SANDIA REPORT
SAND2021-0362
Printed January 2021

Sandia National Laboratories

Dakota-NAERM Integration

Laura P. Swiler, Sarah Newman, Andrea Staid, and Emily Barrett

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

ABSTRACT

This report presents the results of a collaborative effort under the Verification, Validation, and Uncertainty Quantification (VVUQ) thrust area of the North American Energy Resilience Model (NAERM) program. The goal of the effort described in this report was to integrate the Dakota software with the NAERM software framework to demonstrate sensitivity analysis of a co-simulation for NAERM.

ACKNOWLEDGEMENTS

We acknowledge the program managers of the North American Energy Resilience Model (NAERM) program, especially the principal investigator, John Grosh (LLNL), for having the vision to start a Verification, Validation, and Uncertainty Quantification initiative for NAERM and to provide initial funding for collaborative initiatives such as the one described in this report. We also acknowledge the guidance of Earl Lawrence (LANL) who was the VVUQ program lead for the first half of 2020. Finally, we acknowledge the Dakota and NAERM developers, especially the HELICS co-simulation team.

CONTENTS

LIST OF FIGURES AND TABLES

ACRONYMS AND DEFINITIONS

Abbreviation	Definition
SNL	Sandia National Laboratories
PNNL	Pacific Northwest National Laboratory
LLNL	Lawrence Livermore National Laboratory
LANL	Los Alamos National Laboratory
DOE	U.S. Department of Energy
NAERM	North American Energy Resilience Model
VVUQ	Verification, Validation, and Uncertainty Quantification
UQ	Uncertainty Quantification
SA	Sensitivity Analysis
Dakota	Software tool developed at SNL for UQ/SA/calibration/optimization/surrogate modeling, etc.
HELICS	Hierarchical Engine for Large-scale Infrastructure Co-Simulation. Software developed at PNNL for running two or more simulations in a tightly integrated framework.
BES	Bulk Electric System
WECC	Western Electricity Coordination Council
HPC	High Performance Computing System
MPI	Message Passing Interface
PGA	Peak Ground Acceleration
USGS	United States Geological Survey

1. INTRODUCTION

This report summarizes work done under the Verification, Validation, and Uncertainty Quantification (VVUQ) thrust area of the North American Energy Resilience Model (NAERM) Program. The specific task of interest described in this report is focused on integrating the Dakota software developed at Sandia National Laboratories with the NAERM co-simulation software framework. To perform this task, Dakota was coupled with a sample NAERM co-simulation use case involving a BES model and threat model. The focus of this task is to demonstrate the capability of a NAERM model such as a co-simulation to be run in "ensemble mode" where many instances of the co-simulation are able to be launched to perform a parameter study for sensitivity analysis. The initial goal is to use Dakota as the sample generation framework and the NAERM software to drive the co-simulation.

Dakota provides a suite of uncertainty quantification (UQ) and sensitivity analysis (SA) methods. The NAERM software framework is a series of microservices that work together to model, co-simulate and analyze different aspects of integrated energy and other interdependent infrastructure systems. Co-simulation refers to two or more simulations (e.g. BES and natural gas) which are synchronized and passing data bi-directionally. HELICS is a framework for running "co-simulations" leveraged by the NAERM software framework and allows multiple simulations to be coupled tightly. Loose coupling would mean running one simulation (e.g. natural gas) to completion and feeding the results to the next simulation (e.g. BES). Tight coupling refers to synchronizing the time stepping so that it is consistent between the simulations and running the simulations concurrently, passing information back and forth at each time step. The goal is to demonstrate Dakota-NAERM integration by running a Dakota sampling study on a co-simulation, thus generating an "ensemble" or set of NAERM co-simulation runs.

While this report focuses on a specific implementation, we want to emphasize that the goal is to provide an example of a capability that can be widely used in NAERM. Dakota has many different SA and UQ algorithms. The HELICS co-simulation core within NAERM can be used to couple various types of simulations although this example focuses on a BES and threat model to demonstrate the software coupling between Dakota and NAERM. The goal here is to show a particular example that can be extended in different ways, depending on the NAERM models of interest.

The report outline is as follows: Section 2 provides an overview of the individual software tools Dakota and NAERM. Section 3 provides an overview of the software coupling, including capabilities and scripts that were used. Section 4 demonstrates a simple parameter study with Dakota and a NAERM co-simulation run using HELICS. Section 5 provides a summary with ideas for next steps.

2. SOFTWARE

This section describes the two software frameworks that were used in this analysis.

2.1. Dakota

Dakota is a suite of iterative mathematical and statistical methods that interface to computational simulations [Adams, 2020]. Dakota's goal is to make parametric explorations of simulations practical for a computational design-analyze-test cycle. Dakota is an open-source software toolkit (https://dakota.sandia.gov) and has algorithms for optimization, uncertainty quantification (UQ), parameter studies, and model calibration.

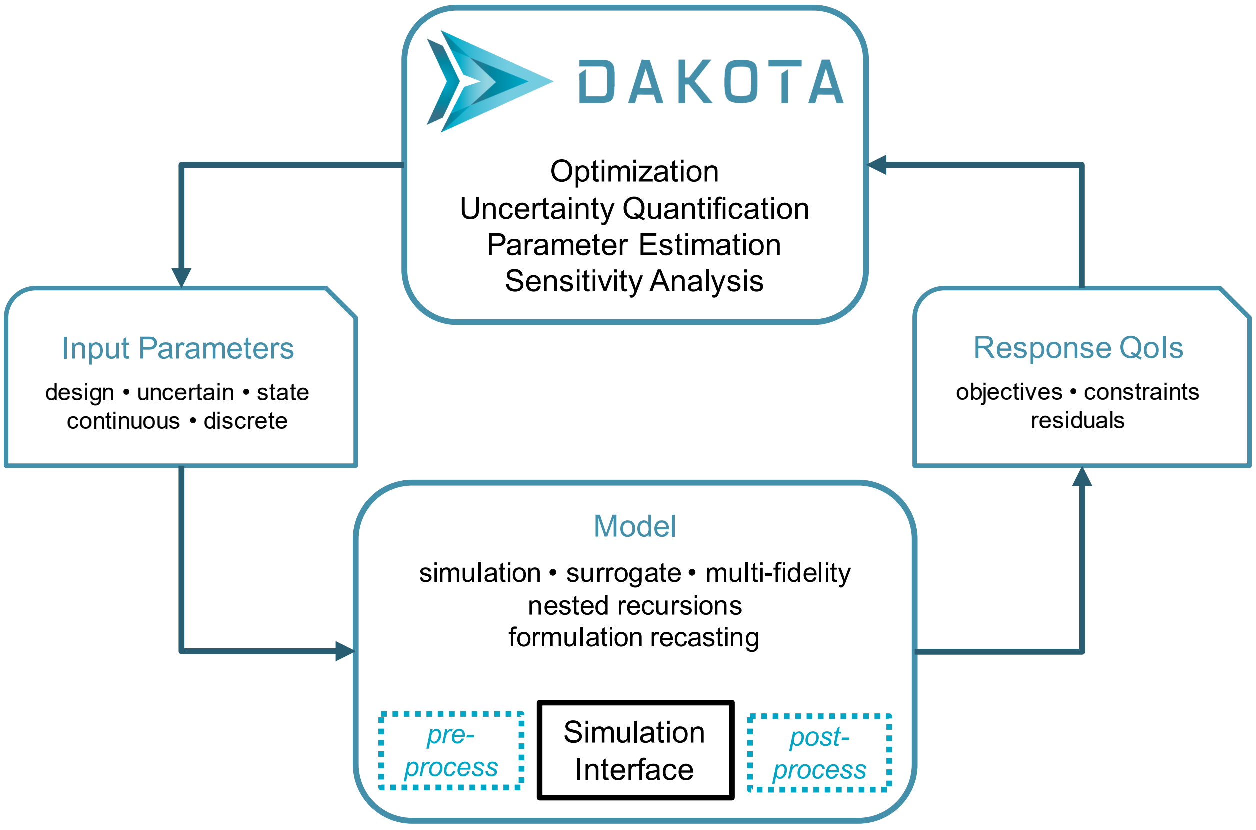
Dakota has been used for a wide variety of uncertainty quantification and sensitivity analysis studies, including of nuclear fuel performance [Gamble, 2016; Pastore, 2014]; computational fluid dynamics [Delchini, 2018], constitutive model material properties [Portone, 2019], and geologic repository performance assessment [Stein, 2019]. Of particular interest to NAERM is the extensive suite of uncertainty analysis methods in Dakota, including sampling methods (Monte Carlo, Latin Hypercube Sampling, quasi-Monte Carlo methods), design of experiments, fractional and full factorial designs, reliability methods, stochastic expansion methods such as polynomial chaos, epistemic uncertainty approaches including interval analysis and Dempster-Shafer evidence calculations, and Bayesian calibration methods including a wide variety of Markov Chain Monte Carlo methods). These are summarized in [Swiler, Adams, Eldred 2015].

Dakota is a separate executable code from the simulation being analyzed. Dakota communicates and runs a model many times, as shown in Figure 2-1. The mechanics of how this is done can vary depending on which Dakota method is chosen (e.g. in a sampling study, the samples can be determined a priori and run concurrently but in an optimization study, subsequent evaluations will depend on prior evaluations in the optimization search), the platform on which the simulation is being run and if the simulation can run on multiple processors, etc.

In a typical study, Dakota specifies the input parameters at which the model should be run. The model is run with those parameter values and the resulting quantities of interest (QoIs) are returned to Dakota as shown in Figure 2-1. Note that the "model" being driven by Dakota in the lower block in Figure 2-1 can be anything: a simulation model, a surrogate model such as a polynomial regression or a neural network model, or more advanced models that we have in Dakota. For the purposes of this report, the model is assumed to be a NAERM co-simulation model.

To provide more detail about how Dakota actually runs the model in the case of a simulation: Dakota writes a parameters file that contains one value for each variable to be varied in the simulation. In a typical use case, Dakota invokes an "interface script" which requires two command line arguments: the filesystem path/names of the parameters file and the results file. This interface script must be created by the Dakota user. It performs three tasks:

1. Pre-processing: Create the simulation input using values from the Dakota parameters file
2. Run: Run the simulation based on the input
3. Post-processing: Extract scalar quantities of interest (responses) from simulation output and write them to the Dakota results file



.

Figure 2-1. Overview of Dakota and its interaction with a simulation model

More detail about the specific interface script for HELICS is provided in Section 3.

2.2. NAERM Co-simulation Framework

The NAERM capability consists of a series of microservices which are coordinated to model, simulate and analyze the integrated energy system and other interdependent critical infrastructure at a national scale. A key component of this analysis is a co-simulation engine, which leverages HELICS. The Hierarchical Engine for Large-scale Infrastructure Co-Simulation (HELICS) is a flexible and scalable open-source co-simulation framework designed to integrate simulators designed for separate domains to simulate regional and interconnection-scale power system behaviors. Websites: https://helics.org, https://store.pnnl.gov/content/helics. In the context of the NAERM software, HELICS is used to couple various infrastructure models by wrapping domain simulators in "Federates," which communicate at each timestep over the HELICS message bus. The Federates are each embedded in separate microservices which are launched upon co-simulation start. The HELICS co-simulation is kicked-off via a series of RESTful API requests to the relevant microservices. In addition to the Federate microservices, several other microservices are required, including one for the HELICS Broker (which coordinates between the Federates) and several modeling, analytical and database management services.

In a NAERM co-simulation, any number of models can be coordinated via HELICS, although for this simple proof-of-concept, only two models were used: one for the BES and a simple threat model. Because the entire NAERM framework was used for this example, it can be easily extended in the future for different use cases involving other models. For our simple use case, the threat model was used to publish generator outages to the BES model using HELICS and then the BES model uses PowerWorld to solve the BES base case with the generator outages applied and reports any bus voltage or line overload violations.

3. COUPLING DAKOTA AND NAERM CO-SIMULATIONS

This section describes the interfacing and scripts used to couple Dakota and NAERM co-simulations.

3.1. Required capabilities for generating ensemble runs

We first describe the capabilities necessary for generating an ensemble of runs with the NAERM software. Note that this is a fairly generic list, not linked to Dakota.

1. Capability to take a list of desired samples from an external code (e.g. Dakota, PSUADE, GPMSA, Matlab, R, etc.) and run a NAERM co-simulation model at each of the input sample points.

   a. Note: some SA/UQ approaches can generate the sample list a priori (e.g. with N samples) and then one can launch the N samples independently. This is a "non-adaptive" mode. Other algorithms will require generating the samples "in-the-loop", so that one sample is generated at a time, and based on the results of that sample(s), the algorithm decides what the next samples are. This is an "adaptive" mode. Examples are optimization, importance sampling, or surrogate model refinement methods where samples are placed adaptively in regions of interest. For this report, we use non-adaptive mode.

2. Develop a clean interface to pass parameters into the NAERM co-simulation and the relevant simulation models being coordinated with HELICS and extract the results.

   a. Note: this assumes that the "driving" application such as Dakota can generate samples over parameter sets for all models defined in the co-simulation (e.g. a natural gas and electric grid model).

   b. In terms of the responses, we need the capability to obtain the results from the models involved in the co-simulation. There may be the need to perform interpolation or aggregation so that the results are on the same timescales. Again, for this simple use case, results from only one of the models was used (the BES model).

3. Framework: This relies on the Dakota workflow, but the steps are fairly generic. For each sample of model parameters:

   a. Create a working directory tagged with the sample number. Copy or soft-link all relevant files (e.g. mesh files, executables, boundary conditions, etc.) to that directory.

   b. Substitute the parameter values in the sample into the input decks for the relevant code.

   c. After the input deck is created, it is necessary to launch a NAERM co-simulation in the working directory. It would be helpful to have some error codes so that if a particular code fails with a given set of sample inputs, it can be identified and the codes don't just hang.

   d. After the co-simulation is run, it is necessary to postprocess the results and return the relevant quantities of interest to both simulations.

3.2. Dakota generation of ensemble runs

This section describes the generation of an ensemble of simulation runs using Dakota. This is a particular instantiation of the requirements discussed in Section 3.1 above.

Figure 3-1 shows the outline of the Dakota-NAERM workflow for one sample. In this figure, we see that Dakota itself requires a text input file. This is a file which describes the variables and responses for the study, the Dakota method (e.g. sampling, parameter study, optimization method, etc.), and other configuration settings for Dakota.

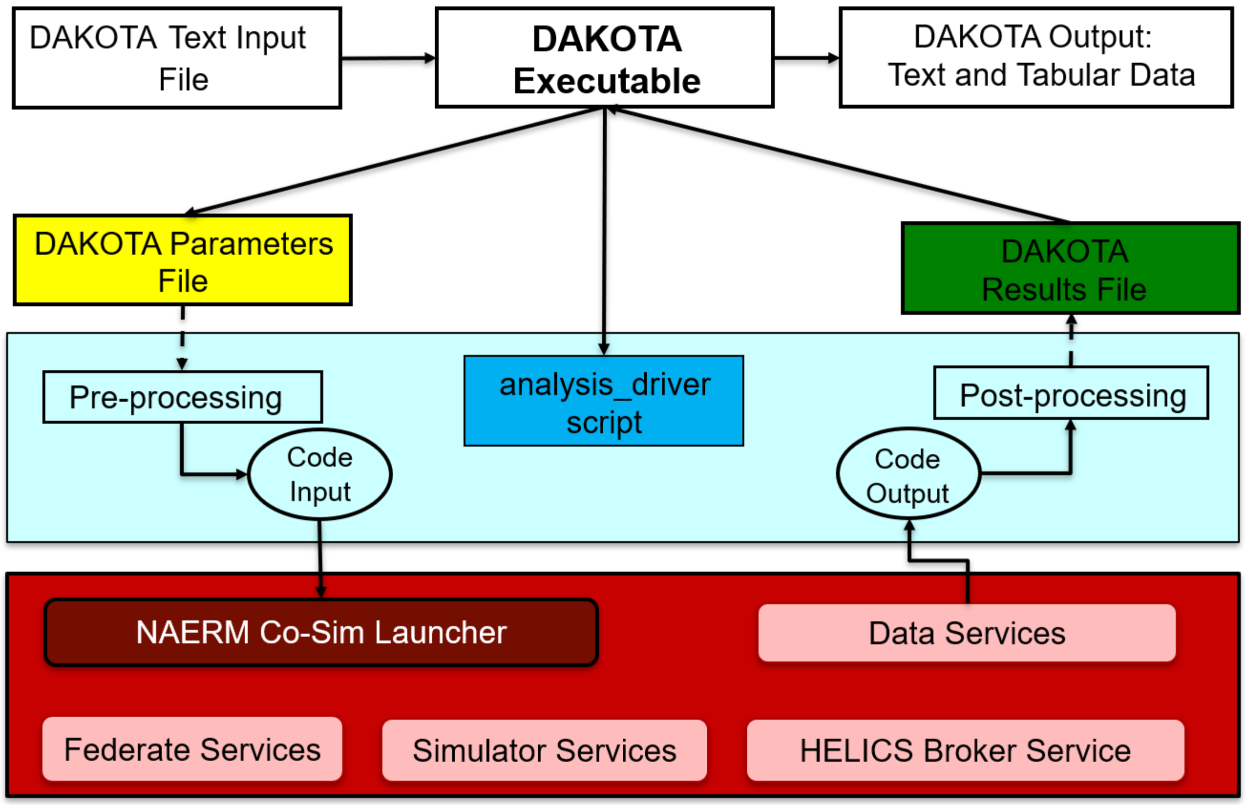


Figure 3-1. Dakota-NAERM workflow

Once Dakota is run, it generates a parameters file for each simulation run as shown in yellow in Figure 3-1. The parameters file contains the actual sample values for that particular run. This parameter file is then called by an analysis_driver script. This is a script that the Dakota user must create: everything in the light blue box is orchestrated by the analysis_driver script.

The analysis_driver script first must do some preprocessing to substitute the Dakota sampled parameter values into the input decks for the NAERM co-simulation. This is typically done with a text processing script provided by Dakota called dprepro (Dakota PreProcessing). The dprepro script takes a labeled parameter, e.g. my_param1, and substitutes into a template file if it is marked up with some delimiter. Another way to extract Dakota-selected parameter values is to use a Python utility called "di" for Dakota input.

An example of the Dakota input file (called dakota_naerm_earthquake.in for this particular study) is given in Figure 3-2. This would be invoked by running the command where "dakota" refers to the Dakota executable, -i specifies the Dakota input file name, and -o specifies the Dakota output file name.

```
dakota -i dakota_naerm_earthquake.in -o dakota_naerm.out
```

```
environment
method_pointer = 'PSTUDY'
tabular_data
tabular_data_file = 'naerm_cosim_earthquake.dat'

method,
id_method = 'PSTUDY'
multidim_parameter_study
partitions = 3 9


variables,
discrete_design_range = 2
lower_bounds = 1 1
upper_bounds = 4 10
descriptor     'fragility_curve_option' 'replicate_num'


interface,
fork
analysis_drivers = 'python.exe dakota_naerm_cosim.py'
parameters_file = 'params.in'
results_file = 'results.out'


responses,
num_objective_functions = 1
no_gradients
no_hessians
```

Figure 3-2. Dakota input file: dakota_naerm_earthquake.in

The study shown in Figure 3-2 involves sampling two variables, an index to a fragility curve (there are four fragility curves) and a replication number (1-10). The sampling is done as a multidimensional parameter study, specified in the Dakota method block as "multidim_parameter_study." This method involves specifying the number of partitions per variable (e.g. 3 partitions means four actual values are sampled between the lower and upper bound of the variables).

Each sample that Dakota generates will be processed by the "analysis_driver" in the interface block. These are the steps outlined in the blue dotted box in Figure 3-1. For NAERM, a Python script called dakota_naerm_cosim.py will be executed. This Python script is shown in Figure 3-3.

Figure 3-3 lists the steps involved in the NAERM setup (steps 1-5) and results processing (steps 6-8):

1. Launch the Kubernetes pods for each co-simulation iteration
2. Parse the Dakota input file with the Python di tool (see the di.read line and subsequent extraction of the Dakota parameter values)
3. Use the fragility curve value to select the appropriate fragility curve and sample it to determine which generators will be turned off
4. Update the Federate launch RESTful API requests to reflect the iteration name (for collecting the results later) and the outaged generators to be published by the threat model
5. Send the Federate and Broker launch requests to kick-off the co-simulation
6. Read in the results file listing voltage and current violations for the final co-simulation timestep
7. Clean up files from the co-simulation iteration
8. Return the number of violations to Dakota (as an example output of interest)

For this particular use case, an additional shell script was used to run set-up steps that only need to occur once and not for each co-simulation iteration. This includes clearing any output from previous Dakota-NAERM runs, starting up a dummy data service, starting up the PowerWorld service, and then launching Dakota.

A dummy data service was created for this Dakota-NAERM proof-of-concept, to avoid overloading the real NAERM database. It consists of a simple Python module which spins up an aiohttp service, redirects the output from the BES and threat model microservices to this service, and writes output to a format which can be used by the analysis_driver script described above.

```
#!/usr/bin/env python
# Dakota will execute this script as
#  dakota_naerm_cosim.py params.in results.out
#  The command line arguments will be extracted by dakota.interfacing automatically.

import os
import subprocess
import sys
import socket
import time
import json
import numpy as np
from http import HTTPStatus
import dakota.interfacing as di
from dakota_utils import query_case, send_post_requests
import multiprocessing
import pandas as pd
import random


ROOT_DIR = os.path.dirname(os.path.realpath(__file__))
PW_DIR = os.path.join(ROOT_DIR, '..', 'powerworld_service', 'src')


if __name__ == "__main__":
    # Required to allow query_case call to work in a process
    multiprocessing.freeze_support()

    # -------------------------
    # Start k8s pods
    # -------------------------
    subprocess.call(['kubectl', 'apply', '-f', 'data/input/naermcosim_cr.yaml'])
    time.sleep(60)

    # ---------------------------
    # Parse Dakota parameters file
    # ---------------------------

    params, results = di.read_parameters_file()
    frac_curve_num = params['fragility_curve_option']
    replicate = params['replicate_num']
    print(f'\nRunning NAERM cosim with fragility curve: {frac_curve_num}, replicate {replicate}')

    # ------------------------------
    # Convert and send to application
    # ------------------------------
    # Sample generators to be outaged based on probabilities
    frag_curves = pd.read_csv(os.path.join(ROOT_DIR, 'data', 'input',
                              'gens_with_failure_probs_wecc_san_andreas.csv')).fillna(0)
    col_name = f'FOR_{frac_curve_num}'
    frag_curves['gen_out'] = frag_curves.apply(lambda x: random.choices([1, 0], [x[col_name], 1-x[col_name]])[0],
                                axis=1)
    gen_out_list = frag_curves.loc[frag_curves['gen_out'] == 1, 'Name'].to_list()

    # Write list of outaged generators to file
    with open(os.path.join('data', 'output', 'outaged_gens.dat'), 'a') as f:
        f.write(f'curve: {col_name}, replicate: {replicate}, gens: {gen_out_list}\n')

    # Read in launch json files and replace cosim_uuid, powerflow filename,  and outages
    # Read in base launch json files
    with open(os.path.join(ROOT_DIR, 'data', 'input', 'bes_base.json')) as f:
        bes_json = json.load(f)
    with open(os.path.join(ROOT_DIR, 'data', 'input', 'threat_base.json')) as f:
        threat_json = json.load(f)
    with open(os.path.join(ROOT_DIR, 'data', 'input', 'broker_base.json')) as f:
        broker_json = json.load(f)
```

```
# Replace cosim uuids
cosim_uuid = 'dakota_test'
bes_json['cosim_uuid'] = cosim_uuid
threat_json['cosim_uuid'] = cosim_uuid
broker_json['cosim_uuid'] = cosim_uuid
bes_json['name'] = f'bes_{frac_curve_num}_{replicate}_outages'
threat_json['name'] = f'threat_{frac_curve_num}_{replicate}_outages'

# Replace powerflow file
bes_json['powerflow_options']['powerflow_file'] = os.path.join(
    ROOT_DIR, 'data', 'input', 'power_flow_file.pwb')

# Replace outages
threat_json['outage_data']['wecc_bes'] += [
    {"id": gen_id, "asset_type": "generator", "time_step": 0} for gen_id in gen_out_list]

# Launch cosim via API requests
service_host = 'localhost'
response, service = send_post_requests(service_host, broker_json, bes_json,
                                       threat_json)
if response.status != 200:
    raise Exception(f'Could not send launch requests to federates, '
                    f'error for {service} > 'f'{response.data}')

# ----------------------------
# Return the results to Dakota
# ----------------------------
# Load output powerflow cases to get number of violations
time.sleep(2)
num_violations = None
data_dir = os.path.join(ROOT_DIR, 'data', 'output')
while num_violations is None:
    violations_filename = os.path.join(data_dir, f'violations_{bes_json["simulation_duration"]}.0.json')
    if os.path.exists(violations_filename):
        # Read in violations file
        time.sleep(1)
        with open(violations_filename, 'r') as f:
            output_json = json.load(f)
        # Check if case converged
        if output_json["converged"]:
            # Count violations from each asset type
            num_violations = 0
            for assets in output_json['assets']:
                violations = pd.Series(assets['ViolationType'])
                num_violations += violations[violations != ''].count()
        else:
            num_violations = -1

    time.sleep(5)

# Delete violations files
violations_files = [file for file in os.listdir(data_dir) if 'violations' in file]
for file in violations_files:
    os.remove(os.path.join(data_dir, file))

# Results.responses() yields the Response objects.
for i, r in enumerate(results.responses()):
    if r.asv.function:
        r.function = num_violations

results.write()
subprocess.call(['kubectl', 'delete', 'cosim', 'abc123'])
```

Figure 3-3. Dakota_naerm_cosim.py file: interface script between Dakota and the NAERM software framework execution

Once the co-simulations are run and the quantities of interest are extracted through whatever post-processing is needed, they must be passed back to Dakota in a file called the results file (the green box in Figure 3-1). This is a flat ASCII text file with a simple format: one response per line. Once Dakota obtains the results file for a particular sample, it will generate the next sample. If the runs can be performed concurrently, Dakota has a wide variety of parallel options so that multiple batches of runs can be launched by Dakota concurrently. This will depend on the platform on which the simulation is run, the job submission system if it is a high-performance computing platform (HPC), and if the simulation itself requires multiple processors and uses MPI. For the purposes of this report, we assumed that the NAERM study requires multiple processors and MPI, but Dakota launches the samples serially.

4. RESULTS

In this section, we present the results of a simple sensitivity analysis study to demonstrate the Dakota-NAERM integration. The sensitivity analysis involved an earthquake threat to the WECC power grid. We selected a sample earthquake from the USGS scenario catalog (https://earthquake.usgs.gov/scenarios/catalog/). Each event has simulated ground motion data for a hypothetical earthquake event. Here, we are using an event that occurs along the San Andreas fault line, namely, the San Andreas (Mojave S) event, with a magnitude of 7.38 and an epicenter located at 34.51492 degrees latitude, -118.02401 degrees longitude. The earthquake threat model takes as input a number of fragility curves for different component types. Each curve maps a level of ground motion intensity (here, in the form of peak ground acceleration, or PGA) to the probability of failure for a specific component type. For example, we have different fragility curves depending on whether a generator is large or small, anchored or unanchored for seismic stability. We also have fragility curves to represent transmission tower failures, substation failures, and gas pipeline failures, but for our demonstration here, we are only dealing with generator failures in the WECC. We performed a parameter study where we chose four different fragility curves and assigned that same curve to every generator in the system and varied the choice of fragility curve (curves 1-4 representing different failure probabilities based on PGA) to then use to assign failure probabilities to each individual generator. From these probabilities, we can draw random samples to represent a single set of generator failures. We generate ten replicates per fragility curve because the sampling of outages based on the fragility curve is stochastic, so we get different component outages for each sample of a particular fragility curve. This combination of fragility curves times replicates makes 40 total samples. The sample design is shown below:

Fragility Curve	Replicates
1	10
2	10
3	10
4	10

Table 4-1. Sampling study over replicates of fragility curves

This experimental design is simple, and the choice of fragility curves used here is not realistic but was instead chosen to show variability among levels of sensitivity. In a realistic application, each component would be assigned the proper fragility curve based on the asset type and construction. Here, the purpose of this study was to illustrate the Dakota-NAERM coupling and provide an example for the types of larger sampling studies that can be performed. The results presented here are meant to be illustrative of sensitivity and uncertainty studies. We caution against overinterpreting these results: a larger, more inclusive study would need to be undertaken for a comprehensive analysis of earthquake threat to the power grid. Again, this is a simple study meant to demonstrate the sampling capabilities.

Figure 4-1 provides a map of a San Andreas earthquake peak ground acceleration values, together with the location of WECC generators (black dots). There are several assets that are exposed to the highest levels of PGA, with many others in a lower risk zone as indicated by low PGA values in yellow.

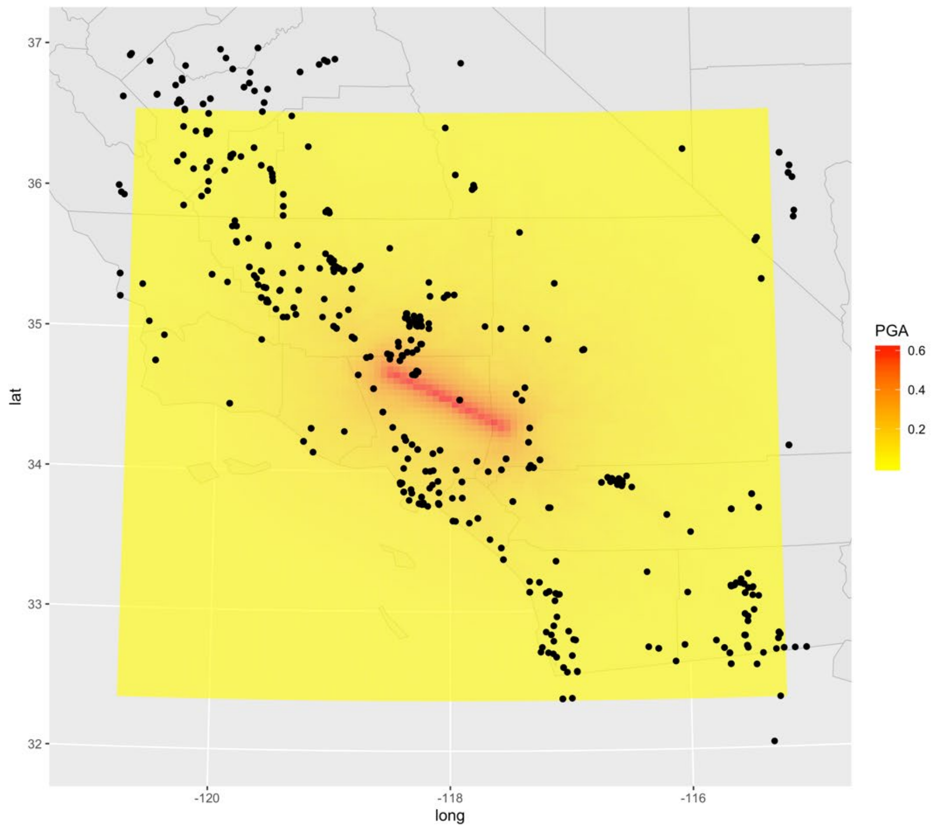


Figure 4-1. Map of a San Andreas earthquake.

The NAERM co-simulation use-case was run on all forty simulations of the WECC grid based on the sampling study defined in Table 4-1, using PowerWorld to solve the WECC powerflow case at each timestep of each simulation. The number of violations resulting from the earthquake-induced generator outages and the number of generators which experienced outages in each of the forty runs were output. Figure 4-2 shows a discrete histogram and smoothed density plot of the number of generators out for each fragility curve. Due to the small number of replicates (10) for each fragility curve, the histograms are a bit sparse, so we recommend not overinterpreting the density of the number of generators out. However, this type of analysis shows what can be performed given realistic earthquake scenarios and a larger number of replicates, which will better represent the range of expected outcomes.

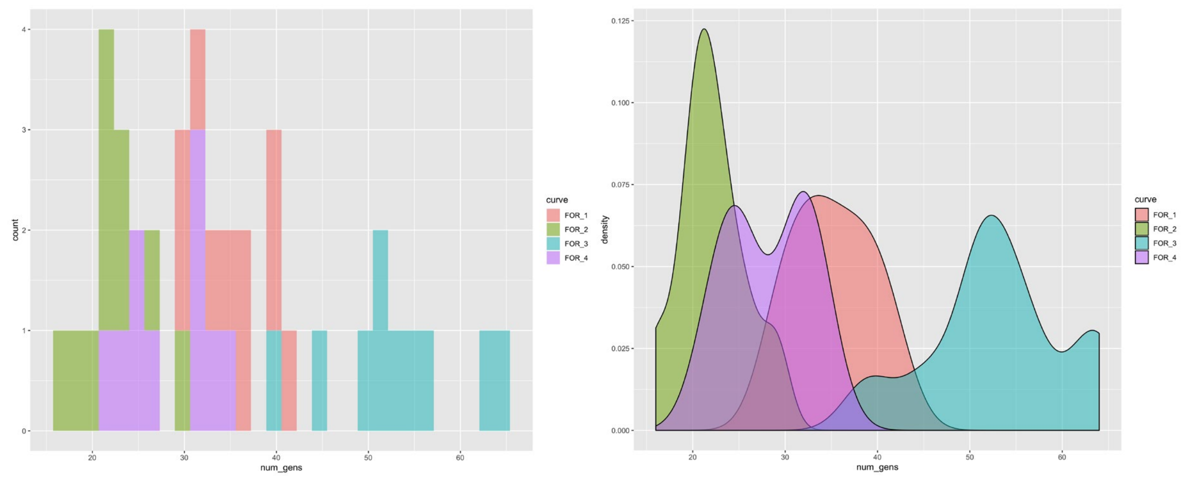


Figure 4-2. Histogram of number of generators out (left plot) per each fragility curve and corresponding smoothed probability density function (right plot) for the sampling study performed.

Figure 4-3 shows a scatterplot of the number of generators out vs. number of violations, colored by fragility curve. Note that the overall correlation between number of generators out and number of violations was not that high: 0.48. Figure 4-3 shows that Fragility curve 3 had the most outages and the most violations: it was the most extreme scenario.

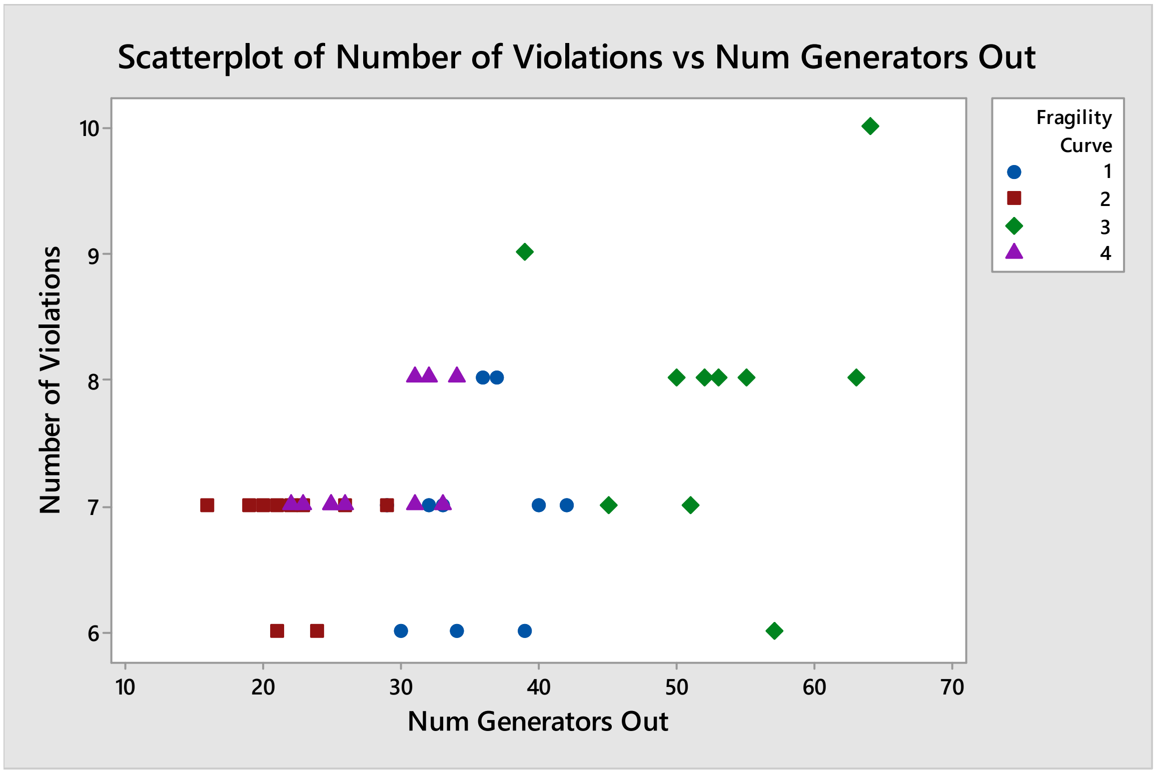


Figure 4-3. Number of generators out vs. Number of violations

Figure 4-4 shows a main effects analysis. In a main effects analysis, one is interested in looking at differences in mean values of the output (in this case, the number of generators out or number of violations) per treatment level (fragility curve).

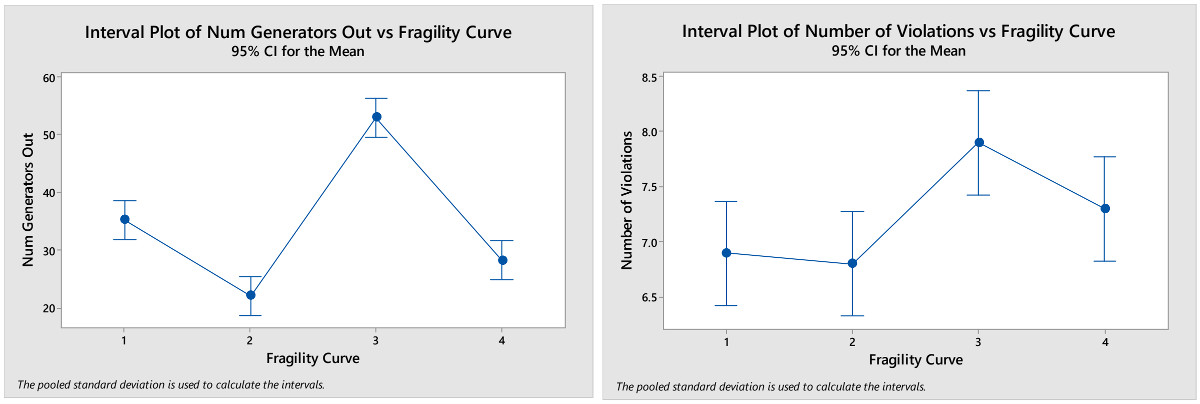


Figure 4-4. Main effects analysis for generators out (left panel) and number of violations (right panel).

Figure 4-4 shows that fragility curve 3 results in the largest number of the average generators out (52.9) and also the largest number of mean violations (7.9). This is consistent with Figure 4-3. Further, one can use a main effects analysis to identify if the differences in the mean number of generators out or mean number of violations are statistically significant across fragility curves. Table 4-2 shows that fragility curves 1 and 3 result in a different mean number of generator failures but fragility curves 2 and 4 have a mean number of generators out that cannot be distinguished from each other. This is shown by the letters grouping together the means that cannot be distinguished according to the Tukey test. That is, although fragility curve 2 has an average of 22.1 generators out and fragility curve 4 has an average of 28.2 generators out, the variance in these estimates is not small enough to identify them from separate populations. For number of violations, the analysis shows a different result: Fragility curve 3 and 4 cannot be considered to have a different number of mean violations, but we also see that fragility curves 1, 2, and 4 cannot be clearly distinguished in terms of their mean number of violations. This is due to a relatively large variance in the mean estimates of number of violations as shown in Figure 4-3 (the variance of the mean contributes to the 95% confidence intervals shown).

Number of generator outages: Tukey test

Grouping Information Using the Tukey Method and 95% Confidence

Fragility Curve	N	Mean	Grouping		
3	10	52.90	A		
1	10	35.20		B	
4	10	28.20			C
2	10	22.10			C

Means that do not share a letter are significantly different.

Number of violations: Tukey test
Grouping Information Using the Tukey Method and 95% Confidence

Fragility Curve	N	Mean	Grouping	
3	10	7.900	A	
4	10	7.300	A	B
1	10	6.900		B
2	10	6.800		B

Means that do not share a letter are significantly different.

Table 4-2. Main effects analysis for number of generators out and number of violations

We now examine the number of violations and generators out for each of the fragility curves in more detail. Figure 4-5 shows a scatterplot of the violations and generators out for each replicate.

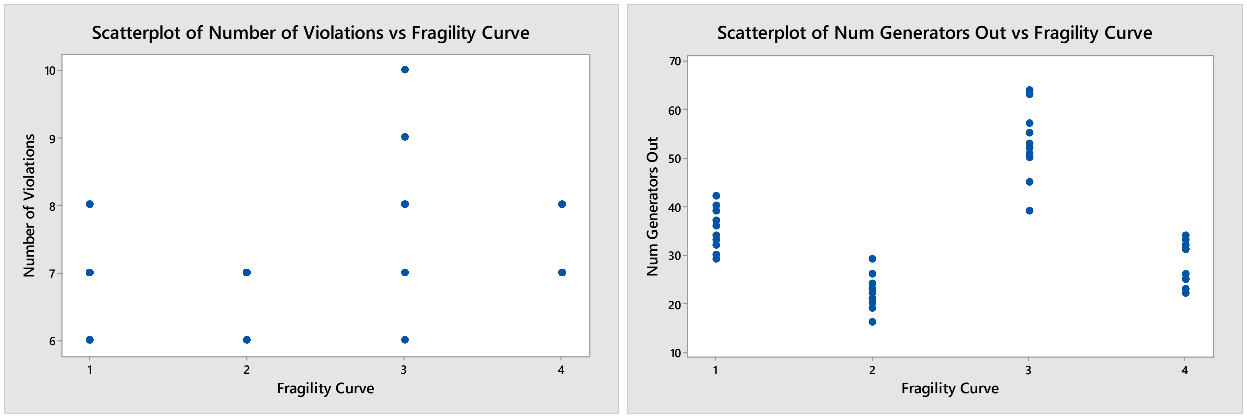


Figure 4-5. Number of violations (left panel) and generators out (right panel) for each of the fragility curves.

Figure 4-5 shows an important finding: although each stochastic replicate of a fragility curve resulted in a unique number of generators out (right panel of Figure 4-5), these generator outages mapped to only a few instances of violations. For example, the generator outages for fragility curves 2 and 4 both mapped to only two unique number of violations (e.g. fragility curve 2 led to 6 or 7 violations). Further analysis shows that of the ten replicates for fragility curve 2, eight of the replicates had 7 violations and two had 6 violations, resulting in a mean outage of 6.8 for fragility curve 2 as shown in the main effects table in Table 4-2. Similarly, seven of the ten replicates of fragility curve 4 had 7 violations and only three had 8 violations. Fragility curve 3 had the widest range of number of violations.

Finally, we examined the actual generators that failed. As expected, the generators that had higher failure probabilities according to the fragility curves had a larger number of failures in the ten replicates, as shown in Table 4-3.

Generator		Fragility 1	Fragility 2	Fragility 3	Fragility 4
Gen. 29454	Probability of failure	0.251	0.164	0.378	0.158
	Number failed (out of 10)	3	2	4	1
Gen. 29490	Probability of failure	0.021	0.011	0.032	0.027
	Number failed (out of 10)	0	0	0	1

Table 4-3. Example generators with respective failure probabilities for the four fragility curves and corresponding number of failures in ten replicates.

As mentioned previously, the four distinct fragility curves used here were chosen to demonstrate the type of analysis possible when coupling Dakota with the NAERM co-simulation platform. In a realistic earthquake scenario, each component in the system would be assigned the fragility curve most appropriate for it, instead of using the same curve for all assets across the board. Thus, the main takeaway should focus on the type of questions that can be answered with this sensitivity analysis and sampling capability. The results shown here surrounding specific generators and violations will not be representative of an actual earthquake event.

5. SUMMARY

This report summarizes work done under the VVUQ Thrust of the NAERM program to integrate the Dakota software with the NAERM co-simulation software framework. As documented in this report, Dakota was coupled with a sample NAERM co-simulation use case involving a BES model and threat model to perform sensitivity analysis. The demonstration of sensitivity analysis involved an earthquake threat to the WECC power grid, where replicates from different fragility curves were sampled to generate failures of different components based on the failure probabilities defined by the fragility curve and the ground motion intensity.

The sensitivity analysis showed the range of outcomes (numbers of generators out and number of violations) based on the sampling of the fragility curves. Statistical analyses such as main effects analysis were performed, demonstrating that there are significant differences in the number of generator outages and the number of violations for different fragility curves. There also was a moderate correlation between the number of generator outages and the number of violations. We emphasize that this is simply an illustrative study: the four distinct fragility curves used here were chosen to demonstrate sensitivity analyses made possible when coupling Dakota with the NAERM co-simulation platform. The results shown here surrounding specific generators and violations will not be representative of an actual earthquake event. In a realistic earthquake scenario, each component in the system would be assigned the fragility curve most appropriate for it, instead of using the same curve for all assets across the board.

However, this analysis shows the potential for using these capabilities to address questions of interest to infrastructure planners and operators. For example, we can assess the range of expected grid impacts to answer questions about not only the worst-case scenario, but the most statistically likely scenarios. We can expand the scope to consider multiple earthquake events within a given region to assess the most vulnerable areas for seismic hazards when it comes to grid impacts. Additionally, we can use different fragility curves to study possible hardening investment decision-making, comparing failure rates and grid impacts if generators and other assets are designed to the highest seismic standards or not. Combined with identification of the most vulnerable areas for an earthquake event, this can be used to select the optimal set of investments to minimize adverse impacts to the grid, should an earthquake occur. By targeting investments to the most critical components, grid planners can minimize their costs while ensuring overall system resilience in the face of earthquake hazards.

This capability could be used to inform optimal planning across multiple hazards. We could, for example, simulate the expected failures due to both earthquakes and wildfires in a region, studying any joint vulnerabilities and therefore co-optimizing resilient investment strategies across hazard types. With advanced threat models and long-term hazard projections, planning decisions can accurately take these coupled vulnerabilities (and coupled uncertainties) into account.

Lastly, when considering shorter planning timescales where major infrastructure upgrades may not be viable, this type of analysis could be used to inform operational planning. By varying the implicit dispatch and operational decisions in the power flow model and/or including corrective actions in the modeling, this type of statistical analysis could inform how the grid is operated to mitigate impact in the event of a major event.

As a final note: the sensitivity analysis here focused on an earthquake scenario. Other threats and different questions can be addressed with the Dakota-NAERM coupling: Dakota has several uncertainty analysis and optimization capabilities that can be used, depending on the question of interest. We anticipate that the Dakota-NAERM framework will enable future studies involving the generation of ensembles of models to address questions about probabilities of various outcomes, worst case scenarios, and effects of mitigations in a variety of settings.

REFERENCES

[1] Adams, B.M., Bohnhoff, W.J., Dalbey, K.R., Ebeida, M.S., Eddy, J.P., Eldred, M.S., Hooper, R.W., Hough, P.D., Hu, K.T., Jakeman, J.D., Khalil, M., Maupin, K.A., Monschke, J.A., Ridgway, E.M., Rushdi, A.A., Seidl, D.T., Stephens, J.A., Swiler, L.P., and Winokur, J.G., "Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.12 User's Manual," Sandia Technical Report SAND2020-5001, May 2020.

[2] Delchini, M.G., L.P. Swiler, E.P. Popov, and W. D. Pointer. "An Uncertainty Quantification of the Computational Fluid Dynamics Solution to the Modeling of the Contact Point in a Wire-Wrapped Fuel Assembly." American Nuclear Society ANS Annual Meeting, Philadelphia June, 2018. ORNL/TM-2018/959.

[3] Gamble, K. and L. Swiler. "Uncertainty Quantification and Sensitivity Analysis Applications to Fuel Performance Modeling." TOPFUEL American Nuclear Society Conference, Sept. 2016.

[4] Pastore, G., L.P. Swiler, J.D. Hales, S.R. Novascone, D.M. Perez, B.W. Spencer, L. Luzzi, P. Van Uffelen, and R.L. Williamson, "Uncertainty and sensitivity analysis of fission gas behavior in engineering-scale fuel modeling." Journal of Nuclear Materials 456 (2015) 398–408.

[5] Portone, T., J. Niederhaus, J. Sanchez, L. Swiler. Bayesian model selection for metal yield models in high-velocity impact. *International Journal of Impact Engineering* Vol.137, March 2020. 103459

[6] Stein, E.R., L. P. Swiler, and S. D. Sevougian. "Methods of Sensitivity Analysis in Geologic Disposal Safety Assessment (GDSA) Framework." International High Level Radioactive Waste Management conference, April 2019. SAND2019-4297C.

[7] Swiler, L.P, B.M. Adams, and M.S. Eldred. "Dakota: Bridging Advanced Scalable UQ Algorithms with Production Deployment." In Springer Handbook on Uncertainty Quantification, Ghanem R., Higdon D., Owhadi H. (eds). 2015. https://doi.org/10.1007/978-3-319-11259-6_52-1.

DISTRIBUTION

Email—Internal

Name	Org.	Sandia Email Address
Daniel Turner	1463	dzturne@sandia.gov
Randy D. Smith	1464	ransmit@sandia.gov
Raymond Byrne	8813	rhbyrne@sandia.gov
Technical Library	01977	sanddocs@sandia.gov

Email—External [redacted]

Name	Company Email Address	Company Name
John Grosh	grosh1@llnl.gov	Lawrence Livermore National Laboratory
John Collins	collins20@llnl.gov	Lawrence Livermore National Laboratory
Russell Bent	rbent@lanl.gov	Los Alamos National Laboratory
Todd Hay	todd.hay@pnnl.gov	Pacific Northwest National Laboratory
Jason Fuller	jason.fuller@pnnl.gov	Pacific Northwest National Laboratory
Jimmy Ly	jly@beamreachgroup.com	Beam Reach Consulting Group

This page left blank