SANDIA REPORT
SAND2020-6469
Printed February 2020

Sandia National Laboratories

A Test Bed for Evaluating Frequency Estimation Algorithms in Synthetic Inertia Control: User Manual

W. Hill Balliet, Felipe Wilches-Bernal, and Josh Wold

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico
87185 and Livermore,
California 94550

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Rd
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: https://classic.ntis.gov/help/order-methods/

ABSTRACT

As penetration of converter interfaced generators (CIGs) increases, the need for CIG frequency control participation increases. Traditionally, research in this area has been performed using positive sequence simulation software, which provides voltage magnitude and phase measurements, but not point-on-wave (POW) measurements. This means that the effect of frequency estimation algorithms cannot be accurately modeled, especially when the voltage waveform is distorted by faults or load connection events. This report serves as a user manual for an electromagnetic transient simulation testbed, which allows for accurate modeling of frequency estimation and control techniques.

ACKNOWLEDGEMENTS

Adam Summers (Sandia National Laboratories) developed a preliminary version of the two-area system used in this project

Patrick Cote (Montana Tech) extended the two-area system and developed a preliminary version of the controllable power injections used in this work

This work was funded by the United States Department of Energy (DoE), Advanced Grid Modeling (AGM) program

CONTENTS

LIST OF FIGURES

LIST OF TABLES

QUICK START

1.) Modify parameters in "`initSim.m`"
   - For many configurations, the only parameters that will need to be changed are in "High level simulation parameters" section. Eg. Simulation length, the choice of synthetic inertia (SI) algorithm, or SI gain
   - For parameters in a vector structure, the elements of the vector represent the value for that parameter for the generator location corresponding to the element's index. Eg. `SI.k = [1, NaN, NaN, 10, 0, 100]` means that CIG1 will use `SI.k = 1`, Generators 2 and 3 cannot be replaced with CIGs, and so on.

2.) Run "`example_KRK_script_6g_faults.m`"
   - Results will be logged in the base workspace
     - Generator speeds: `w_all` <timeseries>
     - CIG data: `SIx` <structure> (x represents the location of the CIG, so `SI1` corresponds to CIG1 data)
     - Simulation configuration parameters: config <structure>
   - Some useful plots will be loaded immediately

3.) Begin exploring code or the rest of this user manual to learn about more advanced features available in the testbed.

ACRONYMS AND DEFINITIONS

Abbreviation	Definition
3ph	3 phase
CIG	converter interfaced generator
EKF	extended Kalman filter
FFT	fast Fourier transform
Hinf	H Infinity
MB-PSS	multi-band power system stabilizer
NLLS	nonlinear least squares
PLL	phase locked loop
POW	point-on-wave
PMU	phasor measurement unit
SI	synthetic inertia
SNR	signal to noise ratio
STG	steam turbine and governor
TKF	Taylor Kalman filter
QPLL	quadrature phase locked loop
UKF	unscented Kalman filter
ZOH	zero order hold

1. SYSTEM REQUIREMENTS

The simulations are computationally intensive, and many useful tests take a non-negligible amount of processing time. For reference, on a machine with 16GB of RAM and a 4-core, 1.9GHz processor, simple simulations take about 20 seconds, while large tests like those needed to create detailed empirical bode plots may take multiple hours. The testbed was designed and tested using the following software.

Table 1: Software specs

Software Type	Version
MATLAB Version Number: `version`	9.2.0.959691 (R2017a) Update 3
Java Version: `version -java`	Java 1.7.0_60-b19 with Oracle Corporation Java HotSpot(TM) 64-Bit Server VM mixed mode
Toolboxes	Computer Vision System Toolbox
	Control System Toolbox
	DO Qualification Kit
	DSP System Toolbox
	Filter Design HDL Coder
	HDL Coder
	Optimization Toolbox
	Signal Processing Toolbox
	Simscape Power Systems
	Simulink
	Symbolic Math Toolbox

This page left blank

2. MODELING DETAILS

2.1. System Block Diagram

Figure 1 shows the topology of the system implemented in the testbed. It is a slightly modified version of the Klein-Rogers-Kundur (KRK) two-area system initially proposed in [1]. The main difference between the system in Figure 1 and the traditional KRK system is that the fourth generator in Area 2 was split into three generators: G4, G5, and G6. In this system, Area 1 includes bus B1 and everything to its left. Area 2, similarly, includes bus B2 and everything to its right. The two areas are linked by two parallel 220km transmission lines. G1, G4, G5, and G6 can all be replaced with CIGs.

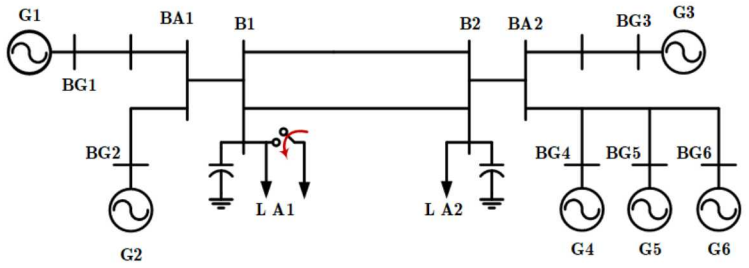

Figure 1: System one-line diagram.

This means that the testbed includes inter-area modes as well as the higher frequency local modes. A detailed view of the GIGs available in the testbed is shown in Figure 5. The block diagram in Figure 2 shows the synchronous generators available in the testbed.

2.2. Synchronous Generators

The Simulink block diagram of the synchronous generator is shown in Figure 2.

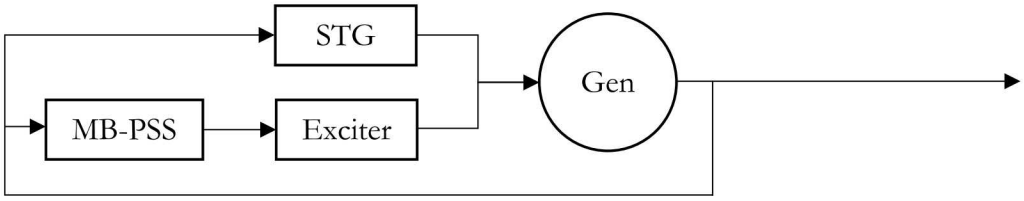

Figure 2: Synchronous generator block diagram

The conventional generators are modeled using the built in Simscape blocks for synchronous generators, steam turbines and governors (STG), and exciters. The exciter is controlled with a multi-band power system stabilizer (MB-PSS) that was tuned so that the shape of the system frequency curve resembles that of an actual power system when the system is subjected to a load connection event as shown in Figure 3. After the load connection event, the system reaches the frequency nadir at 7.5 seconds, and the recovery phase ends around 20 seconds. The frequency response of the MB-PSS is shown in Figure 4 along with the parameters used at the time of the plot.

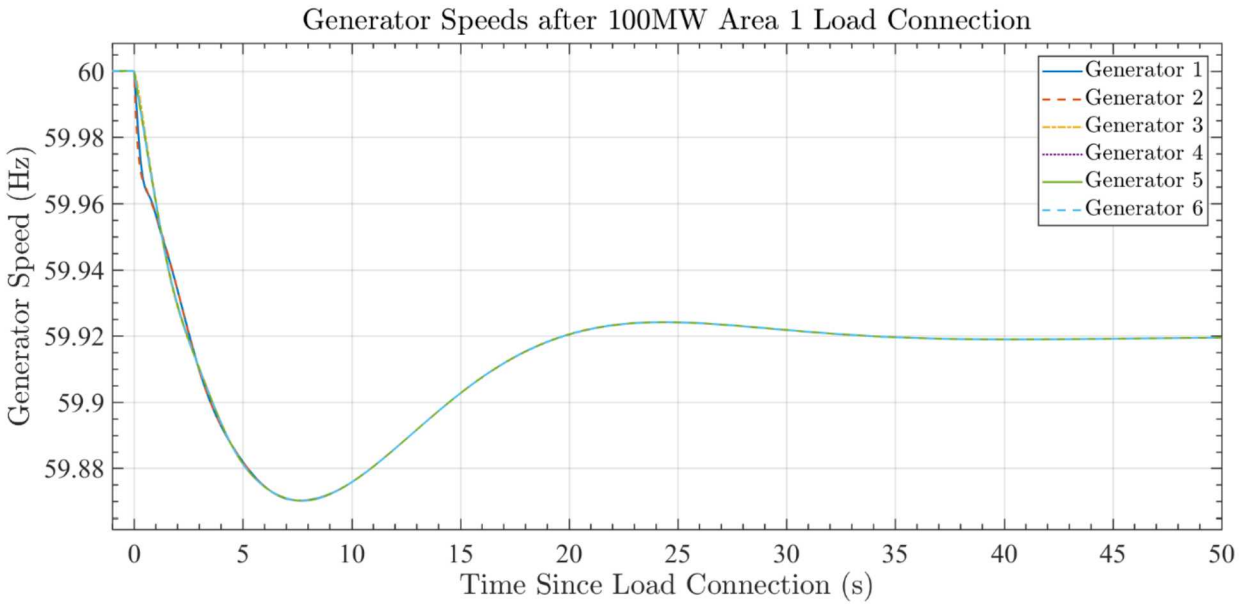

Figure 3: Shape of load connection event

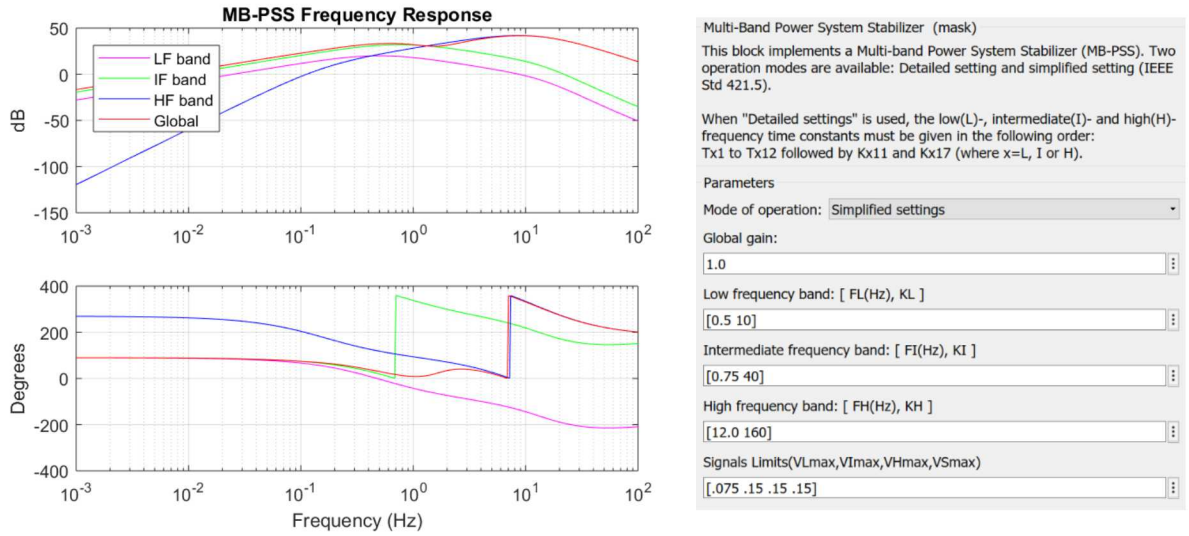

Figure 4: MB-PSS Design

Parasitic loads are included in each generator block to ensure numerical stability of load flow. Mathworks describes the need for this in the Matlab R2017a documentation on the Synchronous Machine block in the "Limitations" section. These parasitic loads are called "Ppara" in the block diagram.

When desired, the governor can be removed for any generator location. This allows the user to compare the dynamics of the system with a synchronous generator and CIG more directly. Since governor droop control can be directly emulated by a CIG, this part of the dynamics is not an inherent difference between the two generation technologies. Removing the STG works by deleting the STG block from the "`G/Turbine & Regulators`" block and then connecting the "`Pref`" input to the "Pm" output. This forces a constant mechanical power at the shaft of the generator,

which means that the system will retain the effect of the generator inertia, but it will not have any effective droop control.

2.3. Converter Interfaced Generators

Figure 5 shows a block diagram of the CIG model. The main components of the CIG are the measurement stage (V/A), the algorithm selection (not pictured), the frequency estimation stage (Freq. Est.), and the output processing stage (Filters through P_{des}).

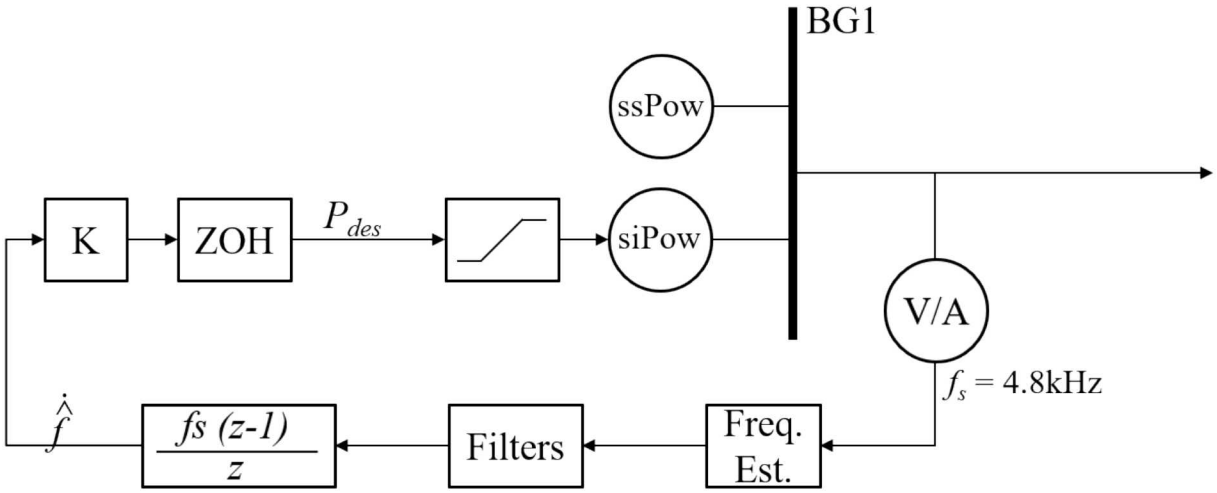

Figure 5: Synthetic inertia control loop block diagram

2.3.1. Measurement Processing

The measurement processing stage is depicted as the "V/A" block in Figure 5. This block samples the voltage and/or current at a specified bus or the generator acceleration or speed at a specified location at the sampling rate specified by `SI.fs`. The Simulink block diagram for the measurement processing stage is shown in Figure 6.

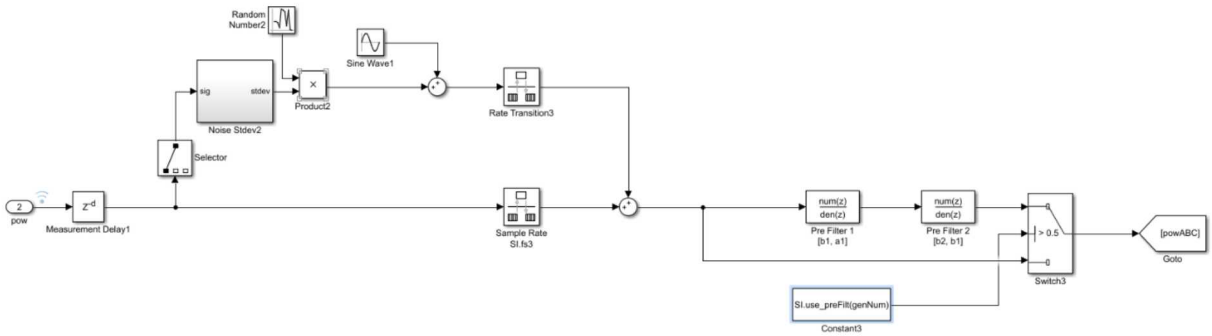

Figure 6: Measurement processing Simulink block diagram

The measurement processing stage allows the user to specify not only the sampling rate, but also the quality of the measurements. This is accomplished through the user specifiable delay, noise injection, sinusoid injection, and digital filtering. The delay functionality is useful for examining the effects of communications delays when using remote measurements, while the noise injection capabilities allow the user to test the performance of the system at different SNR levels (specified in dB) of the input signal. The sinusoid injection allows the user to examine the effects of

electromagnetic coupling the of the measurement signal with nearby signals like the power in the transmission lines at the measurement bus. The "pre-filter" allows the user to test different methods for removing these measurement corruptions.

2.3.2. Algorithm Selection

In order to avoid the computational expense of running every algorithm in every CIG during the simulation, only the chosen algorithm is enabled. The user can select an algorithm using the `SI.alg` variable, and this is generally all that a user interacts with. Sometimes, however, it may be necessary to add a new algorithm to the testbed, in which case it will need to be added to the algorithm selection logic. Since the algorithm selection logic is not entirely obvious in the block diagram, the details are described here.

The algorithm enable inputs are all created using bit manipulation of the `SI.alg` variable. The *enable* signals can be generated in the bits of `enables` with:

$$enables = 1 \ll (SI.alg(genNum) - 1)$$

where $\ll$ is the left bit shift operator and `genNum` is the location of CIG. Therefore, if one were to select algorithm 8, `enables` would be 128, or 00000000000000000000000010000000 in binary. The enable signal for the i^{th} algorithm is then found as the $(i - 1)^{th}$ bit of `enables` counting from the least significant bit. This does not apply to the generator speed or generator acceleration measurement driven SI control scenarios represented by `SI.alg = 1` and `SI.alg = 0` because these measurements do not have enable signals. The estimate of the frequency estimation algorithm is then connected to the system using a switch. The error signal for applicable algorithms is also passed through a switch, but this is hidden in the "Switch Alg Err" block to make the diagram more readable.

2.3.3. Frequency Estimation Algorithms

All of the code for the frequency estimation algorithms can be found in "genLib.slx/Synthetic Inertia/Frequency Estimation Algorithms/".

The one phase algorithms can be used in either 1 or 3 phase mode. If `SI.phase = '3ph'` then the average of the frequency estimate from each phase will be used. Otherwise, if `SI.phase = 'A'` or `'B'` or `'C'` then only the estimate from that phase will be used. In either case, the output of the error signal into the base workspace will be a 3-element vector at each time step. If `SI.phase = '3ph'` then the error for each phase will be represented. Otherwise, the error for the phases that are not being measured will be 0 at each time step. The frequency estimation algorithm options and their corresponding `SI.alg` codes are shown in Table 2.

Table 2: List of available frequency estimation algorithms

Algorithm	`SI.alg` code
Measured Generator Acceleration	0
Measured Generator Speed	1
Phase Locked Loop (PLL) [2]	2
3 Phase PLL [2]	3
Quartz Phase Locked Loop (QPLL) [3]	4
Unscented Kalman Filter (UKF) [4]	5

Algorithm	`SI.alg` code
3 Phase UKF [4]	6
H Infinity (Hinf) [5]	7
3 Phase Hinf [5]	8
Extended Kalman Filter (EKF) [6]	9
3 Phase EKF [6]	10
Clarke Transformation 3 Phase EKF [7,8]	11
Taylor Kalman Filter (TKF) [9,10]	12
3 Phase TKF [9,10]	13
Phasor Measurement Unit (PMU) [11]	14
3 Phase PMU [11]	15
Nonlinear Least Squares (NLLS) [12]	16
3 Phase NLLS [12]	17

In order to maintain the generality of the allowable input signal, the input signal must be normalized. This is accomplished by filling a buffer over the first few seconds of the simulation with the input signal, which is used to determine the normalization factor for the whole simulation. This means that if the buffer is large enough for the signal to come to steady state, then the normalized signal will have an RMS value of 1. The block diagram for determining the normalization factor is shown in Figure 7. The threshold for the switch is the same as the value of "Constant," which evaluated to 1 for this simulation. Similarly, the number of samples in the buffer is set using the `SI.normWindow` variable.

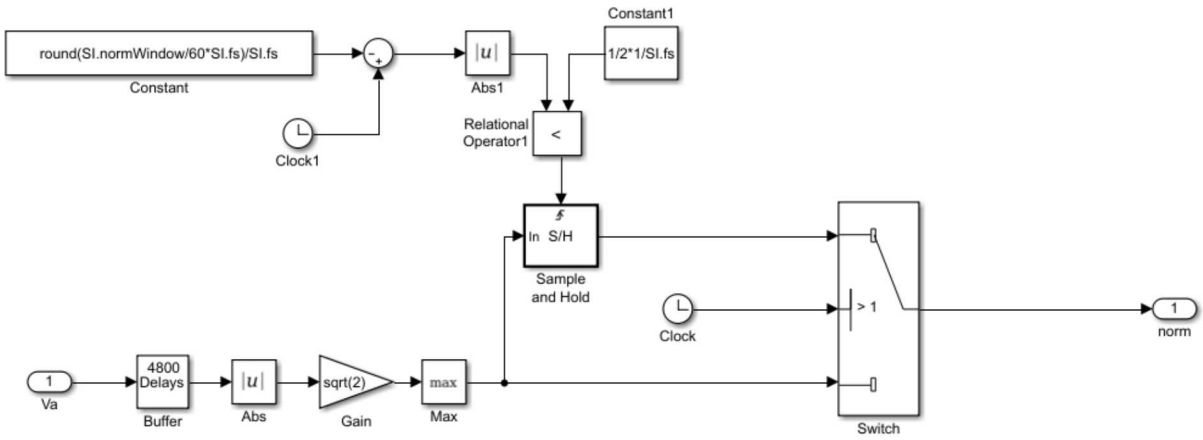

Figure 7: Logic for determining the normalization factor for a frequency estimation algorithm input signal

2.3.4. Post Processing

After the frequency estimation algorithms, there are two optional digital filters, a discrete derivative filter, a gain, and a zero order hold as shown in Figure 5. The gain is set with `SI.k`, and the zero

order hold brings the control signal from the algorithm sampled frequency, `SI.fs`, to the simulation frequency.

The two optional filters are determined by the coefficients of their transfer functions. These coefficients are generated in the "`buildFilters`" function of `initSim.m`. They can be turned on and off using `SI.use_postFilt` and `SI.use_controller`. The controller filter is applied first, and then the post filter is applied to the resulting signal. While the filters could be combined into one filter, splitting them into two parts can help with numerical precision issues. For instance, if there are many poles and zeros, the coefficients may not have enough precision to accurately reproduce them. This is especially true if there are many poles and zeros near `z=1`.

2.4. Solver

In order to obtain the point-on-wave (POW) measurements, the solver for the Simscape portion of the system, specified in the "Powergui" block, cannot be "`Phasor`." Because of the implementation of the CIG block, the "Powergui" solver must be discrete although this could be changed with some effort. The benefit of implementing it as a discrete process is that in reality, the frequency estimation algorithms would be implemented using a digital processor, so a discrete simulation is more realistic. This means that the Simulink solver must also be discrete.

2.5. Transformers

The transformers in the simulation are implemented with delta-wye, three-phase transformer (two windings) blocks from Simscape. The transformers step up the voltage from the generators and loads for the transmission lines to reduce losses. Snubbers are used at each transformer to limit the voltage transients in the system.

2.6. Transmission Lines

The shorter transmission lines, within each area are modeled as a pi section line, which simplifies the necessary calculations without much impact to accuracy. The longer transmission lines, between each area, are modeled as distributed lines since they are long enough that pi section models would not be accurate enough. The transmission line parameters are the same as in the `power_KundurTwoAreaSystem` example provided by Matlab.

3. ADVANCED TESTING

All the advanced testing scripts are found in the "Test Scripts" folder, shown in Table 11, Section 4.4. This section offers a description of how to use these scripts and what they are doing under the hood.

3.1. User Configurable Settings

All of the easily user configurable settings are assigned in "`initSim.m`" and "`initAlgs.m`." These settings can be set directly for single simulations or a set of simulation configurations can be specified using one of the test case input methods. The parameters in the first section of "`initSim.m`" are easily understood, do not have hidden interactions with other important parameters of the simulation, and are useful in analyzing the performance of the physical system under different conditions.

NOTE: The parameters in the later sections after this one offers a great deal of control over the simulation but may require a deeper understanding of the back-end implementation to use successfully and avoid unintended consequences.

3.1.1. Generators and Steady State Power Flow

The parameters that affect the generators and steady state power flow are described in Table 3.

Table 3: Summary of generator and power flow parameters

Parameter	Function	Example
`replaceGens`	Specifies the type of generator at G1, G4, G5, and G6. Locations in this array will be instantiated as CIGs, and all others will be synchronous generators.	`replaceGens = [1, 4, 6];` G1, G4, and G6 will be instantiated as CIGs and G2, G3, and G5 will be synchronous generators.
`removeSTGs`	Specifies generator locations which should not have steam turbine and governors. The frequency response at those locations will be limited to the generators' inertial responses.	`removeSTGs = [1, 2];` G1 and G2 will have no governor response. This only makes sense if G1 and G2 are synchronous generators.
`Hs`	The inertia of each generator is specified in p.u. by the elements of "`Hs`."	`Hs = [2, 1, 2, 2, 2, 2];` The inertia of G2 is 1p.u. and the inertia of all other generators is 2p.u. This only affects synchronous generators.
`nomPows`	Sets the nominal power in watts of each generator.	`nomPows = [1e9, 2e9, 1e9, 5e8, 2e8, 2e8];` The nominal power of G1 is 1e9 watts, the nominal power of G2 is 2e9 watts, and so on.
`ssPows`	Specifies the steady state powers of each generator except G2. Generator 2 is a swing bus, so its steady state power cannot be directly specified. However, it can be indirectly specified by picking the correct total system load, which is done automatically in "`initSim.m`" based on "`Pg2_desired`." The steady state reactive power for the CIGs is automatically calculated by running load flow on the system with all synchronous generators in place. Then, any CIGs will be assigned the steady state reactive power calculated for the generator in that position.	`ssPows = [1e8, 0, 2e8, 5e7, 2e7, 2e7];` The steady state power of G1 is 1e8 watts, the steady state power of G3 is 2e8 watts, and so on. Note: the steady state power of G2 is not necessarily 0. This is determined by the load flow and can be indirectly set using `Pg2_desired`.

`Pg2_desired`	Used to indirectly specify the steady state power of G2. This is done automatically in "`initSim.m`" by calculating the total load in the system (before any load connection events) that would require `Pg2_desired` watts more than is provided by G1 + G3 + G4 + G5 + G6.	`Pg2_desired = 2e6;` The steady state power of G2 will be approximately 2e6 watts. This also dictates how much load is in the system so that total generation matches total load.
`fracInA1`	Specifies the distribution of the load in the system. This dictates the power flow in the system, e.g. if the local generation matches the local load in both areas, then there will be no power transfer between areas at steady state.	`fracInA1 = 0.3;` While the total load power is determined by Pg2_desired, 30% of it will be in area 1, and 70% will be in area 2. If generation is distributed evenly between areas, then power will flow from area 1 to area 2 at steady state.

3.1.2. *Synthetic Inertia Control*

Table 4: Summary of user configurable parameters for SI control

Parameter	Function	Example
`inSI*`	Specifies the measurements that should be used for synthetic inertia.	`inSIgs_CIG1 = 'w2';` This means that CIG1 will use the generator speed measurements from G2 for SI control if the SI alg is set to generator speed.
`SI.alg`	Specifies which number algorithm should be used for the SI control. Can also specify the use of generator speed or generator acceleration measurements for the control signal. A list of which number corresponds to which algorithm can be found in `initSim.m`.	`SI.alg = 10*ones(1,6);` This means that all CIGs will use the EKF to create a frequency estimate from POW data.
`SI.k`	The gain used for SI control, denoted K in Figure 5.	`SI.k = [10, NaN, NaN, 0, 0, 0];` CIG1 will use a gain of 10 to generate the final control signal, but all other CIGs will use a gain of 0.
`SI.phase`	The phase which the POW data should be sampled from.	`SI.phase = 'A';` The POW data used for frequency estimation will be sampled from phase A of the bus specified by `inSI_CIG*`.
`SI.use_prefilt`	Turns the prefilter on or off. Prefilters are applied to the POW data before it is used to create a frequency estimate.	`SI.use_prefilt = [true, NaN, NaN, false, false, false];` CIG1 will use the prefilter, but none of the other CIGs will use the prefilter.
`SI.prefilt`	Sets whether the prefilter will be 1: a Butterworth filter 2: a Chebyshev filter 3: both filters	`SI.preFilt = ones(1,6)*3;` All CIGs will use both a Butterworth filter and a Chebyshev filter.

`SI.use_postfilt`	Turns the post filter on or off for each CIG. The post filter is applied to the frequency estimate as shown with the "Filters" block in Figure 5. This is applied immediately after the controller and is therefore a good place for a low pass filter.	`SI.use_postFilt = [false, NaN, NaN, false, true, false];` CIG5 will use the postfilter, but none of the other CIGs will use the postfilter.
`SI.use_controller`	Turns the controller on or off for each CIG. The controller is applied to the frequency estimate as shown with the "Filters" block in Figure 5. This is applied before the postfilter. If the controller has many poles, there may be numerical inaccuracies that introduce noise. This means it is often better to separate any low pass filtering and apply it with the postfilter.	`SI.use_controller = [false, NaN, NaN, false, true, true];` CIG5 and CIG6 will use the postfilter, but none of the other CIGs will use the postfilter.
`SI.use_corr`	Turns the correction algorithm on or off for each CIG.	`SI.use_corr = [false, NaN, NaN, true, true, true];` CIG4, CIG5, and CIG6 will use the correction algorithm but CIG1 will not.
`SI.fDelay`	Specifies the length of the initialization period for the frequency estimate. During initialization, the frequency estimate will be set to 1 p.u. until the estimate becomes more accurate.	`SI.fdelay = 1.5;` The frequency estimate will be set to 1 p.u. for the first 1.5 seconds of the simulation.
`SI.siOn`	Specifies the time when the synthetic inertia control should be turned on. This allows for transients in the frequency estimate from initialization to settle before any control action is taken.	`SI.siOn = 2;` The SI control will not start injecting power until 2 seconds into the simulation.
`SI.normWindow`	Specifies the number of cycles that should be used to create the normalization factor. The normalization factor is applied to the POW data before frequency estimation so that the algorithms can be applied to any signal that carries the system frequency such as current and voltage.	`SI.normWindow = 60;` The normalization factor will be created by measuring the RMS value of the signal over the first 60 cycles.

`SI.fs`	The sampling frequency for the POW data. Usually this is 4.8 kHz.	`SI.fs = 4800;` The POW data will be sampled at 4.8 kHz.
`SI.delay`	The delay in the POW data. This can be used to simulate delay caused by measurement devices and communications.	`SI.delay = 0.01;` The POW data used for frequency estimation will be delayed by 0.01 seconds.
`SI.injectNoise`	Specifies whether noise should be injected into the POW measurements for each CIG.	`SI.injectNoise = [false, NaN, NaN, false, true, false];` Noise will be injected for the POW data received by CIG5, but not for any other CIG.
`SI.SNR`	Specifies the signal to noise ratio (SNR) for the POW data after noise injection if it is enabled.	`SI.SNR = [80, 80, 80, 80, 80, 80];` The SNR for POW data received by each CIG will be 80 dB if noise injection is enabled for that CIG.
`SI.overshootAmp`	The amplitude of a sinusoidal injection into the POW data in p.u. Setting this to 0 will turn it off. This can be used to simulate signal corruption from coupling in the measurement device or communications network.	`SI.overshootAmp = 0.01;` A sinusoid with amplitude 0.01 p.u. will be injected into the POW data.
`SI.overshootFreq`	The frequency of a sinusoidal injection into the POW data in rad/s. This feature can be used to simulate signal corruption from coupling in the measurement device or communications network. It can be turned on and off using the `SI.overshootAmp` variable.	`SI.overshootFreq = 8*2*pi;` A sinusoid with frequency 8*2*pi rad/s will be injected into the POW data if `SI.overshootAmp` is not 0.
`SI.rateLimit`	Sets a rate limit on the output power of the SI control. This can be set to `Inf` to remove the rate limit.	`SI.rateLimit = 10;` Each CIG is limited to changing its output power by at most 10 watts per second.

`SI.saturation`	Sets the saturation point for the output power of the CIGs as a multiple of the nominal power of each CIG. This can be set to `Inf` to turn off saturation.	`SI.saturation = 1.1;` Each CIG will be limited to producing or consuming 1.1 times its nominal power.

`a1, b1`	a1: denominator coefficients
	b1: numerator coefficients
	
	for the transfer function of the prefilter (used for Butterworth filter)
		`a1 = [1 2];`
		`b1 = 1;`
		
		The transfer function of the first prefilter would be
		$\frac{1}{s+2}$

`a2, b2`	a2: denominator coefficients
	b2: numerator coefficients
	
	for the transfer function of the prefilter (used for Chebyshev filter)
		`a2 = [1 2];`
		`b2 = 1;`
		
		The transfer function of the second prefilter would be
		$\frac{1}{s+2}$

`a3, b3`	a3: denominator coefficients
	b3: numerator coefficients
	
	for the transfer function of the postfilter
		`a3 = [1 2];`
		`b3 = 1;`
		
		The transfer function of the postfilter would be
		$\frac{1}{s+2}$

`a4, b4`	a4: denominator coefficients
	b4: numerator coefficients
	
	for the transfer function of the first filter in the controller
		`a4 = [1 2];`
		`b4 = 1;`
		
		The transfer function of the first controller filter would be
		$\frac{1}{s+2}$

`a5, b5`	a5: denominator coefficients
	b5: numerator coefficients
	
	for the transfer function of the second filter in the controller
		`a5 = [1 2];`
		`b5 = 1;`
		
		The transfer function of the second controller filter would be
		$\frac{1}{s+2}$

3.1.3. Frequency Estimation Algorithm Parameters

User configurable parameters for the algorithms listed in Table 2 are found in "`initAlgs.m`" and are split up by algorithm. They are stored in the `SI` structure, so the extended Kalman filter (EKF) parameters would be in `SI.EKF` for example. Modifying these parameters effectively requires an understanding of the algorithm in question.

The frequency correction algorithm can be switched on and off using the "SI.use_corr" variable, and its parameters are specified in "`initAlgs.m`." This correction algorithm is useful for eliminating large transient errors in frequency estimates that can occur during faults and other waveform distorting events.

If more detailed modifications are required, then you can modify the algorithms directly in "genLib.slx/Synthetic Inertia/Frequency Estimation Algorithms/"

3.1.4. Simulation and Event Settings

Table 5: Summary of general simulation and system event parameters

Parameter	Function	Example
`tfstart`	Specifies the time that the system event should begin. This means that either a load is connected at this time or a fault begins at this time.	`tfstart = 5;` A load connection event or a fault will occur beginning at 5 seconds in the simulation.
`simLength`	The length of the simulation in seconds.	`simLength = 15;` The simulation will run for 15 seconds.
`Ts`	The simulation is run using a discrete solver, so this specifies the size of the simulation time step. This should be a multiple of `SI.fs` for accurate results.	`Ts = 1/24000;` The simulation time step will be 1/2400 seconds.
`fault_type`	Specifies what type of system event will occur. LC1 - Load connection in Area 1 LC2 - Load connection in Area 2 ABCG - Three phase to ground fault ABG - Two phase to ground fault AG - Single phase to ground fault AB - Two phase fault ABC - Three phase fault	`fault_type = 'LC1';` A load connection event will occur in area 1.
`Lconn_a1`	Specifies the size of the load to be connected in area 1 in the event of a load connection event there.	`Lconn_a1 = 100e6;` If a load connection event occurs in area 1, it will be 100 MW.
`Lconn_a2`	Specifies the size of the load to be connected in area 2 in the event of a load connection event there.	`Lconn_a2 = 100e6;` If a load connection event occurs in area 2, it will be 100 MW.
`tdurload`	If there is a load connection event, this specifies how long the load should remain connected to the grid.	`tdurLoad = 1;` The load will be disconnected 1 second after it is connected.
`tdurshort`	If there is a fault in the grid, this specifies how long the fault should persist before it is cleared.	`tdurShort = 0.1;` The fault will be cleared after 0.1 seconds.

3.2. Test Case Input Methods

These scripts work by running `initSim`, before each test case is run, and then overwriting the specified parameters with the specified values. This places some limitations on the parameters that can be modified. For instance, in "initSim.m," the load powers are calculated to yield a desired output power at the swing bus (G2). This is useful for being able to easily guarantee the percentage of generation that is handled at each generator. If the load powers are changed in one of the test scripts, then the swing bus will output a different power than specified, in turn changing the share of the generation at each generator. The other limitation is that if the reactive power flow in the system is affected by a parameter change in a test script, then any converter interfaced generators (CIGs) in the system will no longer produce the correct amount of reactive power. Also note that to change the inertia of generators, the correct element of "`Hs`" must be modified. Modifying the variable "`H2`" will not accomplish anything, but modifying "`Hs(2)`" will.

Any of the scripts in the format "`test*MaxGain.m`" allow for the same method of input specification as their non-max gain counterparts. The only difference is that the script will find the maximum stable gain of the test scenario and save the specified results using that gain.

3.2.1. Case by Case

Using "`Test Scripts/testEachCase.m`," one can specify a set of tests case by case in the code. The parameters in "`initSim.m`" will form the default parameters, but any parameters listed in the "`allTests`" structure will be overwritten. The results to be saved for each test can also be specified.

This script is useful for specifying a small set of tests where more than one parameter is being varied. If only one parameter is being changed, it may be easier to use "`testSensitivityToParam.m`," and if there is a large set of tests, it may be easier to use "`testSpreadsheet.m`."

3.2.2. Parameter Range

Using "`Test Scripts/testSensitivityToParam.m`," one can specify a set of tests as a list of values for a specific parameter. For example, one can test the performance of all algorithms under the same configuration using `testVars('SI.alg') = 0:17;`. The parameters in "`initSim.m`" will form the default parameters, but any parameters listed in the "`allTests`" structure will be overwritten. The results to be saved for each test can also be specified.

This script is useful for specifying a set of tests where one parameter is being varied. If more than one parameter is being changed and there are a small number of test cases, it may be easier to use "`testSensitivityToParam.m`," but if there is a large set of tests, it may be easier to use "`testSpreadsheet.m`."

3.2.3. Spreadsheet

Using "`Test Scripts/testSpreadsheet.m`," one can specify a set of tests using a spreadsheet. The parameters in "`initSim.m`" will form the default parameters, but any parameters

listed in the spreadsheet will be overwritten. The results to be saved for each test can also be specified.

This script is useful for specifying a large set of tests where more than one parameter is being varied. If only one parameter is being changed, it may be easier to use "`testSensitivityToParam.m`," and if there is a small set of tests, it may be easier to use "`testEachCase.m`." Spreadsheets can be in .csv, .xls, or .xlsx format as long as Microsoft Excel is installed on the machine. An example of valid spreadsheet content is shown in Table 6. The name of each test case is in the first row, and the parameter to be overwritten is in the first column. The values are in the middle of the table, and these can be strings or doubles.

Table 6: Example of valid spreadsheet format for "testSpreadsheet.m"

	local_V_PLL	B3_V_PLL	BA2_V_PLL
inSI_CCD6	Vabc_G6	Vabc_B3	Vabc_A2
SI.alg	2	2	2
SI.use_postFilt	1	1	1
PLL_useFilt	0	0	0
PLL_3ph_useFilt	0	0	0

3.3. Empirical Bode Plots

Since this is not a linear system due to the generator and transmission line models, the empirical Bode plot does not perfectly describe the system's dynamics. In fact, our research shows that most input signals above 40Hz excite more than one frequency in the output, and the nonlinear simulations reveal a resonance at 120Hz that is not adequately described by the Bode plot. However, these Bode plots represent the system with enough accuracy to be useful as a tool for designing a compensator for the system.

Matlab and Simulink's system linearization tools do not work on the testbed because there are too many interdependent blocks in the system. As such, creating empirical bode plots has been the only way we have found to create a linear approximation of the system.

3.4. Maximum Stable Gain

Modify "`initSim.m`" to reflect the desired baseline simulation parameters that you would like to test the performance of. Run "`initSim`" and "`completeModel`", then run "`findMaxGain`" to find the maximum stable gain for the environment variables specified in `initSim.m`. This can also be used in combination with one of the scripts described in the Test Case Input Methods section.

NOTE: This script was designed to test max stable gain for a load connection event.

This script is useful for programmatically finding the maximum stable gain for a configuration or set of configurations of the system. This reduces the human effort needed to compare the performance of a set of configurations at their maximum stable gains.

The script uses a binary search to find the maximum gain. The test condition of the binary search is whether the simulation is stable at a gain `K`. If not, then `K` is larger than the maximum gain. Binary search is also known as bisection search, but if you are unfamiliar, more information about the binary search algorithm can be found here.

There are three ways that the script tests for system instability:

- Simulation must run to completion

  If the simulation does not run to completion, then there was an issue with the configuration. An example of a results set that would fail this test is shown in Figure 8. Note that the simulation length was 15 seconds and the load connection event occurred at 5 seconds.

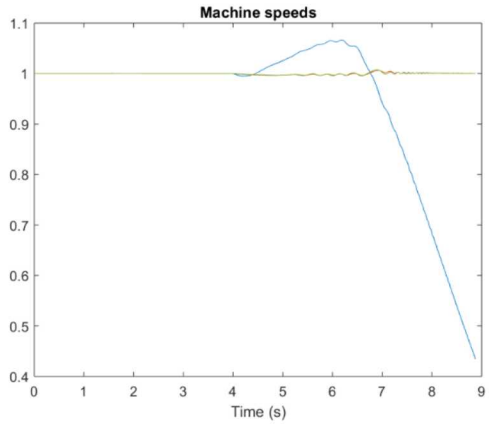

Figure 8: Simulation ends early due to instability

- Synchronism must be maintained

  It is uncommon for a system to fail because of this point. This is because usually when the system loses synchronism, it does not regain synchronism, but rather goes off to infinity, which causes the simulation to end early as shown in Figure 8. Synchronism here is tested by checking if all generators are maintaining the same frequency to within 0.1 pu (6 Hz).

- There must not be significant oscillations induced in the generator speeds

  This condition is tested using the fast Fourier transform (FFT) of the generator speeds. If there is a peak with a prominence of at least 2e-6 above 5 Hz, then the oscillations are considered significant. An example of a results set that fails this test is shown in Figure 9.

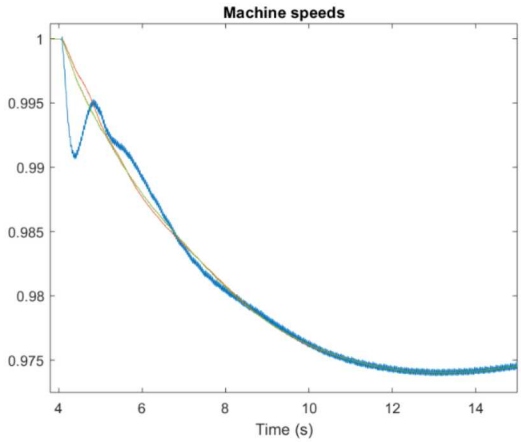

Figure 9: SI induced high frequency oscillations

3.4.1. Generating Data

1.) Set the bode plot generation parameters in "`Test Scripts/example_BodePlot.m`" and note the value of `outSig`. This determines where the output of the system is measured and affects whether dynamics like the derivative filter are included. Also set the system configuration parameters here or in `initSim.m`. The most important parameters to set for this are `SI.alg`, the traditional inertia parameters, and the `SI` input parameters. All of these will have significant effects on the empirical bode plot. Be sure to set the desired filtering in the system appropriately. Turning the filters off is usually best for design since it allows the most modularity.

2.) Generate the empirical bode plot by running "`Test Scripts/example_BodePlot.m`." If there are many frequencies in the "`fmods`" parameter, then this will take a long time. Also note that low frequency tests take longer than high frequency tests because a whole number of cycles must be injected.

Once you have generated the empirical bode plot data, you can use it to design a compensator or you can use it to predict the performance of the system with a different algorithm. This prediction is only accurate when model nonlinearities are not significant, so be sure to check the output csv file to determine the range of frequencies that this is useful for. To perform this prediction, use "`make_system_bode.m`" to remove the original algorithm's frequency response. Then, the resulting system bode plot can be composed with a different algorithm's frequency response, generated with "`make_algorithm_bode.m`" to predict the system frequency response with that algorithm.

The empirical Bode plots are generated by setting the compensator transfer function to 1 and breaking the loop in Figure 10 at P_{des} and the location specified in the "`outSig`" variable. Then, the open loop system is probed at a range of frequencies by applying a sinusoidal input at P_{des} and measuring the output. The frequency response of the system at each input frequency is then calculated by taking the ratio of the magnitude of the Fourier transform of the output at the input frequency to that of the input: `G.tf(1,k)=OUT(idxout)/IN(idxin);`. Where `OUT` is the FFT of the output signal, `IN` is the FFT of the input signal, and `idxin` is the index of the input frequency. `G` is the resulting frequency response.

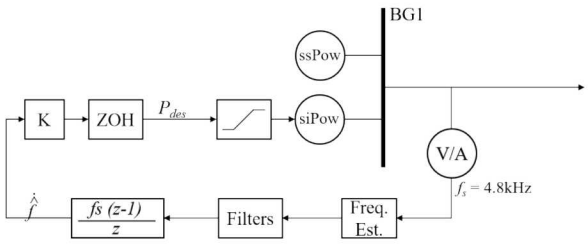

Figure 10: Synthetic inertia structure

3.4.2. Model Nonlinearities

Since this is a nonlinear model, an empirical bode plot will not capture all of the dynamics of the system. In many cases, a pure sinusoid input with frequency ω will induce an output with multiple frequency components. This is detected by examining the FFT of the output. Every peak with prominence at least ¼ as large as the prominence of the peak at ω is recorded in a .csv file. This enables the user to see under which conditions the nonlinearities of the system have a significant effect on the steady state output.

3.4.3. Designing Compensators

A graphical tool for designing a system compensator is provided in "`Bode Design/ example_BodeDesigner.m`." Instruction for using this tool are as follows:

1.) Set the bode plot generation parameters in "`Test Scripts/example_BodePlot.m`" and note the value of `outSig`. This determines where the output of the system is measured and affects whether dynamics like the derivative filter are included. Also set the system configuration parameters here or in `initSim.m`. The most important parameters to set for this are `SI.alg`, the traditional inertia parameters, and the `SI` input parameters. All of these will have significant effects on the empirical bode plot. Be sure to set the desired filtering in the system appropriately. Turning the filters off is usually best for design since it allows the most modularity.

2.) Generate the empirical bode plot by running "`Test Scripts/example_BodePlot.m`." If there are many frequencies in the "`fmods`" parameter, then this will take a long time. Also note that low frequency tests take longer than high frequency tests because a whole number of cycles must be injected.

3.) Run "`Bode Design/ example_BodeDesigner.m`" to open a graphical interface for designing the compensator through loop shaping. If any filters should be accounted for or added to the empirical frequency response, be sure they are included in the argument of `SYS = frd(…);`. This can be accomplished by calculating the frequency response of the filter in question at the same frequencies as the empirical response and then combining them using "`.*`".

4.) Once you have finished your design, change the corresponding code in the "`buildFilters`" function of "`initSim.m`" to initialize `a1` through `a5` and `b1` through `b5` correctly. Note that

if there are a large number of poles and zeros, you may need to split the filter into two parts to avoid negative effects from numerical precision limitations.

3.5. Results Analysis

The tools for analyzing simulation results can be found in the "Simulation Results" folder. This is also where the test scripts save their results by default.

3.6. Comparing System Configurations

There are many cases where one might want to be able to save a snapshot of the simulation parameters for a specific configuration as an identifier for the simulation's results. For example, this allows the user to recreate the simulation later to save a different variable or record the data in higher resolution. Such a snapshot can be found in the "`config`" variable in the base workspace after each simulation.

It may also be useful to compare two different results sets and understand what caused the difference between them. This process is significantly accelerated by using the "Simulation Results/compareConfigs.m" function. This function takes two config variables as inputs and returns a copy of each variable that only retains the differences between them. This way the user does not need to sift through the large number of identical parameters stored in the config structures to find the differences.

3.7. Saving Downsampled Data

When simulations are run at 24kHz, even short simulations create a large amount of data, on the order of 5 Gb. For most applications, 24kHz resolution is unnecessary, so a function is provided in "Simulation Results/downsampleSimResults.m," which downsamples any timeseries data to the desired sampling frequency. This function can be used to save space and time.

This page left blank

4. PROJECT CODE MAP

Provides a high-level description of the function of each .m file in the project.

Table 7: Top level files

File Name	Description
completeModel.m	This script updates the simulink block diagram to reflect the system parameters selected in `initSim.m`
example_KRK_script_6g_faults.m	This script runs the testbed for the configuration specified in `initSim` and `initAlgs`.
genLib.slx	Contains the library models for the traditional/converter interfaced generation blocks.
initSim.m	System parameters are set in this file.
initAlgs.m	Frequency estimation algorithm parameters are set in this file.
KRK_17a_6g_useN.slx	The Simulink model of the entire system, which is modified by "`completeModel.m`" using "`genLib.slx`."
powerlib_Mod.slx	Modified version of the system library that allows direct measurement of the synchronous generator's acceleration, when paired with "`spsSynchronousMachineModel_Mod.slx`".
spsSynchronousMachineModel_Mod.slx	Modified version of the system library that allows direct measurement of the synchronous generator's acceleration, when paired with "`powerlib_Mod.slx`".
quickStart_example.m	A file with a few built-in examples of how the testbed might be used. This file is the same structure as "testEachCase.m," but includes a few useful examples.

4.1. BODE DESIGN/

This folder contains tools for designing compensators with empirical Bode plots. The tools for generating the data for these plots are found in the "Test Scripts" folder.

Table 8: Bode design files

File Name	Description
create_synthetic_data2.m	Creates three-phase synthetic data with different types of corruption such as: phase imbalances, distortion, amplitude modulation, frequency modulation and noise. (Used by `make_pm_sigs_fun.m`)
example_BodeDesigner.m	Useful tool for designing a compensator for the system based on an empirical bode plot. Uses MATLAB's `controlSystemDesigner`.
filters_comp.m	Processes the output of "`freq_est_EKF2.m`" before returning results.
freq_est_*.m	These functions run a frequency estimation algorithm on the input data. They are used to create algorithm bode plots in "`make_algorithm_bode.m`."
init_calc.m	Used in "`freq_est_NLLS_1ph.m`" to determine algorithm initial conditions.
make_algorithm_bode.m	Creates a bode plot of a frequency estimation algorithm by itself.
make_pm_sigs_fun.m	This function creates test signals with sinusoidally varying frequency for use in creating algorithm frequency response plots.
make_system_bode.m	Creates a system bode plot that theoretically excludes the frequency estimation algorithm's effect on the system.
setup_*.m	Initializes algorithm parameters for the matching "`freq_est_*.m`" function.
yhat_J_fun.m	Auxiliary function for NLLS 1phase algorithm.

4.2. ESTIMATION ALGORITHMS/

This is an archive of the estimation algorithms provided at the beginning of June 2019. It is used in "`testMergeReadiness.m`" to check if the algorithms in the simulation testbed perform the same as the algorithms outside of the testbed.

NOTE: Discrepancies between these two may be caused by updates to the algorithms in the simulation testbed, and the algorithms in the simulation testbed should be assumed to be the more accurate versions at the time of this archive.

Table 9: Estimation algorithm files

File Name	Description
alg_selector.m	A function that selects, initializes, and runs the chosen frequency estimation algorithm.
alphabetaT.m	Helper function for the UKF 3 phase (3ph) and TKF 3ph algorithms that conditions the input.
example_test_frestalg.m	Provides a user-friendly wrapper to `alg_selector` and plots the results compared to the expected results (when using spreadsheet data with known true values).
freq_est_*.m	These functions run a frequency estimation algorithm on the input data.
init_calc.m	Used in "`freq_est_NLLS_1ph.m`" to determine algorithm initial conditions.
init_calc_3ph.m	Used in "`freq_est_NLLS_3ph.m`" to determine algorithm initial conditions.
sim_*.slx	The Simulink block diagrams for the frequency estimation algorithms that are implemented in block diagram rather than code form. These are used in their respective "`freq_est_*.m`" files.
sim_example_test_frestalg.m	Like "`example_test_frestalg.m`," but modified for use by "Test Scripts/testMergeReadiness.m."
syntheticdat_reader.m	Reads .xlsx files containing POW data.
yhat_J_fun.m	Auxiliary function for NLLS 1 phase algorithm.
yhat_J_fun_3ph.m	Auxiliary function for NLLS 3ph algorithm.

4.3. SIMULATION RESULTS/

Simulation results are saved here when using the test script files. This folder also contains tools for analyzing simulation results.

Table 10: Simulation results files

File Name	Description
compareConfigs.m	Compares two structures containing the system configuration variables and creates two new structures, which only contain the differences between the originals.
cursor_pointsAmp.m	Finds the amplitude of a sine wave given an exported `cursor_info` structure. This structure is created by selecting the peak and trough of the wave with the datatip tool on a plot and then exporting the data to the workspace.
cursor_pointsFreq.m	Finds the amplitude of a sine wave given an exported `cursor_info` structure. This structure is created by selecting two adjacent peaks of the wave with the datatip tool on a plot and then exporting the data to the workspace.
downsampleSimResults.m	Downsamples time series data to the specified `fs` to save space when saving large amounts of data.
plotBodeComparison.m	Plots multiple bode plots on the same axes for comparison. Data is drawn from a list of .mat files.

4.4. TEST SCRIPTS/

This folder contains test scripts that allow for easy specification of system configurations. The results of these test scripts are saved in the "Simulation Results" folder. Use these tools to collect data for later analysis.

Table 11: Test script files

File Name	Description
empiricalBodePlot.m	Base script for creating an empirical bode plot.
example_BodePlot.m	Uses `empiricalBodePlot` to create an empirical bode plot for the system configuration specified in `initSim.m` and `initAlgs.m`. Parameters specified in this file overwrite other definitions.
findMaxGain.m	Running this after running `initSim.m` and `completeModel.m` finds the maximum stable gain for the system configuration.
testEachCase.m	Allows the user to specify a set of test scenarios that are then run, and the results are saved to the specified file.
testEachCaseMaxGain.m	Allows the user to specify a set of test scenarios to find the maximum stable gain for. These tests are then run, and the results are saved to the specified file.
testMergeReadiness.m	Compares the outputs of the frequency estimation algorithms in the simulation testbed to the outputs of the algorithms in the "Estimation Algorithms" folder for the same input.
testSensitivityToParams.m	Allows the user to specify a range of values for a parameter to test the system at, and the results are saved to the specified file.
testSpreadsheet.m	Allows the user to specify a set of test scenarios using a spreadsheet in .xsl, .xslx, or .csv which are run and the results are saved.
testSpreadsheetMaxGain.m	Allows the user to specify a set of test scenarios to find the maximum stable gain for. The tests are specified using a spreadsheet in .xsl, .xslx, or .csv, and they are run and the results are saved.

This page left blank

5. EXAMPLE

The code base includes a few pre-built examples for quickly familiarizing oneself with using the testbed. These can be found in "quickStart_example.m." This file can be used the same way as "testEachCase.m," but it includes some examples that show how some of the most commonly used features can be modified. These examples include

1.) testing a system configuration under different fault conditions.

2.) testing a synthetic inertia configuration for different system power flow scenarios.

3.) testing a system configuration under different synthetic inertia controls.

Each of these examples have descriptions in the "quickStart_example.m" file, and this section goes though the process of obtaining and interpreting the results of example 3.

The first step to running an example is to set the `whichExample` variable to the name of the example you would like to run. Then, the `filename` variable should be set to the desired output file name (do not include a file extension in the name) on line 162. For example 3, set `whichExample` on line 40 to "faultConditions" and set `filename` to "test" before running the script. Once the simulation finishes, the command window will output "Done with simulations!!!!"

Your data can then be accessed in "Simulation Results/test.mat." To analyze your data, you can navigate to the "Simulation Results" folder in MATLAB, and run the command `load test.mat`, which will load your simulation results into a map called "Results" in the base workspace. Typing `Results.keys` into the command window will print out the data series that are saved in this variable. The first part of the printed variable name shows which test the data series is from, and the second part shows which variable is saved from that test. For example, `ABfault_config` is the `config` structure from the "ABfault" test.

To confirm the differences in the system configuration between each test case, you can use the "compareConfigs" function as shown in Figure 11. For this example, the variables that changed between each test are `Lconn_a1`, `fault_type`, `tab_off`, `tab_on`, and `tdis_Lc_a1`, and each variable's values in the `ABfault` test are shown.

```
>> compConfig = compareConfigs(Results('ABfault_config'), Results('LCarea1_config'), 'config');
>> compConfig.config

ans =

  struct with fields:

      Lconn_a1: 100000000
    fault_type: 'AB'
       tab_off: 5.1000
        tab_on: 5
    tdis_Lc_a1: 20
```

Figure 11: Comparing system configurations

Now, we can plot the generator speeds that we saved. If you would like to save more or different variables on subsequent trials, you can change the `varsToSave` variable on line 163. Figure 12

shows the result of plotting these generator speeds using the provided code in the "Simulation Results" folder.

```
>> abwall = Results('ABfault_w_all');
lcwall = Results('LCarea1_w_all');
figure
subplot(2,1,1)
plot(abwall)
ylabel('Generator Speed (p.u.)')
xlabel('Time (s)')
title('Generator Speeds for ABfault Test')
subplot(2,1,2)
plot(lcwall)
ylabel('Generator Speed (p.u.)')
xlabel('Time (s)')
title('Generator Speeds for LCarea1 Test')
```

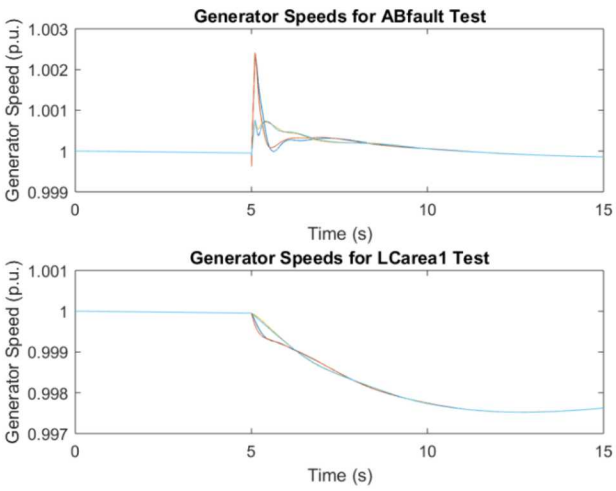

Figure 12: Plotting example results

As expected, this shows that a line-line fault occurred at 5 seconds in the ABfault test and a load connection event occurred in the LCarea1 test. This example, and the others can be modified to explore a host of dependencies and performance attributes of the system. Once thoroughly understood, the testbed is a valuable and flexible tool for investigating the performance of an electrical system with a variable amount of CIG penetration and different types of SI control.

REFERENCES

[1] M. Klein, G. J. Rogers, and P. Kundur, "A fundamental study of inter-area oscillations in power systems," *IEEE Trans. Power Syst.*, vol. 6, no. 3, pp.914–921, Aug. 1991.

[2] Golestan, Saeed, Malek Ramezani, Josep M. Guerrero, Francisco D. Freijedo, and Mohammad Monfared. "Moving Average Filter Based Phase-Locked Loops: Performance Analysis and Design Guidelines." *IEEE Transactions on Power Delivery*, vol. 29, no. 6, pp. 2750-2763, June 2014.

[3] Karimi, Houshang, Masoud Karimi-Ghartemani, and M. Reza Iravani. "Estimation of frequency and its rate of change for applications in power systems." *IEEE Transactions on Power Delivery* 19.2 (2004): 472-480.

[4] D. Halbwachs, P. Wira, and J. Mercklé, "Adaline-based approaches for time-varying frequency estimation in power systems," *IFAC Proceedings Volumes*, vol. 42, no. 19, pp. 31–36, 2009.

[5] P. Dash, A. Pradhan, and G. Panda, "Frequency estimation of distorted power system signals using extended complex kalman filter," *IEEE Transactions on Power Delivery*, vol. 14, no. 3, pp.761–766, 1999.

[6] P. Dash, A. Pradhan, and G. Panda, "Frequency estimation of distorted power system signals using extended complex kalman filter," *IEEE Transactions on Power Delivery*, vol. 14, no. 3, pp. 761–766, 1999.

[7] P. K. Dash, A. K. Pradhan and G. Panda, "Frequency estimation of distorted power system signals using extended complex Kalman filter," in IEEE Transactions on Power Delivery, vol. 14, no. 3, pp. 761-766, July 1999.

[8] K. Nishiyama, "A nonlinear filter for estimating a sinusoidal signal and its parameters in white noise: on the case of a single sinusoid," in IEEE Transactions on Signal Processing, vol. 45, no. 4, pp. 970-981, April 1997.

[9] J. Liu, "Measurement system and technique for future active distribution grids," Ph.D. dissertation, RWTH Aachen University, 2013.

[10]J. Liu, F. Ni, J. Tang, F. Ponci, and A. Monti, "A modified taylor-kalman filter for instantaneous dynamic phasor estimation," in *Innovative Smart Grid Technologies (ISGT Europe)*, 2012 *3rd IEEE PES International Conference and Exhibition on. IEEE*, 2012, pp. 1–7.

[11] IEEE Draft Standard for Synchrophasor Data Transfer for Power Systems," in IEEE PC37.118.2/D3.2, May 2011 , vol., no., pp.1-54, 30 June 2011

[12] R. Chudamani, Krishna Vasudevan, and C. S. Ramalingam. "Real-Time Estimation of Power System Frequency Using Nonlinear Least Squares." *IEEE Transactions on Power Delivery*, vol. 24, no. 3, pp. 1021-1028, July 2009.

DISTRIBUTION

Email—Internal

Name	Org.	Sandia Email Address
Raymond H. Byrne	8813	rhbyrne@sandia.gov
Charles J. Hanley	8810	cjhanle@sandia.gov
Brian J. Pierre	8813	bjpierr@sandia.gov
David A. Schoenwald	8813	daschoe@sandia.gov
Felipe Wilches-Bernal	8813	fwilche@sandia.gov
Technical Library	01977	sanddocs@sandia.gov

Email—External (encrypt for OUO)

Name	Company Email Address	Company Name
Hill Balliet	wballiet@usc.edu	University of Southern California
Alireza Ghassemian	alireza.ghassemian@hq.doe.gov	DOE
Daniel J. Trudnowski	dtrudnowski@mtech.edu	Montana Tech
Josh Wold	jwold@mtech.edu	Montana Tech

This page left blank

Sandia National Laboratories

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.