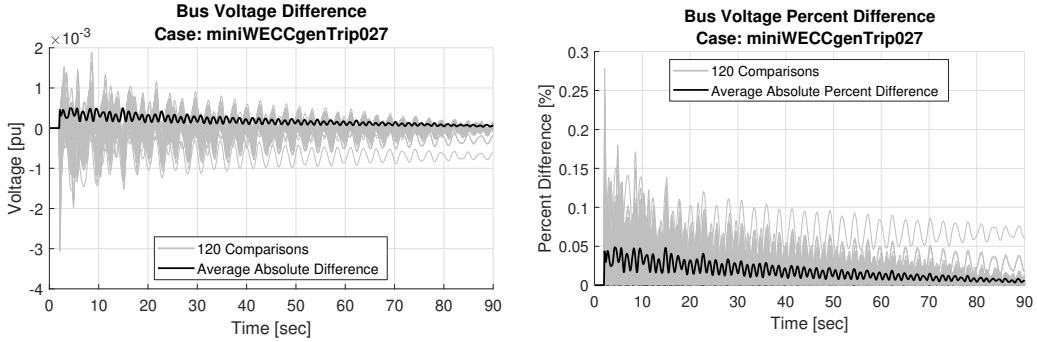

Figure 58: Mini WECC generator trip voltage comparison.

4.4.3.6. Voltage Angle Results

Bus voltage angle comparison results are presented for each simulated scenario in Figures 59 60 and 61.

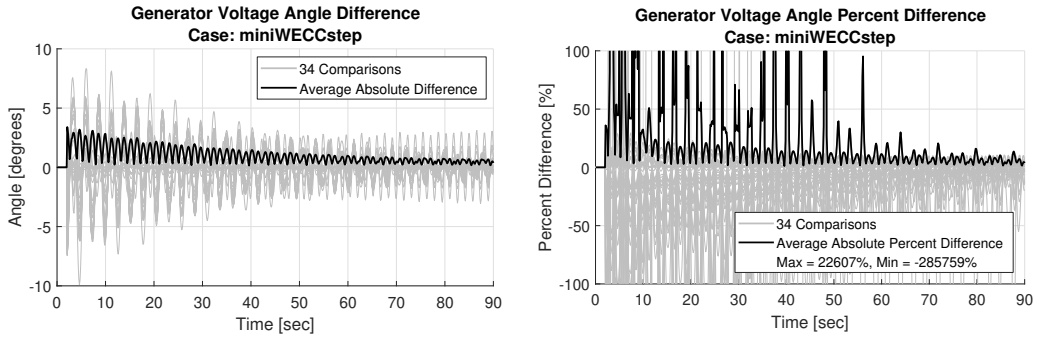

Figure 59: Mini WECC load step voltage angle comparison.

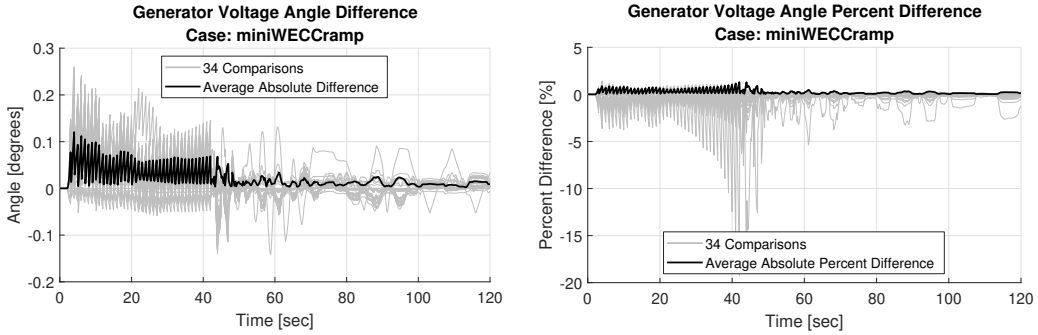

Figure 60: Mini WECC load ramp voltage angle comparison.

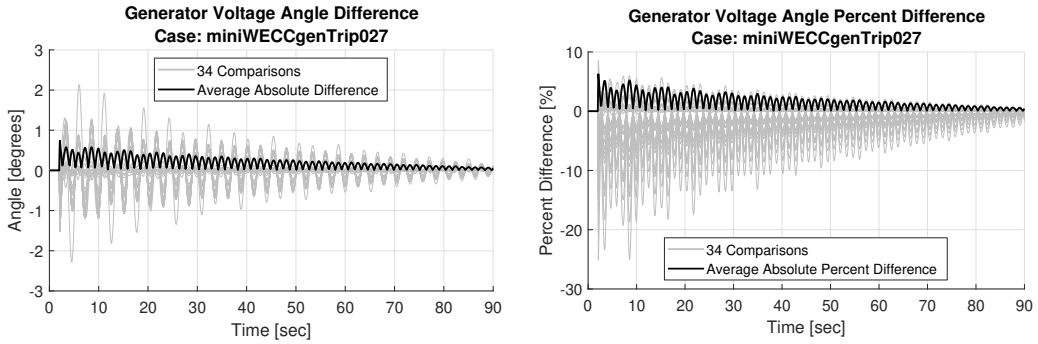

Figure 61: Mini WECC generator trip voltage angle comparison.

4.4.3.7. Generator Reactive Power Results

Generator reactive power comparison results are presented for each simulated scenario in Figures 62 63 and 64.

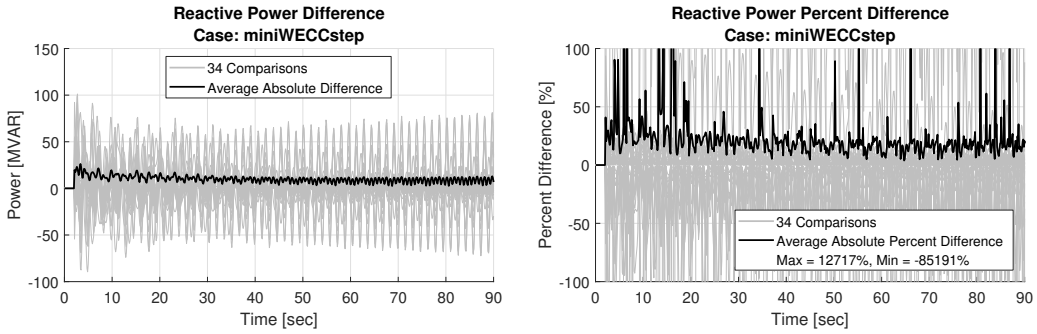

Figure 62: Mini WECC load step reactive power comparison.

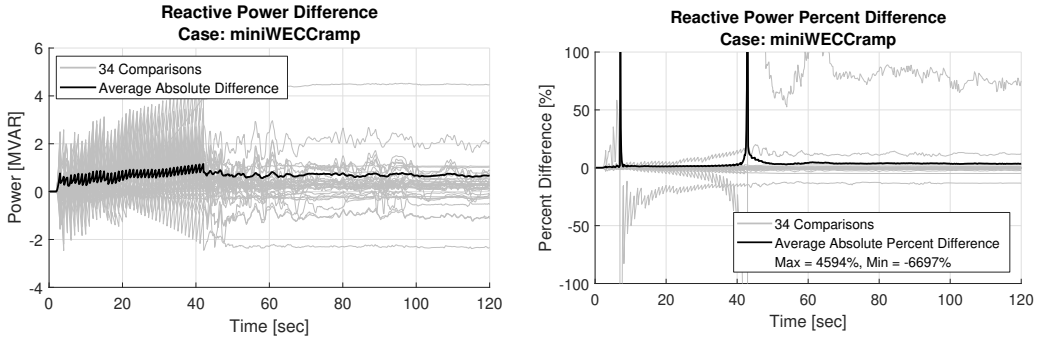

Figure 63: Mini WECC load ramp reactive power comparison.

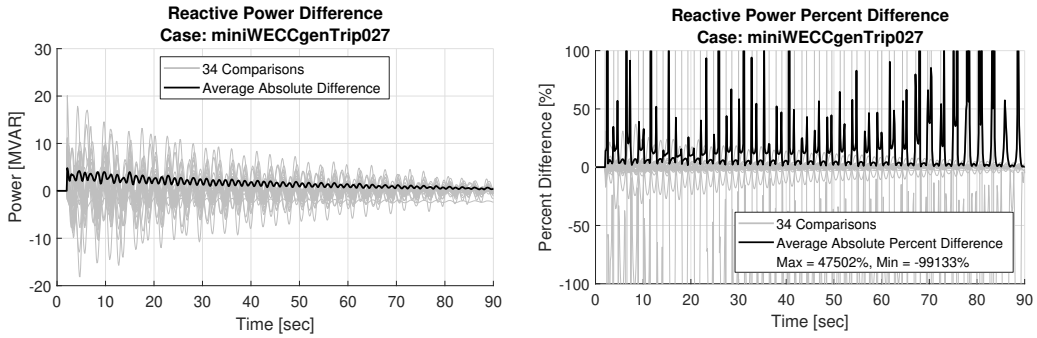

Figure 64: Mini WECC generator trip reactive power comparison.

4.4.3.8. Branch Current Results

Branch current comparison results are presented for each simulated scenario in Figures 65 66 and 67.

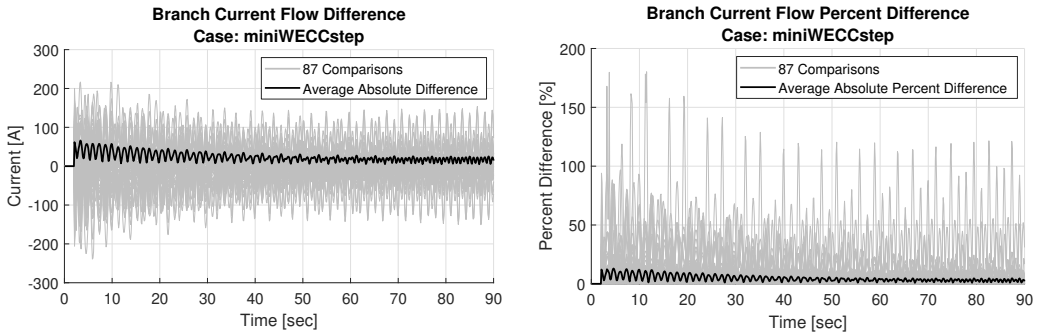

Figure 65: Mini WECC load step branch current flow comparison.

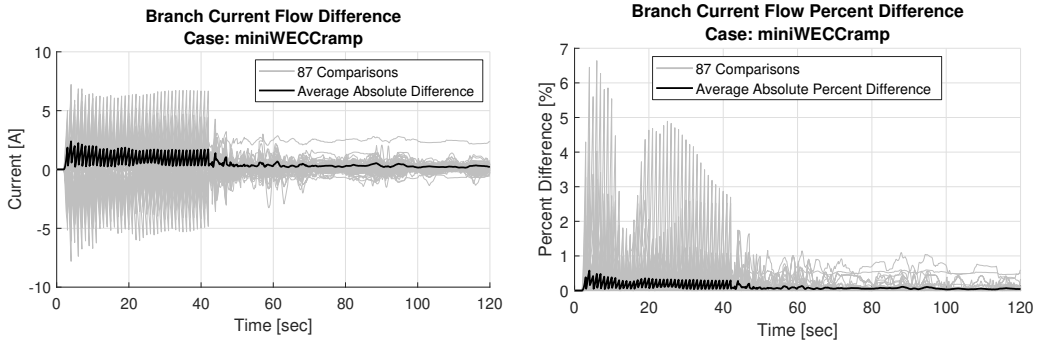

Figure 66: Mini WECC load ramp branch current flow comparison.

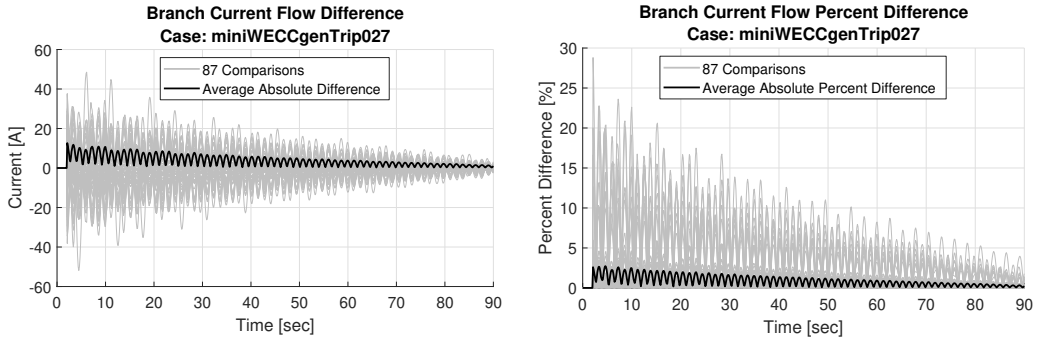

Figure 67: Mini WECC generator trip branch current flow comparison.

4.4.3.9. Branch Real Power Flow Results

Branch real power flow comparison results are presented for each simulated scenario in Figures 68 69 and 70.

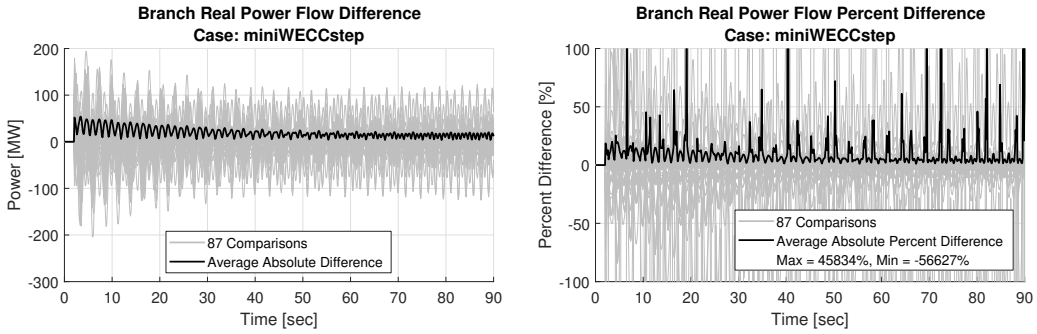

Figure 68: Mini WECC load step branch real power flow comparison.

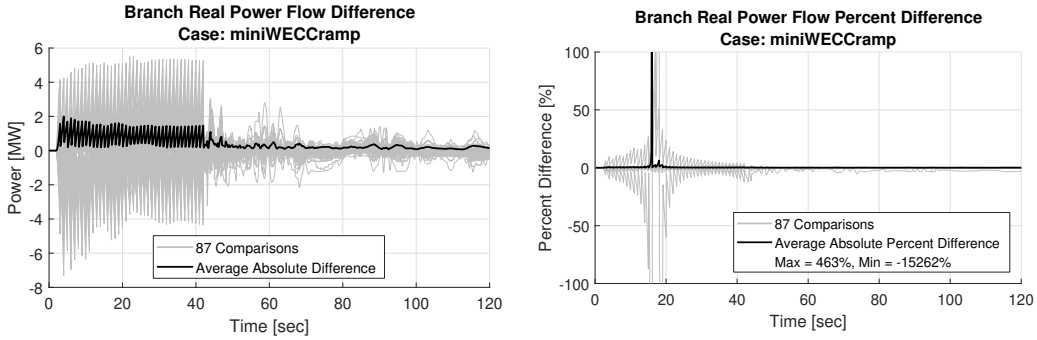

Figure 69: Mini WECC load ramp branch real power flow comparison.

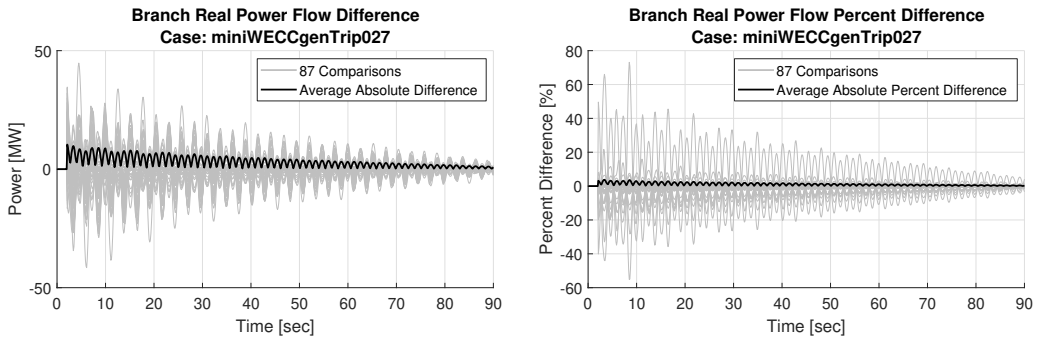

Figure 70: Mini WECC generator trip branch real power flow comparison.

4.4.3.10. Branch Reactive Power Flow Results

Branch reactive power flow comparison results are presented for each simulated scenario in Figures 71 72 and 73.

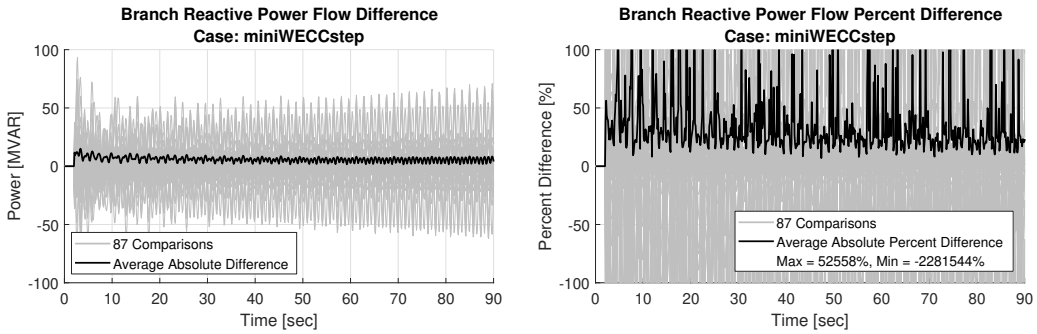

Figure 71: Mini WECC load step branch reactive power flow comparison.

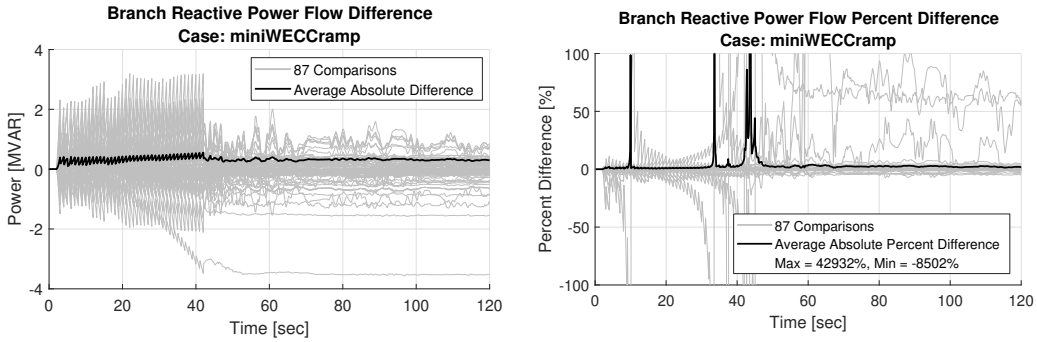

Figure 72: Mini WECC load ramp branch reactive power flow comparison.

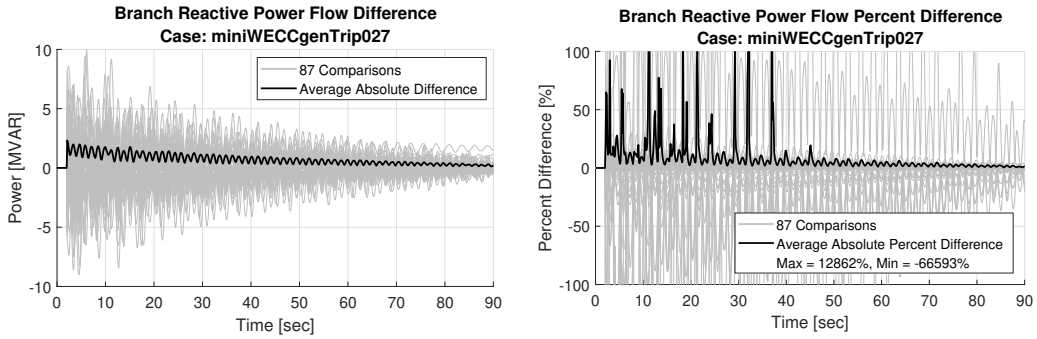

Figure 73: Mini WECC generator trip branch reactive power flow comparison.

4.4.4. Mini WECC with PSS System

The mini WECC, previously shown in Figure 46, had it's power system stabilizers (PSS) turned off in the pervious validation tests. PSS was turned on for the following validations tests.

4.4.4.1. Simulated Scenario Descriptions

The same simulation tests were ran from the previous mini WECC section.. At $t = 2$ the loads on buses 16,21, and 26 were each increased by 400 MW. The ramp perturbance affects the same loads the same amount, but over 40 seconds. The ramp is equivalent to 40 30 MW steps taking place every second from $t = 2$ to $t = 42$. Generator 27 was tripped at $t = 2$ and was initially generating ≈200 MW.

4.4.4.2. Frequency Results

System frequency comparison results are presented for each simualated scenario in Figures 74 75 and 76.

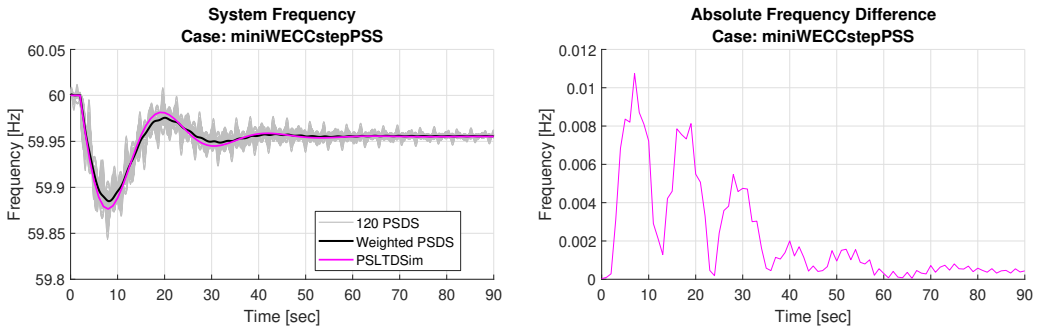

Figure 74: Mini WECC with PSS load step system frequency comparison.

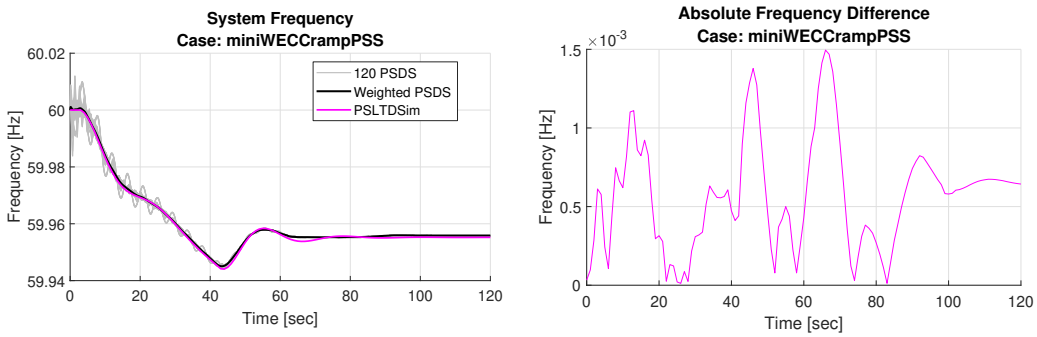

Figure 75: Mini WECC with PSS load ramp system frequency comparison.

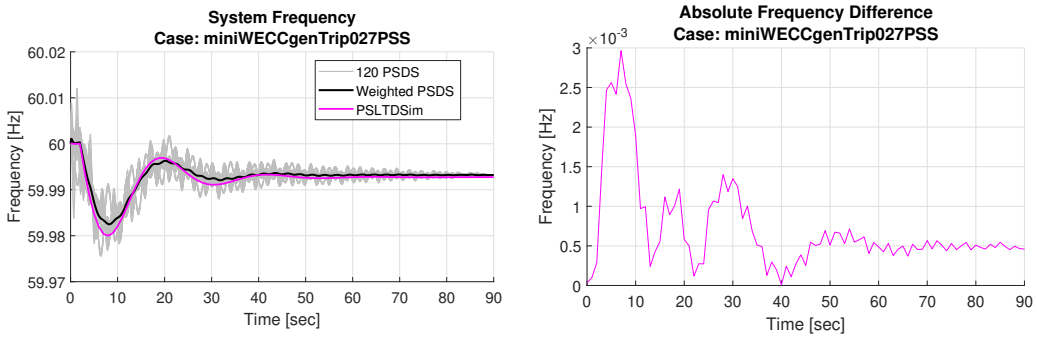

Figure 76: Mini WECC with PSS generator trip system frequency comparison.

4.4.4.3. Generator Mechanical Power Results

Generator mechanical power comparison results are presented for each simulated scenario in Figures 77 78 and 79.

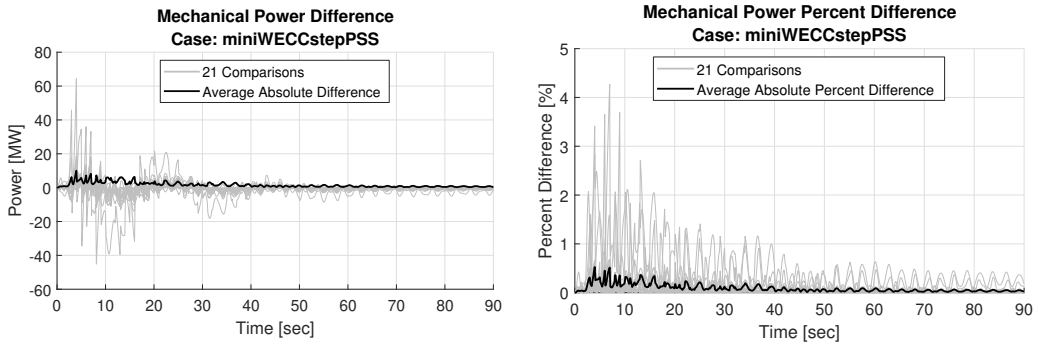

Figure 77: Mini WECC with PSS load step mechanical power comparison.

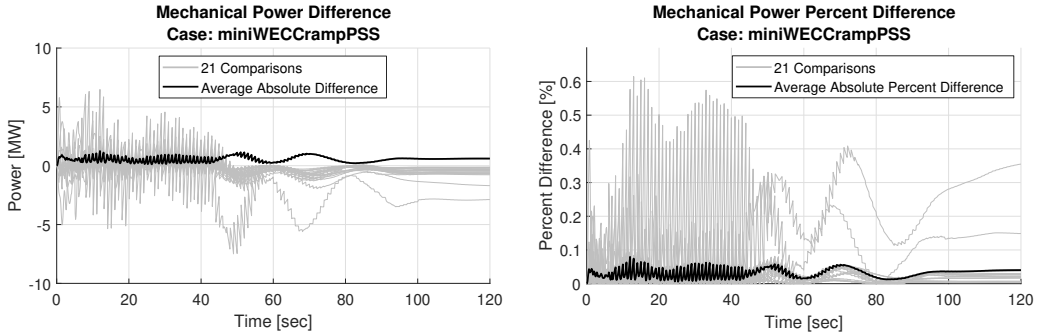

Figure 78: Mini WECC with PSS load ramp mechanical power comparison.

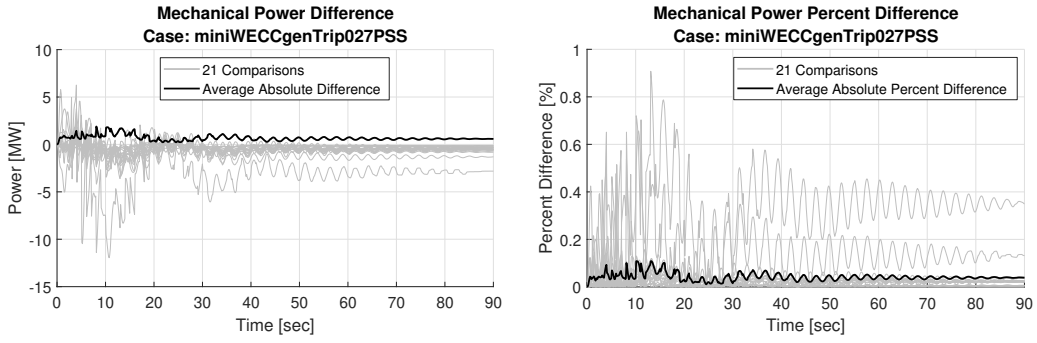

Figure 79: Mini WECC with PSS generator trip mechanical power comparison.

4.4.4.4. Generator Real Power Results

Generator real power comparison results are presented for each simulated scenario in Figures 80 81 and 82.

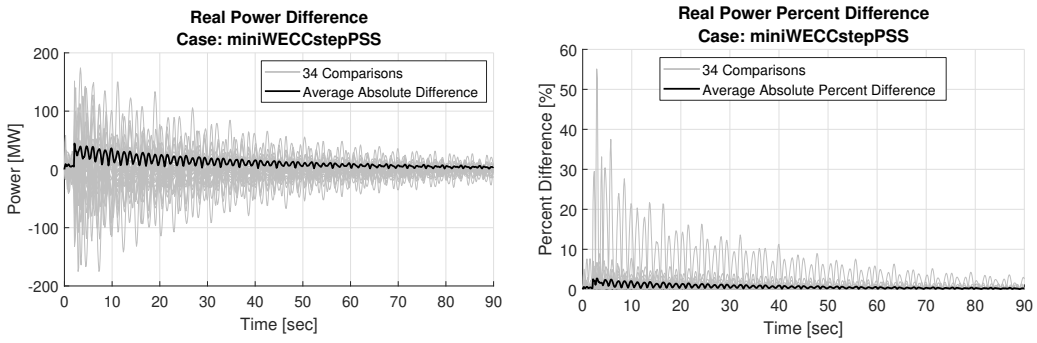

Figure 80: Mini WECC with PSS load step real power comparison.

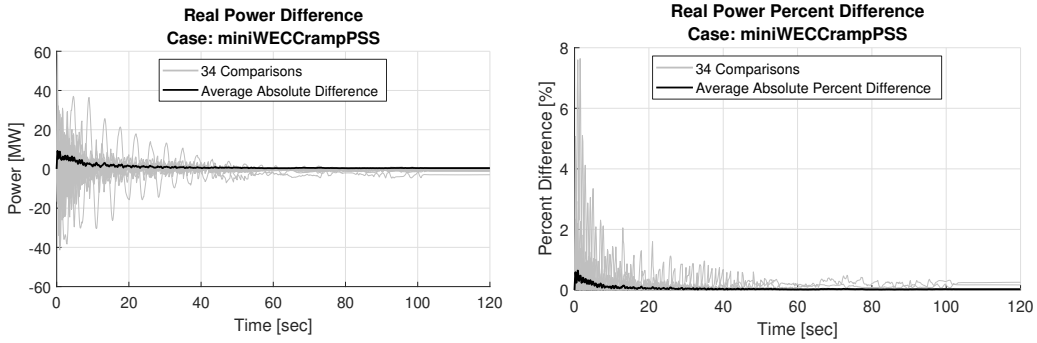

Figure 81: Mini WECC with PSS load ramp real power comparison.

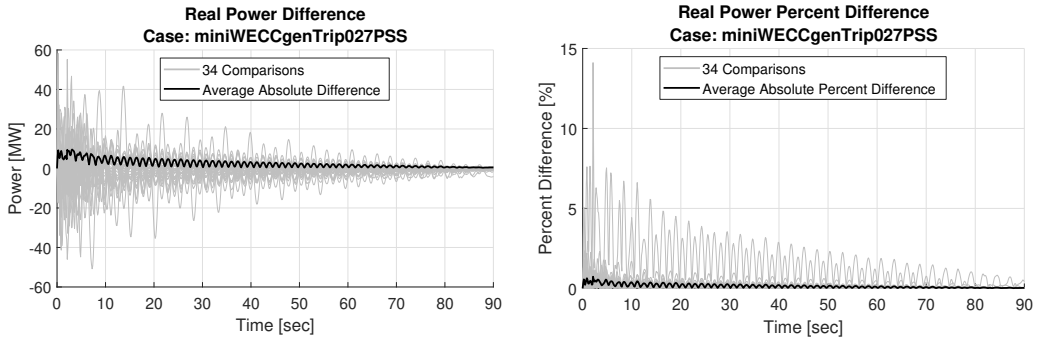

Figure 82: Mini WECC with PSS generator trip real power comparison.

4.4.4.5. Voltage Magnitude Results

Bus voltage magnitude comparison results are presented for each simulated scenario in Figures 83 84 and 85.

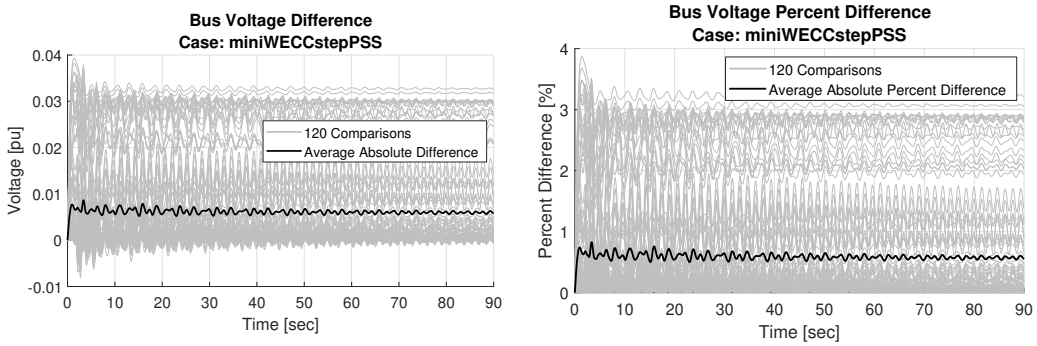

Figure 83: Mini WECC with PSS load step voltage comparison.

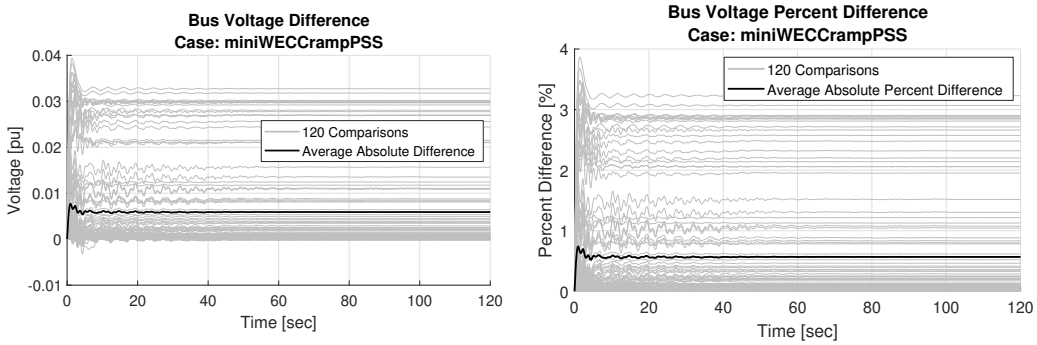

Figure 84: Mini WECC with PSS load ramp voltage comparison.

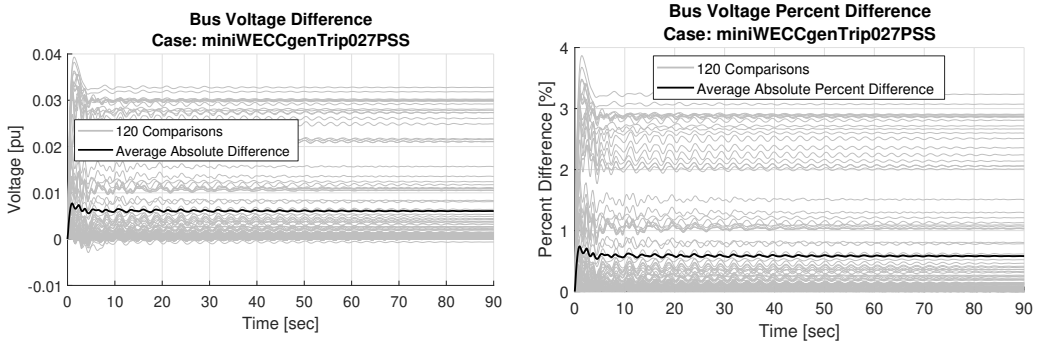

Figure 85: Mini WECC with PSS generator trip voltage comparison.

4.4.4.6. Voltage Angle Results

Bus voltage angle comparison results are presented for each simulated scenario in Figures 86 87 and 88.

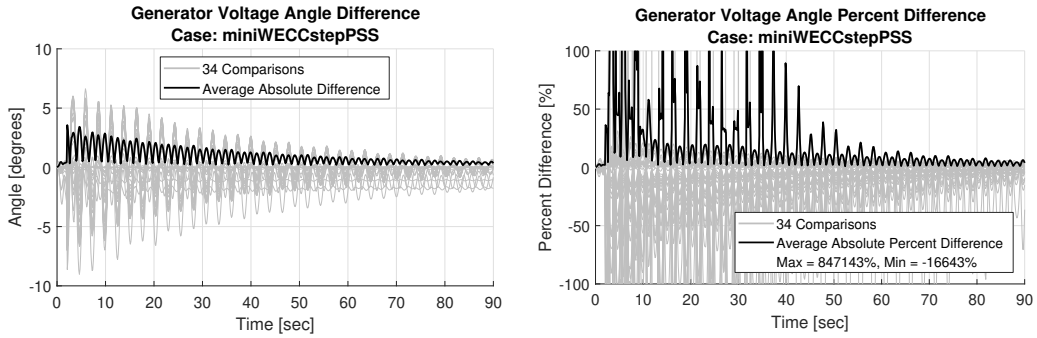

Figure 86: Mini WECC with PSS load step voltage angle comparison.

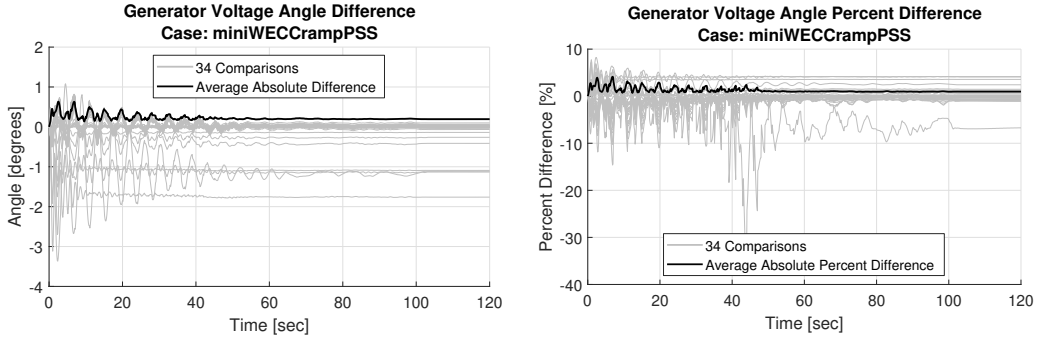

Figure 87: Mini WECC with PSS load ramp voltage angle comparison.

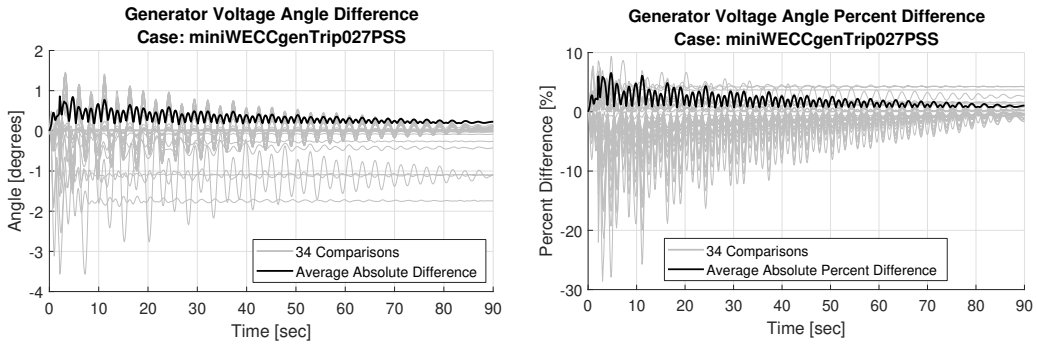

Figure 88: Mini WECC with PSS generator trip voltage angle comparison.

4.4.4.7. Generator Reactive Power Results

Generator reactive power comparison results are presented for each simulated scenario in Figures 89 90 and 91.

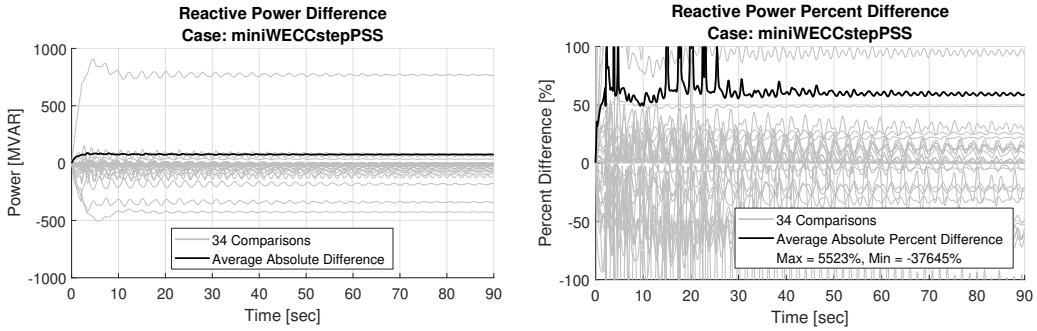

Figure 89: Mini WECC with PSS load step reactive power comparison.

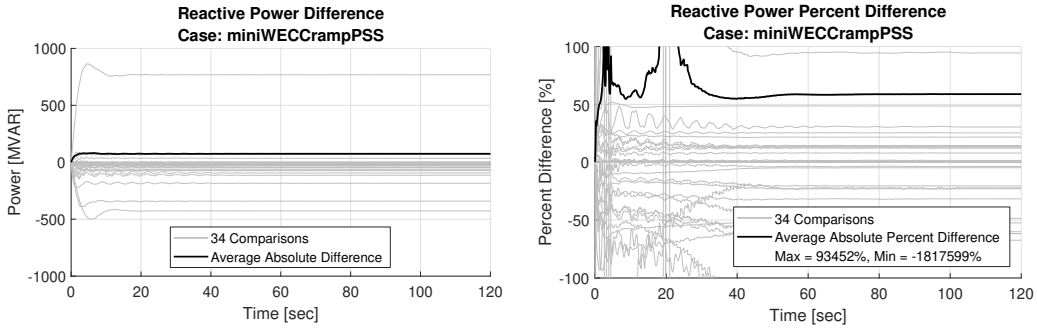

Figure 90: Mini WECC with PSS load ramp reactive power comparison.

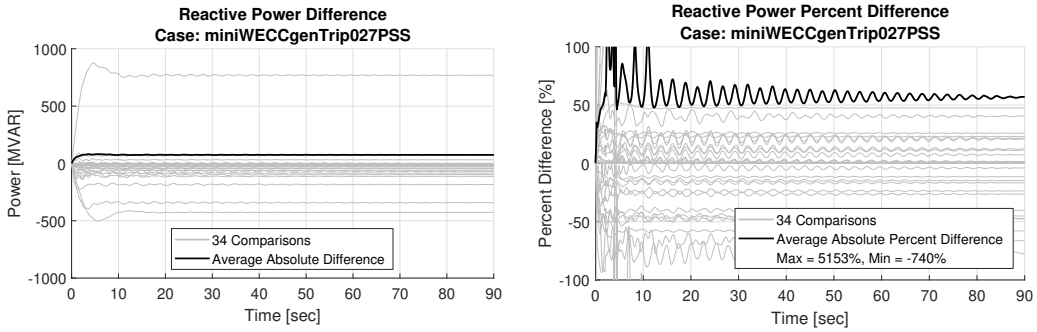

Figure 91: Mini WECC with PSS generator trip reactive power comparison.

4.4.4.8. Branch Current Results

Branch current comparison results are presented for each simulated scenario in Figures 92 93 and 94.

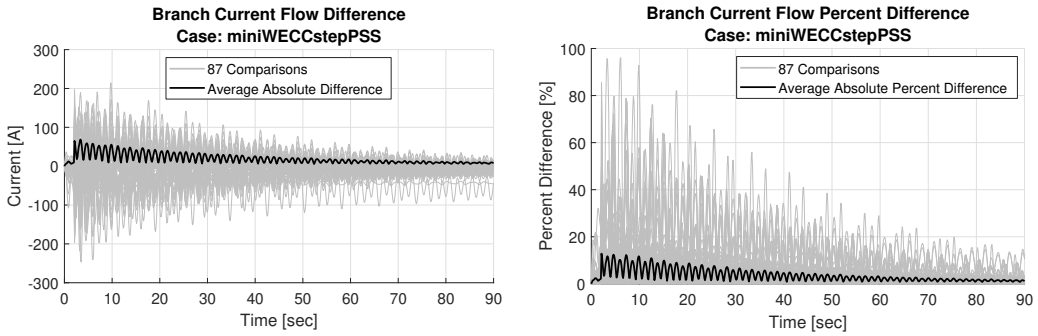

Figure 92: Mini WECC with PSS load step branch current flow comparison.

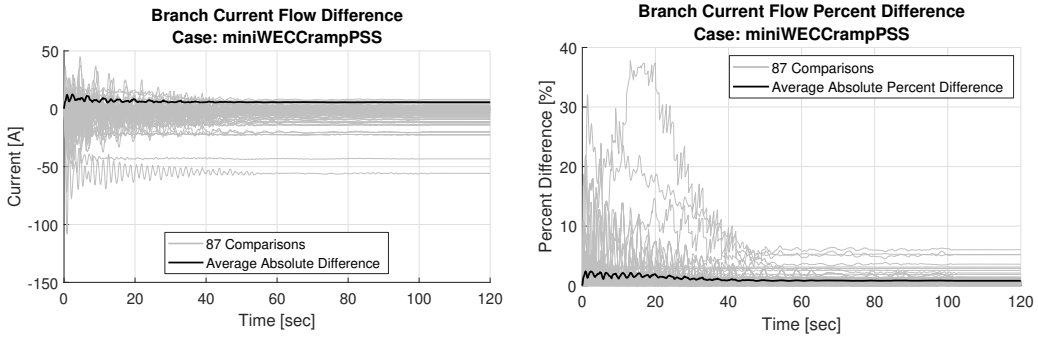

Figure 93: Mini WECC with PSS load ramp branch current flow comparison.

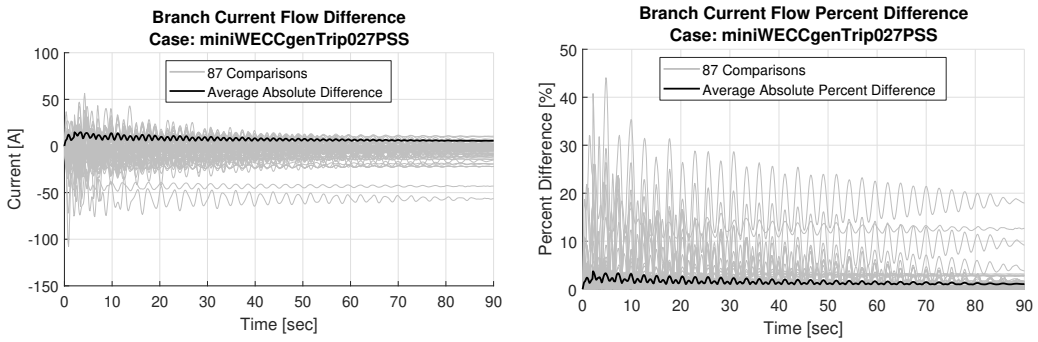

Figure 94: Mini WECC with PSS generator trip branch current flow comparison.

4.4.4.9. Branch Real Power Flow Results

Branch real power flow comparison results are presented for each simulated scenario in Figures 95 96 and 97.

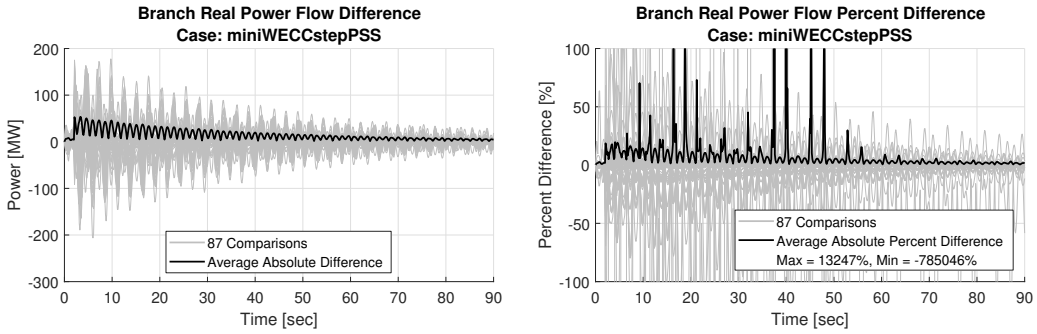

Figure 95: Mini WECC with PSS load step branch real power flow comparison.

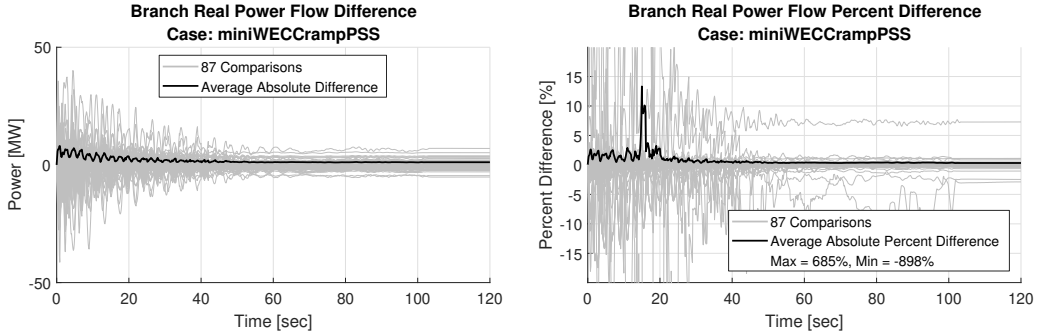

Figure 96: Mini WECC with PSS load ramp branch real power flow comparison.

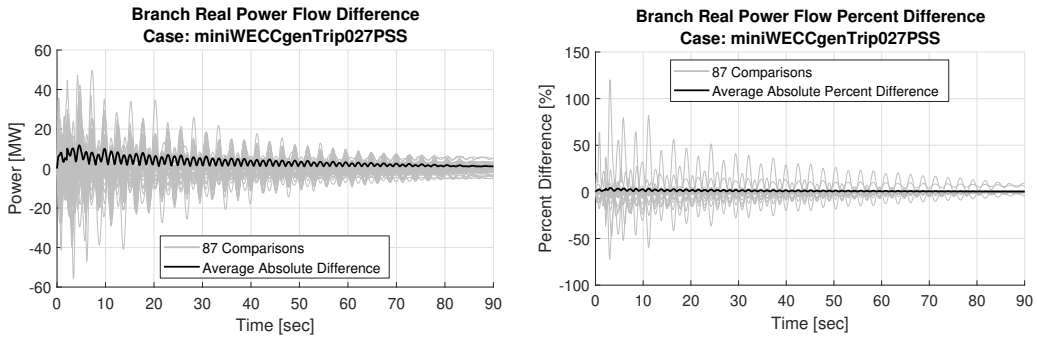

Figure 97: Mini WECC with PSS generator trip branch real power flow comparison.

4.4.4.10. Branch Reactive Power Flow Results

Branch reactive power flow comparison results are presented for each simulated scenario in Figures 98 99 and 100.

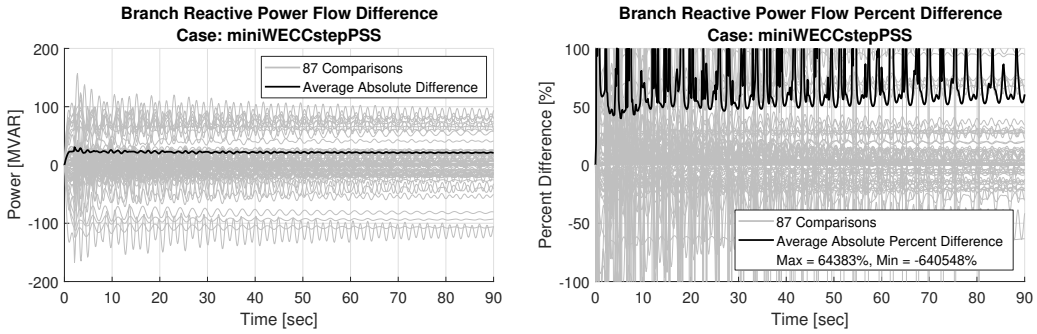

Figure 98: Mini WECC with PSS load step branch reactive power flow comparison.

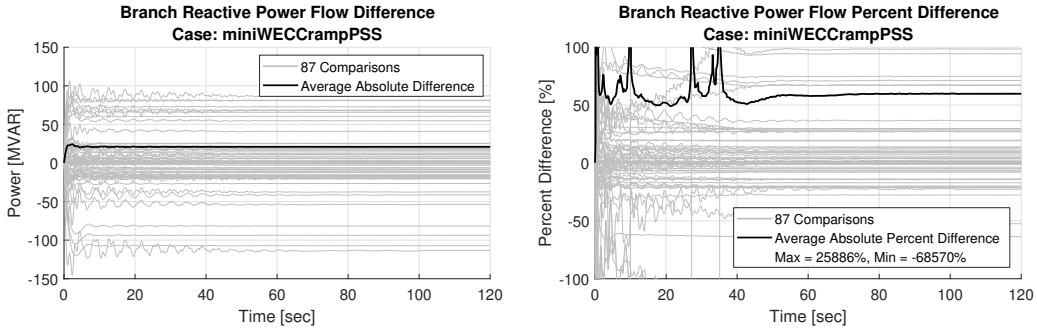

Figure 99: Mini WECC with PSS load ramp branch reactive power flow comparison.

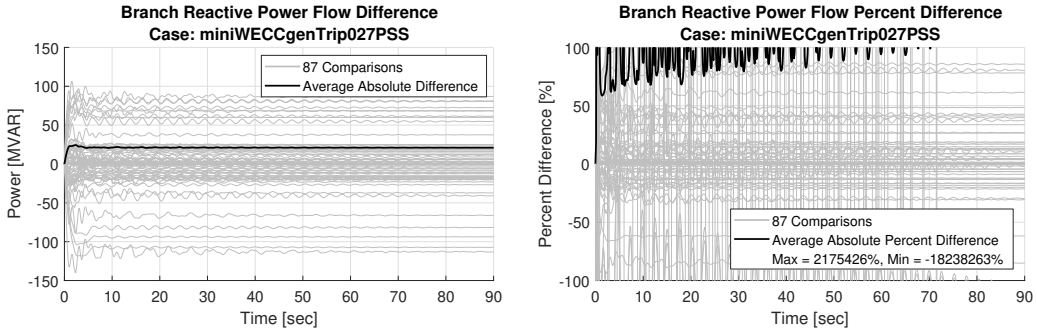

Figure 100: Mini WECC with PSS generator trip branch reactive power flow comparison.

4.4.5. Full WECC

WECC system with generic governors to estimate unmodeled governors (put some number to it). Only a load step and load ramp are considered and only frequency is compared.

4.4.5.1. Load Step

4.4.5.2. Load Ramp

4.4.6. Validation Summary

Recap PSDS vs LTD results, hit high points, more like a discussion section, average the quantifiable stuffs…

System frequency ok in ramps - software not designed for transient events or oscillatory systems.

As expected, more assumtions in simulation lead to larger differences between LTD and CTS.

5. Engineering Problem

Problem of focus still yet to be decided. However, this section should offer a semi-restate of specific problem this research was focused on and solution predictions. Governor deadband and delay settings / AGC settings?

5.1. Relevant NERC Standards

NERC has various mandatory standards subject to enforcement in the United States. Standards starting with BAL are related to resource and demand balancing. BAL standards that are applicable to BA simulation are described here.

5.1.1. BAL-001-2

NERC standard BAL-001-2 has two requirements. The first requirement has to do with a monthly calculation that is out of the scope of PSLTDSim. The second requirement specifies reported ACE (RACE) not exceeding a certain frequency trigger limit (FTL) for more than 30 continuous clock-minutes [35]. Since PSLTDSim can simulate longer than 30 minutes this requirement is important to BA simulation. NERC provides the following equations describing how to calculate a balancing authority's ACE limit (BAAL).

$$BAAL = -10B(FTL - F_s)\frac{FTL - F_S}{F_A - F_S} \tag{15}$$

$$FTL = F_S \pm 3\epsilon \tag{16}$$

If actual system frequency F_A is above the scheduled frequency F_S, then the $\pm$ in equation 16 is a $+$. If the opposite is true ($F_A < F_S$), then the $\pm$ is a $-$. NERC has set values for each interconnection to use for ϵ. In the Western Interconnection $\epsilon = 0.0228$ Hz. Note that minute average frequency is used as F_A the above equations.

Figure 101 visualizes the NERC equations for various settings of B. When frequency is at the system rated 60 Hz ACE is unlimited. Larger values of B equate to larger values of BAAL.

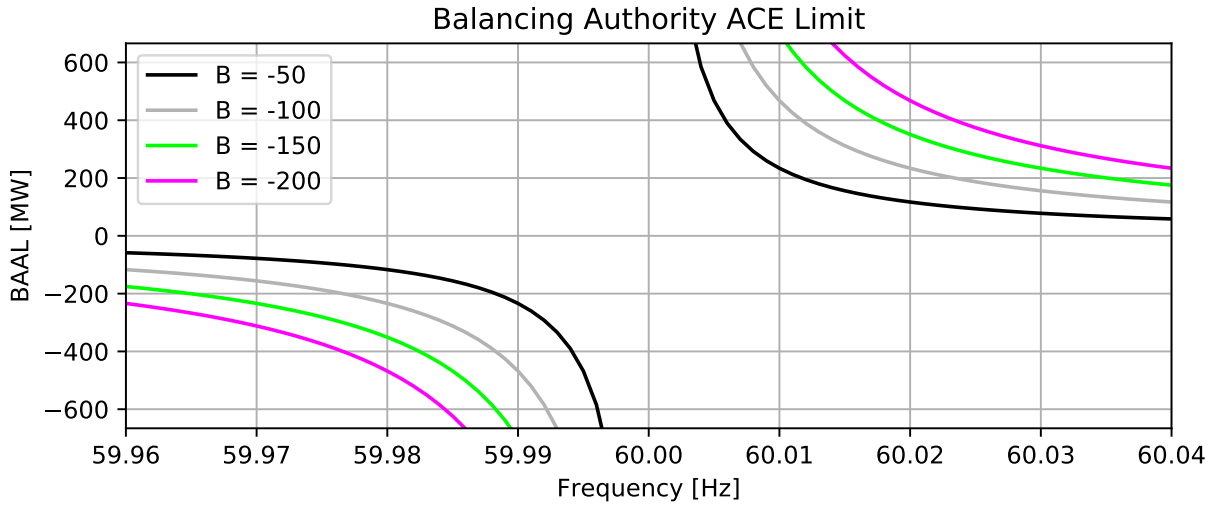

Figure 101: Balancing authority ACE limit for different values of B.

5.1.2. BAL-002-3

NERC standard BAL-002-3 requires a BA to return reporting ACE to zero if pre-contingency ACE was positive or zero, or the pre-contingency value if ACE was negative within the contingency event recovery period. The NERC definition for contingency event recovery period is 15 minutes after the start of a contingency [36].

5.1.3. BAL-003-1.1

NERC stanard BAL-003-1.1 deals with the frequency bias setting used by a BA. While realistic frequency bias B is calculated based on real system values and dictated to a BA, this doesn't apply for theoretical models. Therfore, for simplicity, a setting of 0.9% maximum generation capacity will be used for B. This setting is in line with recommendations made in [34].

5.1.4. NERC Standard Summary

To follow NERC settings, a frequency bias of 0.9% maximum generation capcity will be used for all areas. Additionally, RACE must return to zero or the pre-contingency level within 15 minutes of a contingency. Finally, any control that allows for more than 30 consecutive minutes of RACE exceeding the calculated BAAL will not be acceptable.

5.2. Events of Interest

Simulate 4-6 hours of noise, generation ramps (wind), or daily load cycles and/or various combinations of to generate some useful to real life scenario.

Proposed events:

- Large governor deadband, Fast AGC (seconds)
- normal governor deadband, Slow AGC (minutes)
- normal governor deadband, Fast AGC (seconds)

5.3. 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 XII in Appendix 11 and a detailed description of the more involved control is described in the following sections.

5.3.1. Governor Deadbands

The FERC maximum deadband is 36 mHz[12]. 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 102 graphically depicts how the various deadband options differ.

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 in-

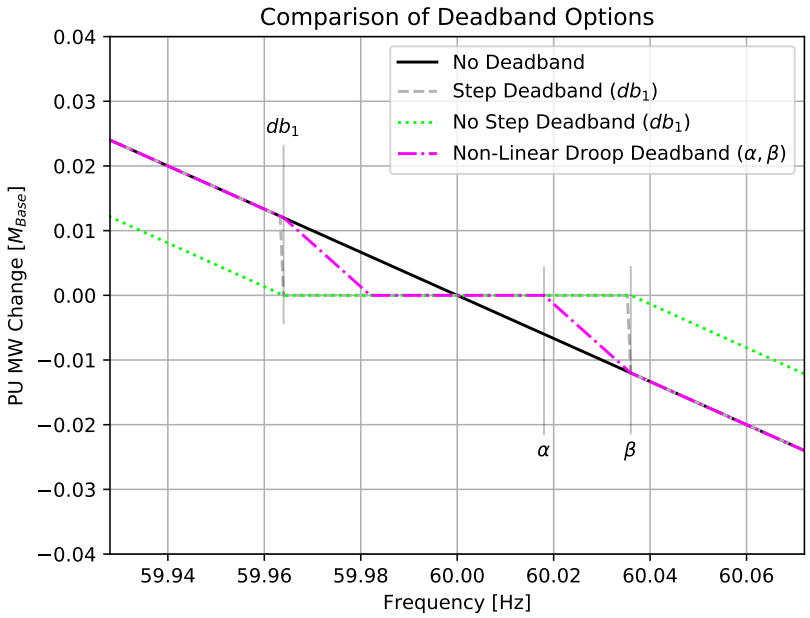

Figure 102: Examples of available deadband action.

put, (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.

5.3.2. 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.

5.3.3. Area Wide Governor Droops

The typical FERC droop is 5%[12]. 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 config-

uration of simulations aimed at exploring droop settings.

5.3.4. Automatic Generation Control

The general workings of AGC is shown in Figure 103. Simulation settings related to frequency bias, ACE integrating and filtering, and conditional and weighted summing are explained in the following sections.

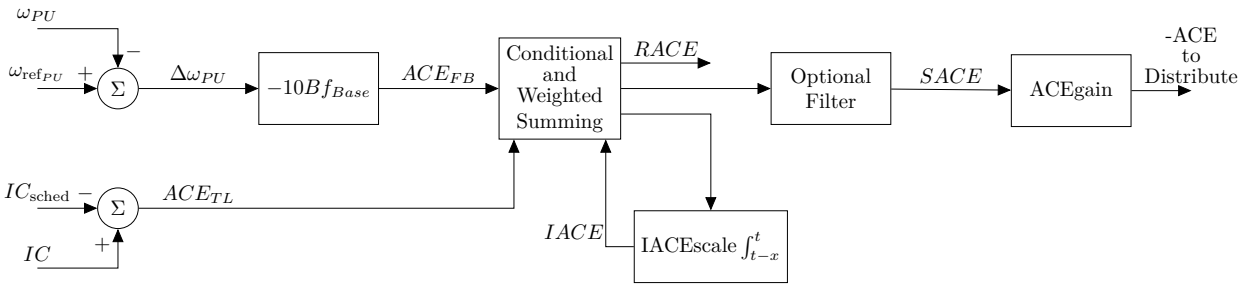

Figure 103: Block diagram of ACE calculation and manipulation.

5.3.4.1. 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.

5.3.4.2. Integral of Area Control Error

As previously shown in Figure 103, 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.

5.3.4.3. 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 V shows the various conditional summations.

Table V: 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...

5.3.4.4. 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.

5.3.4.5. 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.

5.4. Simulation Methodology

How simulations were run and comparative metrics calculated NERC mandate adherence valve travel

5.5. Simulation Results

Using miniWECC with no PSS

5.5.1. AGC Tuning

Required tests for setting of various AGC settings - such as update time, filtering, and gain. Load steps in each area?

5.5.2. System Noise Only

Baseline responses

5.5.3. System Noise and Generation Ramp

Supoose a wind ramp occurs

5.5.4. System Noise and Daily Load Cycle

maybe during a peak - system must be moving

5.5.5. System Noise, Generation Ramp, and Daily Load Cycle

essentially a combination of all previous events...

6. Conclusions

6.1. Simulation Findings

6.2. Engineering Problem Findings

7. 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

8. Bibliography

[1] (2019). About ferc, FERC, [Online]. Available: https://www.ferc.gov/about/about.asp.

[2] (2017). About nerc, FERC, [Online]. Available: https://www.nerc.com/AboutNERC/Pages/default.aspx.

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

[4] 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.

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

[6] 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.

[7] 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.

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

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

[10] (2017). Ercot-internconnection_branded.jpg, ERCOT, [Online]. Available: http://www.ercot.com/news/mediakit/maps.

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

[12] 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.

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

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

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

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

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

[18] 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.

[19] (2020). Glossary of terms used in nerc reliability standards, NERC, [Online]. Available: https://www.nerc.com/files/glossary_of_terms.pdf.

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

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

[22] 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.

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

[24] 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.

[25] (2019). July2019map.png, U.S. Energy Information Administration, [Online]. Available: https://www.eia.gov/electricity/data/eia860m/.

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

[27] 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.

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

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

[30] 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.

[31] (2004). Montana electric transmission grid: Operation, congestion, and issues, DEQ, [Online]. Available: https://leg.mt.gov/content/publications/Environmental/2004deq_energy_report/transmission.pdf.

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

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

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

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

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

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

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

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

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

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

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

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

[44] 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.

[45] 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.

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

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

[48] (2017). Thread in operating system, GeeksforGeeks, [Online]. Available: `https://www.geeksforgeeks.org/thread-in-operating-system/`.

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

[50] (2016). U.s. electric system is made up of interconnections and balancing authorities, EIA, [Online]. Available: `https://www.eia.gov/todayinenergy/detail.php?id=27152`.

[51] (2019). U.s. electric system operating data, U.S. Energy Information Administration, [Online]. Available: `https://www.eia.gov/realtime_grid/`.

[52] (2019). U.s. energy mapping system, U.S. Energy Information Administration, [Online]. Available: `https://www.eia.gov/state/maps.php?v=Electricity`.

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

9. Six Machine System Details

Relevant values from PSLF tables describing the six machine system used are presented in this appendix. The ‘busd’ table from PSLF describing system buses is shown in Table VI. The ‘secdd’ table from PSLF describing system lines is shown in Table VII. The ‘tran’ table from PSLF describing system transformers is shown in Table VIII. The ‘gens’ table from PSLF describing system generators is shown in Table IX. The ‘load’ table from PSLF describing system loads is shown in Table X. The ‘shunt’ table from PSLF describing system shunts is shown in Table XI. The dyd file used for validation is shown in Figure 104.

Table VI: Six machine bus table.

BUS-NO	NAME	KV	TP	VSCHED	V-PU	DEG	AREA
6	6	138.00	1	1.00	0.9537	-15.91	1
7	7	138.00	1	1.00	0.9533	-16.07	1
8	8	138.00	1	1.00	0.9532	-16.52	1
9	9	138.00	1	1.00	0.9533	-16.54	2
10	10	138.00	1	1.00	0.9543	-16.14	2
11	11	138.00	1	1.00	0.9549	-15.78	2
1	1	22.00	0	1.00	1.0000	0.00	1
2	2	22.00	2	1.00	1.0000	11.41	1
3	3	22.00	2	1.00	1.0000	1.27	2
4	4	22.00	2	1.00	1.0000	1.27	2
5	5	22.00	2	1.00	1.0000	-10.72	2

Table VII: Six machine line table.

FROM	FNAME	FKV	TO	TNAME	TKV	CK	SE	R-PU	X-PU	B-PU	AF	AT
6	6	138.00	7	7	138.00	1	1	0.0001	0.001	0.0018	1	1
7	7	138.00	8	8	138.00	1	1	0.0001	0.001	0.0018	1	1
8	8	138.00	9	9	138.00	1	1	0.0001	0.001	0.0018	1	2
8	8	138.00	9	9	138.00	2	1	0.0001	0.001	0.0018	1	2
8	8	138.00	9	9	138.00	3	1	0.0001	0.001	0.0018	1	2
11	11	138.00	10	10	138.00	1	1	0.0001	0.001	0.0018	2	2
10	10	138.00	9	9	138.00	1	1	0.0001	0.001	0.0018	2	2

Table VIII: Six machine transformer table.

FROM	FNAME	FKV	TO	TNAME	TKV	MVA	VNOMF	VNOMT	R	X	BMAG	AREA
1	1	22.00	6	6	138.00	100.00	22.00	138.00	0.00	0.10	0.00	1
2	2	22.00	7	7	138.00	100.00	22.00	138.00	0.00	0.10	0.00	1
3	3	22.00	11	11	138.00	100.00	22.00	138.00	0.00	0.10	0.00	2
4	4	22.00	11	11	138.00	100.00	22.00	138.00	0.00	0.10	0.00	2
5	5	22.00	10	10	138.00	100.00	22.00	138.00	0.00	0.10	0.00	2

Table IX: Six machine generator table.

BUS-NO	NAME1	KV1	ID	PGEN	QGEN	IREG	AREA	MBASE	PMAX
1	1	22.00	1	261.40	82.90	1	1	900.00	1000.00
2	2	22.00	1	220.00	77.10	2	1	900.00	1000.00
2	2	22.00	2	220.00	77.10	2	1	900.00	1000.00
3	3	22.00	1	280.00	87.10	3	2	900.00	1000.00
4	4	22.00	1	280.00	87.10	4	2	900.00	1000.00
5	5	22.00	1	90.00	49.90	5	2	900.00	1000.00

Table X: Six machine load table.

BUS-NO	NAME	KV	ID	ST	PLOAD	QLOAD	AREA
8	8	138.00	1	1	600.00	100.00	1
9	9	138.00	1	1	750.00	100.00	2

Table XI: Six machine shunt table.

FROM	FNAME	FKV	ID	CK	ST	G-PU	B-PU	AREA
8	8	138.00	1	1	1	0.00	1.00	1
8	8	138.00	2	2	1	0.00	0.50	1
8	8	138.00	3	3	0	0.00	0.50	1
8	8	138.00	4	4	0	0.00	0.50	1
9	9	138.00	1	1	1	0.00	1.00	2
9	9	138.00	2	2	0	0.00	0.50	2
9	9	138.00	3	3	0	0.00	0.50	2
9	9	138.00	4	4	0	0.00	0.50	2

```
# Six Machine, all gens and govs the same
# exciters default settings

# Metering of frequency, current, and voltages
fmeta   1 "1"   22.00 "1 "         : #9 0 1
ameta   1 "1"   22.00 "1 "         : 0
vmeta   1 "1"   22.00 "1 "         : 0

# current meters
imetr        6 ! ! ! 7 ! ! !        : #9 0
imetr        7 ! ! ! 8 ! ! !        : #9 0
imetr        8 ! ! ! 9 ! ! !        : #9 0
imetr        11 ! ! ! 10 ! ! !        : #9 0
imetr        10 ! ! ! 9 ! ! !        : #9 0

#   loads (Using wlwscc on any load sets the dynamics characteristics of all loads.)
wlwscc  9   "9"      138.0 "1 " : #9 0.0  0.0  0.0  0.0  0.0  1.0  1.0  0.0   0.0   0.0  0.0

#   generators
genrou   1 "1" 22.00 "1 " : #9 mva=900.00   "tpdo" 6.50 "tppdo" 0.079 "tpqo" 0.53 "tppqo" 0.072 "h" 4   "d"
↪  0.0000 "ld" 1.24  "lq" 1.22 "lpd" 0.23 "lpq" 0.36000 "lppd" 0.17 "ll" 0.14 "s1" 0.173 "s12" 0.447 "ra"
↪  0.0000 "rcomp" 0.0000 "xcomp" 0.0000 "accel" 1.0
genrou   2 "2" 22.00 "1 " : #9 mva=900.00   "tpdo" 6.50 "tppdo" 0.079 "tpqo" 0.53 "tppqo" 0.072 "h" 4   "d"
↪  0.0000 "ld" 1.24  "lq" 1.22 "lpd" 0.23 "lpq" 0.36000 "lppd" 0.17 "ll" 0.14 "s1" 0.173 "s12" 0.447 "ra"
↪  0.0000 "rcomp" 0.0000 "xcomp" 0.0000 "accel" 1.0
genrou   2 "2" 22.00 "2 " : #9 mva=900.00   "tpdo" 6.50 "tppdo" 0.079 "tpqo" 0.53 "tppqo" 0.072 "h" 4   "d"
↪  0.0000 "ld" 1.24  "lq" 1.22 "lpd" 0.23 "lpq" 0.36000 "lppd" 0.17 "ll" 0.14 "s1" 0.173 "s12" 0.447 "ra"
↪  0.0000 "rcomp" 0.0000 "xcomp" 0.0000 "accel" 1.0
genrou   3 "3" 22.00 "1 " : #9 mva=900.00   "tpdo" 6.50 "tppdo" 0.079 "tpqo" 0.53 "tppqo" 0.072 "h" 4   "d"
↪  0.0000 "ld" 1.24  "lq" 1.22 "lpd" 0.23 "lpq" 0.36000 "lppd" 0.17 "ll" 0.14 "s1" 0.173 "s12" 0.447 "ra"
↪  0.0000 "rcomp" 0.0000 "xcomp" 0.0000 "accel" 1.0
genrou   4 "4" 22.00 "1 " : #9 mva=900.00   "tpdo" 6.50 "tppdo" 0.079 "tpqo" 0.53 "tppqo" 0.072 "h" 4   "d"
↪  0.0000 "ld" 1.24  "lq" 1.22 "lpd" 0.23 "lpq" 0.36000 "lppd" 0.17 "ll" 0.14 "s1" 0.173 "s12" 0.447 "ra"
↪  0.0000 "rcomp" 0.0000 "xcomp" 0.0000 "accel" 1.0
genrou   5 "5" 22.00 "1 " : #9 mva=900.00   "tpdo" 6.50 "tppdo" 0.079 "tpqo" 0.53 "tppqo" 0.072 "h" 4   "d"
↪  0.0000 "ld" 1.24  "lq" 1.22 "lpd" 0.23 "lpq" 0.36000 "lppd" 0.17 "ll" 0.14 "s1" 0.173 "s12" 0.447 "ra"
↪  0.0000 "rcomp" 0.0000 "xcomp" 0.0000 "accel" 1.0

#         exciters
sexs     1 "1" 22.00 "1 " : #1 0.1 10.0 100.0 0.05 -5.0 5.0 0.08 0.0 -5.0 5.0 0.0
sexs     2 "2" 22.00 "1 " : #1 0.1 10.0 100.0 0.05 -5.0 5.0 0.08 0.0 -5.0 5.0 0.0
sexs     2 "2" 22.00 "2 " : #1 0.1 10.0 100.0 0.05 -5.0 5.0 0.08 0.0 -5.0 5.0 0.0
sexs     3 "3" 22.00 "1 " : #1 0.1 10.0 100.0 0.05 -5.0 5.0 0.08 0.0 -5.0 5.0 0.0
sexs     4 "4" 22.00 "1 " : #1 0.1 10.0 100.0 0.05 -5.0 5.0 0.08 0.0 -5.0 5.0 0.0
sexs     5 "5" 22.00 "1 " : #1 0.1 10.0 100.0 0.05 -5.0 5.0 0.08 0.0 -5.0 5.0 0.0

#        governors
tgov1 1 "1" 22.00 "1 " : #1 mwcap=800.0000 0.050000 0.4 1.000000 0.0   3.0000  10.0000 0.0
tgov1 2 "2" 22.00 "1 " : #1 mwcap=800.0000 0.050000 0.4 1.000000 0.0   3.0000  10.0000 0.0
tgov1 2 "2" 22.00 "2 " : #1 mwcap=800.0000 0.050000 0.4 1.000000 0.0   3.0000  10.0000 0.0
tgov1 3 "3" 22.00 "1 " : #1 mwcap=800.0000 0.050000 0.4 1.000000 0.0   3.0000  10.0000 0.0
tgov1 4 "4" 22.00 "1 " : #1 mwcap=800.0000 0.050000 0.4 1.000000 0.0   3.0000  10.0000 0.0
```

Figure 104: Dyd file used in six machine validations.

10. Code Examples

```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 105: An example of a full .py simulation file.

11. Large Tables

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

Table XII: 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 XIII: 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.