ARL-TR-8218 • Nov 2017

US Army Research Laboratory

Survey of Existing Uncertainty Quantification Capabilities for Army-Relevant Problems

by James J Ramsey

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TR-8218 ● Nov 2017

US Army Research Laboratory

Survey of Existing Uncertainty Quantification Capabilities for Army-Relevant Problems

by James J Ramsey
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
November 2017	Technical Report	October 2016–September 2017

4. TITLE AND SUBTITLE
Survey of Existing Uncertainty Quantification Capabilities for Army-Relevant Problems

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
James J Ramsey

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
US Army Research Laboratory
ATTN: RDRL-CIH-C
Aberdeen Proving Ground, MD 21005–5066

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-8218

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This report surveys the current state of the art in uncertainty quantification. It provides a brief overview of uncertainty quantification methods, a survey of currently available software implementations of these methods, and a discussion of how these implementations may be integrated into a workflow on a high-performance computing cluster. Finally, it provides an example of how existing uncertainty quantification capabilities may be applied to an example problem of Army interest, namely simulation of the dynamic penetration of armor.

15. SUBJECT TERMS
uncertainty quantification, readiness, high-performance computing

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			James J Ramsey
Unclassified	Unclassified	Unclassified	UU	70	19b. TELEPHONE NUMBER (Include area code) 410-278-5614

Contents

List of Figures

List of Tables

Acknowledgments

We would like to thank Daniel Hornbaker and Robert Doney for their contributions to our uncertainty quantification of a model problem in dynamic penetration of armor, especially their guidance on the use of CTH.

This work was supported in part by a grant of computer time from the Department of Defense High Performance Computing Modernization Program at the US Army Research Laboratory Department of Defense Supercomputing Resource Center.

1. Introduction

Uncertainty quantification (UQ) is needed to systematically account for the uncertainties in computational models and to express how these predictions may deviate from the ground truth. It is thus important for the US Army Research Laboratory (ARL) in its own work in computational prediction. However, the *2015–2016 Assessment of the Army Research Laboratory* has noted,[1] "A challenge that cuts across the predictive simulation sciences portfolio [at ARL] is the lack of cutting-edge R&D [research and development] efforts in validation, verification, and uncertainty quantification." Because of this assessment, efforts have begun to address this lack of routine UQ practice in modeling and simulation at ARL. A part of these efforts has been to survey the current state of the art in UQ, especially with regard to methods and software, and to examine the extent to which this state of the art is capable of addressing problems of interest to the Army. This report contains several results from this survey.

The structure of this report is as follows. Sections 2 and 3 are essentially introductory and are primarily for providing the background information needed in later portions of the report. The first of these introductory sections is an overview of UQ and its various methods. The second of these discusses issues pertaining to the use of UQ software on the Army's high-performance computing (HPC) systems, especially the issues that arise when interfacing UQ software with "black-box" applications that run on parallel HPC clusters. Section 4 is a survey of existing UQ software. It includes both applications and libraries, and it outlines not only the capabilities of these works of software, but how each of them interfaces with external applications on HPC systems. Finally, Section 5 discusses an application of UQ to a very typical example of an Army-relevant problem, namely the modeling of the dynamic penetration of armor.

2. Overview of Uncertainty Quantification

To introduce some of the concepts that will be mentioned and discussed in later sections of this report, a brief overview of UQ is provided here. First, we discuss the classification of uncertainties and its practical importance for UQ. After this, we describe the forward and inverse problems of UQ. Forward UQ is the problem of how uncertainties in the input of a computational model propagate to its output. Inverse UQ, on the other hand, is the problem of how to determine both the input

parameters of a computational model and their uncertainties, given some experimental values that this model is meant to at least approximately reproduce. This overview then concludes with a discussion of topics that are frequently involved with UQ, even if they are not in principle a necessary part of it.

2.1 Classification of Uncertainties

Uncertainties are typically classified into aleatoric and epistemic. *Aleatoric* uncertainties cannot realistically be reduced by performing additional experiments. Rather, they are attributed to inherent randomness in the physical world. On the other hand, *epistemic* uncertainties are due to lack of knowledge.[2] As an example of both kinds of uncertainty, one may suppose the existence of a material whose microstructure varies from sample to sample. If there are a large number of samples, then the mean values of the elastic parameters associated with the material can be readily assessed, as well as the variance or other statistical measures of the distribution of parameters. The uncertainty in the parameters is aleatoric. However, if only a few samples are available, then the mean value determined from these samples may not be representative of the properties of future samples, and the variance in the material's properties becomes difficult or impossible to characterize. This uncertainty is epistemic.

In principle, the distinction between aleatoric and epistemic uncertainty is imprecise. For example, seemingly random behavior could have a deterministic explanation that is merely currently unknown. One could also argue that if one were to know the exact microstructure of a given sample of material, then the parameters that characterize it could be determined precisely, regardless of the variations across samples.

In practice, though, the difference between the 2 kinds of uncertainties is that the former is typically treated within a classical probability framework. For example, the aleatoric uncertainty of a continuously variable parameter would be characterized by a probability density function (e.g., a Gaussian bell curve).[2] By contrast, epistemic uncertainty is often treated in an alternative framework where, for example, the most one may be able to say about a particular parameter is that it likely is somewhere within an interval, and it may be unclear how likely it is to be within a particular subset of that interval. Accordingly, different UQ methods are often used

for the different kinds of uncertainties.[3,4*]

2.2 Uncertainty Propagation

Uncertainty propagation is the estimation of how uncertainties in the inputs of a computational model manifest themselves as uncertainties in the quantities of interest (QoI) output by the model. The choice of propagation methods depends on whether the uncertainty in model inputs is aleatoric, epistemic, or both. Three common classes of uncertainty propagation methods for aleatoric uncertainties are Monte Carlo sampling methods, stochastic expansion, and reliability methods. For epistemic uncertainties, interval analysis and Dempster-Shafer evidence theory may be employed.

Monte Carlo sampling methods are perhaps the most straightforward. Probability distributions are specified for model inputs, and the inputs then are randomly sampled accordingly. For each sample of random inputs, the model produces corresponding values for the QoI. Given enough samples, one may obtain a probability distribution of each QoI. The method is simple and robust,[6] and it does not suffer from the so-called "curse of dimensionality", that is, its rate of convergence is independent of the number of parameters in a model.[7] However, it has 2 disadvantages. First, the convergence rate to the true probability distribution is $O(1/\sqrt{N_{\text{sam}}})$, where N_{sam} is the number of samples.[8] Second, the number of samples may be so high as to be computationally infeasible, especially if the computational model under consideration is expensive[2] or if the tails of the probability distribution of the QoI need to be accurately characterized.[6] The computational feasibility may be improved to some degree by using sampling techniques that are not purely random. For example, while purely random sampling can lead to clumping of samples within the space of possible inputs, Latin hypercube sampling (LHS)[9] leads to a more even distribution of samples. On the other hand, importance sampling[10] deliberately samples some regions of the space of possible inputs more densely, such as the ranges of inputs that would cause a QoI to exceed some threshold indicating failure,[4] and then compensates for the sampling bias when calculating moments of the probability distribution(s) of the QoI.

In stochastic expansion methods, the model output is expressed as a linear combi-

*However, O'Hagan and Oakley have made arguments made for treating epistemic uncertainty within a classical probability framework.[5]

nation of basis polynomials. These methods include polynomial chaos and stochastic collocation.[4] In polynomial chaos methods,[11] the basis consists of orthogonal polynomials that are functions of the model inputs. The forms of these polynomials depend upon the probability distributions of the model inputs. The coefficients of the expansion are integrals that may be determined numerically, with the computational model evaluated at the quadrature points. Alternatively, the coefficients may be determined via regression. Once the coefficients are found, the moments of the probability distribution of the model output are available in closed form. The probability distribution itself may also be estimated.[4] In stochastic collocation, the basis consists of (possibly piecewise) polynomials that interpolate over model inputs sampled on regular (and possibly sparse) grids. The expansion coefficients are values of the model output at those model inputs, and possibly values of partial derivatives of the model output at those model inputs as well, depending on the means of polynomial interpolation. Moments of the probability distribution of the model output are again available in closed form,[8] and again, the probability distribution of the model output may also be estimated.[4] Stochastic expansion methods can potentially require far fewer evaluations of the model than Monte Carlo sampling methods, especially if the number of model inputs is relatively small, about 5 to 10 or so.[6] Stochastic expansion methods, though, can suffer a curse of dimensionality, where the computational cost explodes as the number of model parameters m increases. For example, a full k-order grid used for quadrature or interpolation has k^m points. Use of a sparse grid can mitigate the curse of dimensionality but not remove it entirely; the number of points in a Smolyak sparse grid scales as $k^{\log m}$ when employing a k-order rule.[11]

The above methods for accounting for aleatoric uncertainties are capable of providing estimates for the whole probability distribution of the QoI. However, often what is desired is an estimate of the probability that the QoI will exceed some threshold, indicating failure of the system being modeled. To determine this without determining the full probability distribution of the QoI, *reliability methods*[4,8,12] are used.

A common, though problematic, approach for dealing with epistemic uncertainty in a model input is to simply assume that possible values for the input are within some interval and that each possible value is as likely as any other. Then methods such as those described previously, which are suited for aleatoric uncertainties, are used. This approach, though, may underestimate the likelihood that a model will

predict an extreme value for a QoI, one that may indicate high stress or even failure of a system.[3] One method of avoiding this pitfall is to use interval analysis. In this method, it is still assumed that the possible values of each model input are within some interval, but there is no assumption that one random value in that interval is just as probable as another. The output of this analysis is a set of intervals, one for each QoI, rather than a probability distribution. Interval analysis employs optimization algorithms to determine the extrema of the QoI given the ranges of possible input values,[4] and these algorithms may be subject to a curse of dimensionality. For example, the UQ software Dakota uses the DIRECT algorithm in its implementation of one of its interval analysis methods,[13] and this algorithm is limited to about 20 dimensions (i.e., model inputs).[14]

A method for accounting for epistemic uncertainty that goes beyond interval analysis is Dempster-Shafer evidence theory.[3] Here, a model input can instead be within one or more intervals, rather than just a single one as in interval analysis. Associated with each interval is a so-called "basic probability assignment" (BPA), which, despite its name, is not the same as a classical probability and has been described as a measure of subjective belief instead. Obtaining BPAs is an ongoing research problem, and several methods of determination have been discussed in recent literature.[15,16] Once BPAs have been assigned to model inputs, uncertainty measures can then be determined for the QoI calculated from the model. Typically, these take the form of lower and upper bounds, respectively, on the probability distribution of a QoI.[3,4]

2.3 Inverse Problems

Typically, a computational model has various input parameters. Some or all of these may be estimated via a calibration process, where experimental values of QoI are collected and then the parameters of the model are fitted so that it approximately reproduces the experimental results. This fitting is not necessarily a form of UQ. Fitting is often ad hoc, with uncertainty in the fitted parameters only roughly taken into account through noting discrepancies between the model predictions and the experimental values used to calibrate the model.[17] However, there are more formal methods, which may be classified as either frequentist or Bayesian, that do provide measures of the uncertainty in the parameters.

In frequentist methods of model calibration, the model parameters are not taken to

be random variables, but rather as fixed but unknown values, while the experimental values are taken to be subject to random errors. A common example of a frequentist method would be ordinary least-squares (OLS) minimization, which estimates the values of the model parameters to be those that minimize the sum of the squared differences between the experimental values and the model predictions that correspond to them. Unlike the "true" values of the model parameters, these estimates are instances of random variables, and ones with Gaussian probability distributions. However, these distributions reflect the uncertainty in the estimation process itself and may not be appropriate to use as inputs to the aleatoric uncertainty propagation methods described in Section 2.2.[2]

In Bayesian methods of model calibration, the model parameters are taken to be random variables, and the goal of these methods is to estimate the probability distributions of the parameters. This is often done via a family of Monte Carlo methods called *Markov chain* Monte Carlo (MCMC) methods. All of the algorithms in this family of methods entail generating chains of random states in which the last state depends on the previous one, but the details of these algorithms have various advantages and disadvantages. For example, the MCMC algorithms called Hamiltonian Monte Carlo (or alternatively, Hybrid Monte Carlo) use gradients of the outputs of the computational model in order to efficiently sample the regions of the space of model parameters that contribute the most to the parameters' probability distribution. These algorithms also tend to scale better with the number of model parameters than other MCMC algorithms. However, if one is working with black-box computer models (e.g., proprietary applications), the gradients may not be available, and other approaches, such as various forms of the Metropolis-Hastings algorithm, may be needed instead.[18] Unlike the probability distributions associated with OLS estimates of model parameters, the distributions from Bayesian methods can be used as inputs to the aforementioned aleatoric uncertainty propagation methods.[2]

2.4 Related Topics

2.4.1 Sensitivity Analysis

Sensitivity analysis (SA) evaluates how strongly each input of a computational model influences its output. In principle, it is not required to do UQ, but in practice, it may be used to pare down the number of inputs used in a subsequent uncertainty propagation analysis. There are several methods that may be used for SA. For example, the Monte Carlo sampling methods for uncertainty propagation described in

Section 2.2 may be modified by replacing an estimation of the probability distribution of the QoI with an estimation of the correlation between each input and the QoI. Another method decomposes the variance in the QoI into first-order contributions from each input, second-order contributions from the combined influence of 2 inputs, etc. This decomposition is then used to construct sensitivity indices for each input.[19–21] These are examples of *global* SA, which estimates the sensitivity of model outputs as model inputs vary over the entirety of their domains. There is also *local* SA, which estimates how model outputs vary with respect to perturbations in model inputs about some set of nominal input values. This may involve estimates of the values of the derivatives of the model output at these nominal values.[2]

2.4.2 Model Emulation

Computational models can be so expensive or time-consuming to execute that running hundreds or even thousands of instances of them—which would be needed for several UQ and SA methods—is unfeasible. An emulator may be used to work around this difficulty. This is an approximate model that is fit to a relatively small number of samples of the original computational model and then used in place of the original.[2] Techniques for constructing emulators include Kriging methods, multivariate adaptive regression splines (MARS), polynomial fits, and radial basis function expansions.[13,22] Even artificial neural networks may be used.[13] Emulators are also known as response surfaces, meta-models, or surrogate models.[2] One catch with the construction of emulators is that they may be subject to the curse of dimensionality. For example, with the standard methods for building a Kriging emulator, the number of samples of the computational model needed to ensure the accuracy of the emulator increases exponentially as the number of model parameters increases.[23] The computational cost of even determining a MARS response surface also increases exponentially as the number of model parameters increases.[24]

2.4.3 Optimization under Uncertainty

Optimization under uncertainty (OOU), also known as stochastic optimization,[25] describes processes or methods of finding a solution to a problem that is in some sense optimal, while still accounting for uncertainties in the information needed to solve the problem. Some OOU methods are used to minimize or maximize the mean value of some quantity, while others are used to minimize the likelihood that some quantity exceeds some threshold (e.g., a yield stress or some limit that would indicate failure if exceeded).[26] Also, methods are available for optimizing under both

aleatoric and epistemic uncertainties.[4] OOU often involves nesting an uncertainty propagation problem within an iteration of what would otherwise be a deterministic optimization problem,[4,25] thus combining the computational expenses of both optimization and UQ. Reducing this expense to allow OOU to be applied to nontrivial problems is an ongoing topic of research.[26]

3. Uncertainty Quantification and High-Performance Computing: General Issues

3.1 High-Performance Computing Clusters and Their Typical Usage

(This section is primarily intended for those altogether unfamiliar with how high-performance computing systems are used.)

Typically, HPC systems are clusters of interconnected computers, each of which is called a node, that behave almost as if they were a single very large computer. These nodes generally share at least one common parallel file system. Typically, a small fraction of these nodes are set aside for users to log in remotely and perform various operations on the cluster, and these are called *login nodes*. For the most part, the rest of the nodes are used for parallel computation, and these nodes are called *compute nodes*. To run parallel software applications on a cluster, users typically write *job scripts*, which are usually written in a scripting language, such as Bourne shell or C shell. These job scripts generally specify how many compute nodes on which the parallel software will run and the maximum amount of wall-clock time these nodes will be allowed to run the software, and these scripts also contain commands to launch the software with certain specified inputs. On login nodes, users submit these job scripts to a batch scheduler, which is responsible for actually running the job scripts, often in such a fashion that no single user takes up all of the cluster's computing resources. Although the batch scheduler routinely runs several job scripts at once (on different compute nodes, of course), the scheduler is nonetheless typically called a "queue" or "queuing system".[27]

3.2 Interfacing of UQ Software and External Applications

Many kinds of UQ software[4,22,28,29] are designed to interface with external software applications, which are treated as "black box" computational models. There are 3 general approaches to this. One of these, called here *launch-and-use*, is to have the UQ software launch either instances of the external application itself or instances of a wrapper script for the external application, and then have those instances return their model outputs to the UQ software for further analysis. The second approach, called here *launch-and-quit*, has 2 stages. The first stage is to have the UQ software launch either instances of the external application itself or instances of a wrapper script for the external application, and then quit. The instances of the external applications still keep running until they have written their model outputs to one or more files. In the second stage, the user takes the outputs from the external application to create input for the UQ software to postprocess and produce an analysis. The third approach, called here the *launchless* approach, has 3 stages. In the first stage, the UQ software writes model inputs (e.g., the results of a Monte Carlo sampling of inputs, or model input values at quadrature or collocation points) to one or more files. The second stage begins with the user taking what the UQ software has written in the first stage to generate input files for the external application and/or job scripts for an HPC cluster that will run instances of this application, possibly in parallel. The second stage ends when all instances of the application have finished running and produced their outputs. The third stage is essentially the same as the second stage of the launch-and-quit approach, that is, the user taking the outputs from the external application to create input for the UQ software to postprocess.

The launch-and-use approach can introduce complications when running UQ software on an HPC cluster. If a script is used and the external application is parallel, then this script will typically contain commands for launching the parallel application, such as `mpiexec` or, on a Craycluster, `aprun`, and HPC systems may have restrictions on where those commands may execute. For example, a system may not allow software launched with `mpiexec` to in turn launch a wrapper script containing `mpiexec`, which would limit the UQ software itself to run serially if the external application is parallel. The HPC cluster may also require that the arguments to the command for launching the parallel application ensure that proper tiling occurs. That is, if each instance of the external application is supposed to run on M nodes, and the UQ software is supposed to launch N parallel instances, then

not only does the job script that launches the UQ software need to allocate MN or $MN + 1$ nodes, but the wrapper script needs to provide the correct arguments to `mpiexec` (or a similar command) so that when an instance of the external application is launched, it runs on a free set of M nodes, rather than one on which another instance of the application is already running.[4] Examples of proper and poor tiling are shown in Fig. 1. Cray clusters introduce an additional complication because they have intermediary nodes between their login and compute nodes, called service nodes, on which the job scripts themselves are run.[30] Service nodes are not intended for heavy computation. Rather, the scripts that run on them are primarily intended to execute commands that launch parallel software on the compute nodes. Also, parallel software running on a Cray compute note cannot launch instances of parallel software to run on other compute nodes. This means that when a job script runs UQ software on a Cray cluster, only the external application launched by the UQ software actually runs on the compute nodes, while the UQ software itself is left running on a service node.[31] This may be an issue if the UQ software does significant computation that causes the service node to be overburdened.

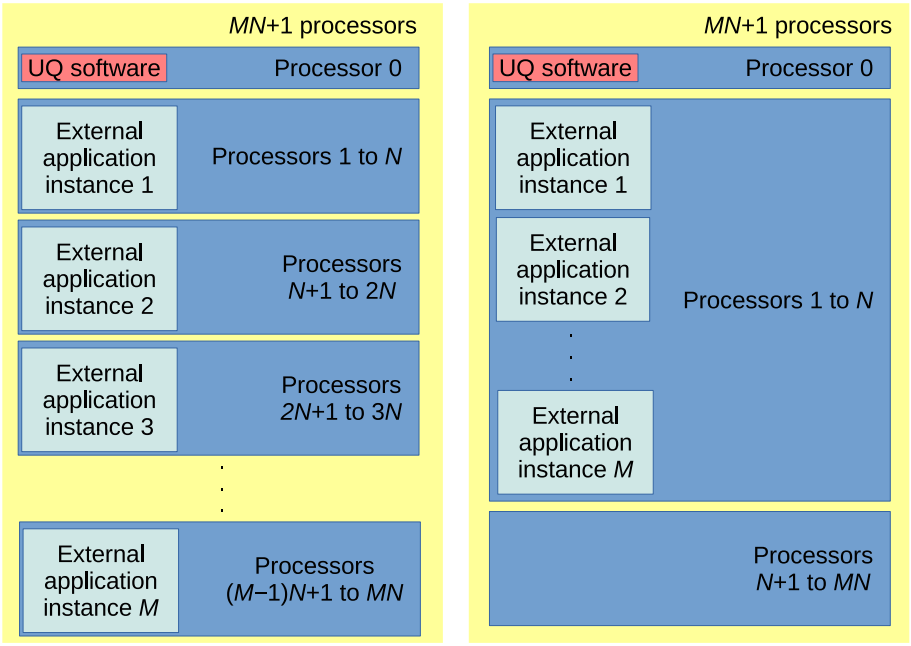

Fig. 1 (a) Proper tiling of the UQ software and N external application instances over $MN + 1$ processors. (b) Error in tiling where nodes $N+1$ to $2N$ have been inadvertently oversubscribed while the rest of the allocated processors are idle.

In the launch-and-quit approach, either the UQ software uses system calls that al-

low the external applications it launches to continue running after it quits, or—far more commonly—either the UQ software or the wrapper scripts submit to a queue the job scripts that run instances of the external application. The latter option may limit the UQ software to run on a login node, since an HPC cluster may not allow other kinds of nodes to submit jobs. Also, if each job script only runs a single instance of the external application, this may lead to submitting large numbers of jobs to a queue, which may be a problem if the HPC cluster limits the number of jobs that a single user can submit. However, if the UQ software is designed with the batch scheduler in mind, then it can write job scripts that run several instances of the external application in succession, each with its own set of model inputs, thus limiting the number of submitted jobs.[29] The launch-and-quit approach is tolerant of failures of the external application. For example, if one of the job scripts launching the external application fails to complete (perhaps because it specifies an insufficient amount of wall-clock time, or because a node malfunctions), that job can be restarted and finished before the second stage begins.

The launchless approach obviously does not have the problems of the launch-and-use approach above. It has the same fault tolerance as the launch-and-quit approach, and it allows the user much flexibility in how to interact with the queuing systems of various clusters. However, neither the launch-and-quit nor the launchless approaches work with UQ analyses that do not have distinct sampling and postprocessing stages. As an example, interval analysis may use optimization to find the minimum and maximum values of model outputs. This optimization can involve an iterative process, where the current iteration may entail generating a set of model inputs that depends upon model outputs calculated in a previous iteration.[4] For UQ analyses that cannot be divided into distinct stages, the launch-and-use approach is required.

4. Survey of Available Uncertainty Quantification Software

4.1 Dakota

Dakota[4] is an open-source project from Sandia National Laboratories, licensed under version 2.1 of the GNU Lesser General Public License (LGPL). It was originally designed in 1994 as an optimization toolkit, but its capabilities have since expanded to encompass UQ. Its primary interface is text-based, that is, users write input files specifying the desired UQ methods, various parameters, and so on. However, a GUI

front-end is available.[32]

The following available aleatoric uncertainty propagation methods are provided in Dakota: Monte Carlo sampling, using either ordinary random sampling or LHS; stochastic expansion, including both polynomial chaos and stochastic collocation; and reliability methods. Methods to propagate epistemic uncertainty include interval analysis and Dempster-Shafer theory. Dakota also has implementations of methods to account for combinations of aleatoric and epistemic uncertainties. For inverse UQ problems, Dakota supports Bayesian model calibration using gradient-free MCMC methods that can be used with black-box computational models. It also supports a frequentist approach via its nonlinear least-squares implementation, which can output confidence intervals for each model input.

Some SA capabilities are available in Dakota as well. When sampling with LHS, it outputs partial correlation coefficients and partial rank correlation coefficients for each model input.[4] A larger magnitude of a coefficient may indicate a larger sensitivity of the model output to the coefficient's corresponding model input. However, these coefficients should be interpreted with care, since the calculation of a partial correlation coefficient is based on the assumption that the relationship between model input and output is linear, and the calculation of a partial rank correlation coefficient is based on the assumption that the relationship between model input and output is monotonic.[19] Alternatively, SA may be performed by decomposing the variance (as discussed in Section 2.4.1), which in Dakota involves approximating integrals via Monte Carlo sampling. A third method provided by Dakota implements the algorithm from Campolongo et al.,[33] a modified form of the Morris "one-at-a-time" (MOAT) approach.[34] The latter 2 methods do not require any assumption of linearity or monotonicity.

Emulators of models can be constructed in Dakota via Kriging methods, MARS, polynomial fits, radial basis function expansions, and artificial neural networks. Dakota is also capable of performing OOU.

There is limited support for the launchless approach to interfacing with external software described in Section 3.2. When doing Monte Carlo sampling or SA, it can be executed in "pre-run" and "post-run" modes. The pre-run mode corresponds to the first stage of the launchless approach, and it consists of writing a table containing several sets of model inputs (e.g., Monte Carlo samples) to a text file. The

post-run mode corresponds to the third stage of the launchless approach, and it consists of reading a table of model inputs and outputs that had been written in the second stage, and then outputting the results of a UQ or SA analysis. Unfortunately, some analyses that could in principle be split into separate sampling and post-processing stages, such as stochastic collocation, cannot be used with Dakota's pre-run and post-run modes. The documentation of Dakota recommends emulating the launch-and-quit and launchless approaches by a workaround implemented with the launch-and-use approach to interfacing with external software. In this emulation, 2 wrapper scripts are used, one to elicit sets of parameters from Dakota and one to do post-processing. That said, this emulation entails that the first wrapper script return "dummy" model outputs, on which Dakota will waste computations. All forms of UQ and SA analysis support the launch-and-use approach. In Dakota's implementation of this approach, the external application, or the wrapper script to that application, reads in a file written by Dakota that contains model inputs and writes a file to be read by Dakota that contains the model outputs.

4.2 PSUADE

PSUADE[22] is from Lawrence Livermore National Laboratory. It appears to be distributed under the LGPL, but this is not entirely clear. (On the one hand, its source code archive contains a copy of version 2.1 of the LGPL, and comments in its source code indicate that it is under that license. On the other hand, the source code archive also contains a copyright statement with the admonition "Commercialization of this product is prohibited without notifying the Department of Energy (DOE) or Lawrence Livermore National Laboratory (LLNL)," which conflicts with the LGPL.[35,36]) PSUADE has 2 text-based interfaces. One of these is a batch interface where users provide input files specifying the desired UQ methods, various parameters, and so on. The other is an interactive command line interface.

Several Monte Carlo sampling methods for aleatoric uncertainty propagation, including ordinary random sampling and LHS, are available in PSUADE. It supports *mixed* aleatoric-epistemic uncertainty propagation as well, but support of propagation of purely epistemic uncertainties, such as Dempster-Shafer theory, is not currently documented. For inverse UQ problems, PSUADE supports Bayesian model calibration using MCMC methods, but it applies these methods to an emulator of the model rather than directly to the model itself. PSUADE constructs emulators of models via Kriging methods, MARS, polynomial fits, and radial basis function

expansions. It is also capable of performing OOU. Gan et al. have documented the wide range of global SA capabilities in PSUADE.[21] The modified form of the MOAT algorithm[34] from Campolongo et al.[33] is supported. PSUADE also supports several methods of decomposing the variance (as discussed in Section 2.4.1). This includes not only approximating integrals via Monte Carlo sampling, but also the Fourier amplitude sampling test (FAST).[37] After performing Monte Carlo sampling, PSUADE can output partial correlation coefficients and partial rank correlation coefficients for each model input.[21,22] If the relationship between model inputs and outputs is linear or monotonic, then a larger magnitude of a coefficient can indicate a larger sensitivity of the model output to the coefficient's corresponding model input. The documentation of PSUADE does not recommend using correlation coefficients for non-monotonic models. Other SA methods supported include the Delta test[38] and the sum-of-trees method.[39]

PSUADE supports all 3 approaches to interfacing with external software described in Section 3.2. When using the launch-and-use and launch-and-quit approaches, the external application, or the wrapper script to that application, takes 2 arguments: the name of a file written by PSUADE that contains model inputs, and the name of a file that, in launch-and-use mode, would contain the model outputs and be read by PSUADE. To support the launchless approach, PSUADE can write the parameter files for every set of desired inputs and then quit. In the launch-and-quit and launchless approaches, the outputs from the launched instances of the external software are harvested and collected into a file that PSUADE can read in order to finish its analysis.

4.3 OpenTURNS

OpenTURNS[28] is an open-source Python module, licensed under version 3 of the GNU General Public License (GPL). It is the result of a collaboration of academic institutions and the French industrial companies Airbus Group, Électricité de France Research and Development, Phimeca Engineering, and Ingénierie Mathématique et Calcul Scientifique.[40] Users utilize OpenTURNS by writing Python scripts that import the module and use its functions and class methods to execute various steps of a UQ analysis.

This Python module provides several methods for propagating aleatoric uncertainty, such as Monte Carlo sampling, including ordinary random sampling and LHS; poly-

nomial chaos; and reliability methods. Support for propagating epistemic uncertainties is not currently documented. Bayesian model calibration using MCMC methods is available for solving inverse UQ problems. SA capabilities include determination of partial correlation coefficients and partial rank correlation coefficients for each model input, where—if the relationship between model inputs and outputs is linear or monotonic—a larger magnitude of a coefficient can indicate a larger sensitivity of the model output to the coefficient's corresponding model input. SA can also be done in OpenTURNS via several methods of decomposing the model output variance (as discussed in Section 2.4.1). Emulators may be constructed by Kriging methods, and also by approximating the computational model as an expansion in an orthogonal basis, where the basis functions may be polynomials, sinusoidal functions, or wavelets.

OpenTURNS does not *explicitly* support interfacing with external software. Instead, users supply Python classes that implement computational models, which can then be used as arguments to various functions and class methods in OpenTURNS. However, the functionality in Python to call external processes allows OpenTURNS to interact with outside applications in the launch-and-use approach described in Section 3.2. There is also support for the launch-and-quit and launchless approaches (also described in Section 3.2) via the `NumericalSample` class. This class holds a sequence of vectors, each of which can be a realization of model inputs or outputs. In the first stage of the launchless approach, a Python script can use OpenTURNS to create a sample of model inputs stored in an instance of the `NumericalSample` class, and then write the contents of this instance to a file. In the last stage of the launchless approach, a Python script can use OpenTURNS to read the outputs of a computational model into an instance of this `NumericalSample` class. Various functions and class methods in OpenTURNS are capable of taking instances of the `NumericalSample` class as inputs to perform various analyses.

4.4 Chaospy

Chaospy[41] is an open-source Python module, licensed[42] under the MIT license.[43] It implements 2 methods of uncertainty propagation: polynomial chaos and Monte Carlo sampling. There is also additional functionality for calculating some statistics of the outputs of these methods, such as the moments of the probability distributions of model outputs. The module does not *explicitly* support interfacing with external software. Rather, ChaosPy functionality is used to generate a list of model inputs,

either random samples[7] or quadrature points.[41] Other functions of ChaosPy then accept this list of model inputs and a corresponding list of model outputs. Generation of the latter list is left up to the user, and this generation can be implemented through any of the launch-and-use, launch-and-quit, or launchless approaches discussed in Section 3.2. Python's capability to call external software applications facilitates the first 2 approaches. Python functionality can also be used to write the list of model inputs to one or more files, implementing the launchless approach.

4.5 QUESO

QUESO[44,45] is an open-source C++ library licensed under version 2.1 of the GNU LGPL. However, since it depends upon the GNU Scientific Library, which is licensed under the GNU GPL, it effectively is under the GPL itself.[46] The QUESO library is primarily designed for solving inverse UQ problems[47] via a Bayesian approach that uses MCMC. It has some capability for solving uncertainty propagation problems, but rather than use the "embarrassingly parallel" Monte Carlo sampling methods described in Section 2.2, it uses a Monte Carlo method similar to the one that it uses for inverse problems, where there is a chain of random states, each of which is dependent on the previous state. While generation of a chain of random states (for either forward or inverse problems) is a serial operation, multiple chains may be generated in parallel. Computational models used with QUESO may also be parallel.

Users employ QUESO by writing C++ programs that use the functionality from the QUESO library. They write C++ classes that use computational models, which certain QUESO classes then use as inputs. While interfacing with external software is not *explicitly* supported, there is indirect support since C++ supports using functions that call external software programs. Due to QUESO's use of chains of states in which the last state depends on the previous one, only the launch-and-use approach to interfacing with external applications (see Section 3.2) is supported.

Dakota can be compiled against the QUESO library in order to use the library's inverse UQ capabilities.[4]

4.6 PyMC and PyMC3

PyMC[48] and PyMC3[49] are open-source Python modules that implement Bayesian model calibration via MCMC. PyMC is licensed under the Academic Free License,[50] while PyMC3 is under version 2.0 of the Apache License. PyMC3, in addition to having a different programming interface from PyMC, also implements Hamiltonian Monte Carlo and automatically determines the gradients needed for that MCMC method. In principle, external software applications may be invoked using normal Python functionality so that they may be used with PyMC and PyMC3. However, unless gradients for the outputs of the external software are supplied, only the gradient-free MCMC implementations in PyMC3 may be used when interfacing with external software. (In PyMC, all available MCMC implementations are gradient-free.) Since MCMC entails generating chains of random states in which the last state depends on the previous one, only the launch-and-use approach to interfacing with external applications (see Section 3.2) is supported.

4.7 Stan

Stan[51] is open-source software, licensed under the 3-clause BSD License,[52] that implements Bayesian model calibration via MCMC. The documentation for Stan does not explicitly identify it as UQ software. It has a compiler that translates a domain-specific language into C++ code, and tools for creating from that code either command-line executables or objects in the Python or R languages. The domain-specific language specifies the computational model and the model parameters to be calibrated. In addition to the Python and R interfaces, there are also interfaces to Stan from MATLAB, Julia, Stata, and Mathematica. Stan can incorporate externally defined C++ functions into the executable objects that it creates, and in limited cases, this *may* allow external software to be used with Stan via the standard C++ functions that can call external software applications. However, in Stan, there is an additional complication to interfacing with external code, because it uses Hamiltonian Monte Carlo, which requires gradients. Normally, these would be supplied via the auto-differentiation functionality in Stan (so that users would not need to concern themselves with gradients), but in C++ code used to interface with the external application, gradients pertaining to the output from that application may need to be manually supplied.[53] If these cannot be supplied, one may need alternatives to Stan that do not require gradients, such as QUESO (Section 4.5), PyMC (Section 4.6), or UQTk (Section 4.14).

4.8 UQTools

UQTools[54] is a MATLAB toolbox from NASA that is available without charge to qualified American citizens that request it.[55] Unlike many works of UQ software, UQTools is focused mainly on reliability methods, including one developed by its authors.[12] In this method, the space of model inputs is divided into 2 sets, a failure domain, consisting of model inputs that would yield undesirable model outputs, and the complement to this domain, that is, the safe domain. A hyper-sphere or hyper-rectangle is centered around a point within the safe domain and then expanded (or contracted) until it just borders the failure domain. The lower bound on the probability that the system will not fail is taken to be the probability that the model inputs are within the expanded (or contracted) hyper-sphere or hyper-rectangle. In addition to support for reliability methods, UQTools also has limited support for SA, namely an implementation of local SA developed by the authors of UQTools.[56] There is also some support for model emulation using either polynomials or radial basis functions.

There is no explicit support for interfacing with external applications, but MATLAB does have support for calling external software commands. Some of the functionality in UQTools, such as that for SA, allow the user to provide model inputs and outputs as arguments to MATLAB functions. This allows for a launchless approach to interfacing with external software (see Section 3.2). On the other hand, the optimization algorithms involved in the implementations of the reliability methods make the launch-and-use approach to interfacing with external software unavoidable.

4.9 PUQ

PUQ[29] is an open-source Python-based framework based upon work supported by the National Nuclear Security Administration. It is available under the MIT license.[43] It consists of a command-line program called "puq" that takes control scripts as input. These control scripts are written in Python and import a module that is also called `puq`. Alternatively, a user may avoid using the command-line program and write a Python script (which still uses the "puq" module) that runs a UQ analysis.

Currently, the only available aleatoric uncertainty propagation methods available in PUQ are Monte Carlo sampling methods and polynomial chaos. The sampling

methods supported are ordinary random sampling and LHS. PUQ does not currently offer epistemic uncertainty propagation. To solve inverse UQ problems, PUQ uses the external python module PyMC,[48] which performs Bayesian model calibration. The documentation for PUQ describes its model calibration functionality as "in progress". The only documented method for SA is the modified form of the MOAT algorithm[34] from Campolongo et al.[33]

Interfacing with external software is explicitly supported, but only via the launch-and-use approach described in Section 3.2. The software is expected to accept input parameters at the command line and dump its output to a file in a format readable by PUQ. PUQ can launch the external software directly. In order to launch multiple instances of the external software in parallel, PUQ can also launch the external software by automatically generating job scripts in the Portable Batch System (PBS) format, and submitting those scripts to a queue. However, instead of submitting the jobs and quitting, PUQ monitors the progress of jobs in the queue and reads the outputs from the jobs that have finished. PUQ continues running until all outputs have been collected and analyzed. From the user's perspective, the main difference between a direct launch and launching via a PBS queue is that the latter runs jobs in parallel, and there are additional parameters required for the latter approach, such as the maximum wall-clock time per script, or the number of instances of the external software run in succession by each automatically generated job script. Users are not expected to submit the job scripts manually or even to have direct access to them.

4.10 UQ-PyL

UQ-PyL[57] is an open-source software application written in Python whose functionality may be accessed through either a graphical user interface or through Python scripts that import the module "UQ".[58] It is available under the GNU GPL, version 3.[59] Monte Carlo sampling methods for aleatoric uncertainty propagation, such as ordinary random sampling and LHS, are supported. The user manual indicates that MCMC is supported,[58] but this functionality is not exposed through the graphical interface. Various global SA methods are supported, such as the modified form of the MOAT algorithm[34] from Campolongo et al.[33] algorithm and several methods of decomposing the variance, including the Fourier Amplitude Sampling Test (FAST).[37] UQ-PyL is also capable of constructing emulators of models via support vector machines,[60] various generalized linear models,[61] MARS, and Gaussian

process regression.

UQ-PyL indirectly supports interfacing with external software. Sampled model inputs are written to files, and model outputs are read from files. The documentation indicates that the model outputs are supposed to be written by using Python modules that define functions that are expected to call external applications. However, the use of such scripts does not appear to require this, so both the launch-and-use and launchless approaches to interfacing with external software (described in Section 3.2) appear to be supported.

4.11 SmartUQ

SmartUQ[62] is a proprietary software application with a graphical user interface. Publicly available documentation for this software is currently limited to introductory descriptions of its features on pages of the web site for SmartUQ and a YouTube video demonstrating its capabilities.[63] No user manual appears to be offered. From what is publicly available, it is not clear what the supported operating systems of SmartUQ are, but the user interface shown in one of the videos appears to indicate that it is available at least for Windows.

There are 2 ways of performing uncertainty propagation in SmartUQ, polynomial chaos and a so-called "emulator-based" method. The details of the latter method are not described, but the output of the method is a probability distribution of the model outputs, characterized via both moments and histograms. It is not clear what algorithms are used to implement emulators in SmartUQ. Inverse UQ is also available, though it is not clear if it is Bayesian or if it even involves MCMC. SA capabilities in SmartUQ are described on its website as being implemented via polynomial chaos and "emulator-based" methods. OOU is also available, and it is done through a combination of proprietary sampling techniques and emulators. Interfacing with external software appears to largely if not entirely be done via a launchless approach (see Section 3.2), where SmartUQ writes model inputs to comma-separated value files that are then used to generate input files for external software.

4.12 OpenCOSSAN and COSSAN-X

OpenCOSSAN[64] is an open-source MATLAB toolbox and has been described as the "computational core of the COSSAN project". The Institute for Risk and Uncertainty at the University of Liverpool, UK, has made it available under the LGPL. COSSAN-X is proprietary UQ software with a graphical user interface, available under a commercial or academic license, and although it appears to be based on OpenCOSSAN, it does not require a MATLAB license to use.[65]

COSSAN-X has several toolboxes: (1) a UQ toolbox, which implements aleatoric uncertainty propagation via Monte Carlo sampling (including purely random sampling and LHS); (2) a reliability toolbox, which implements reliability methods; (3) an optimization toolbox, which implements reliability-based optimization, a form of OOU; (4) a meta-modeling toolbox, which implements model emulation using neural networks or polynomial fitting; (5) a stochastic finite elements toolbox, which implements stochastic finite element methods; (6) a SA toolbox, which implements both local SA and global SA, with the latter including variance decomposition methods such as the FAST method[37]; and (7) an HPC toolbox, which allows COSSAN-X to submit jobs to grid computing systems that run external applications that implement computational models. COSSAN-X is designed to interface with certain third-party finite element solvers, such as NASTRAN, ABAQUS and ANSYS, though it can interface with other external applications as well. It is designed to interface with external software via the launch-and-use approach (see Section 3.2). OpenCOSSAN has the same capabilities as COSSAN-X, but users interact with it via the MATLAB command line or scripts.

4.13 MUQ

MUQ[66] is an open-source C++ library that also provides wrappers so that its functionality may be accessed via Python modules. It is made available under the 3-clause BSD License.[52] Currently, documentation for this library is limited, and includes a user manual with several empty chapters,[67] online Doxygen[68] pages for reference documentation, and some examples on the website for MUQ. Uncertainty propagation capabilities appear to be limited to polynomial chaos methods. MUQ, though, does provide Bayesian model calibration via MCMC for inverse UQ problems.

MUQ does not *explicitly* support interfacing with external software. Instead, users

supply C++ or Python classes that implement computational models, which can then be used as arguments to various functions and class methods in the MUQ library. Both C++ and Python, though, support using functions that call external software programs. Since MCMC entails generating of chains of states in which the last state depends on the previous one, only the launch-and-use approach to interfacing with external applications (see Section 3.2) is supported for the inverse UQ functionality. The adaptive polynomial chaos methods in MUQ, where the order of the polynomials is adjusted until the error between the polynomial expansion and the model is within some tolerance, is also limited to the launch-and-use approach as well. Polynomial chaos methods utilizing a fixed number of terms are available, but these also appear to be limited to using the launch-and-use approach, since functionality for allowing users to access the values of the model inputs at quadrature points is not documented. This means that users cannot use a launchless approach where, for example, they write the model inputs at the quadrature points to a file so that they can go on to calculate the values of model outputs without using MUQ.

4.14 UQTk

UQTk[69] is an open-source collection of libraries and command-line utilities from Sandia National Laboratories, licensed under the LGPL. The libraries are written in C++ but may be accessible from a Python interface. The command-line tools largely focus on aspects of polynomial chaos expansion. For example, one tool generates quadrature points, while another generates a polynomial chaos expansion from model responses sampled at those points, and another extracts moments and sensitivity indices from the expansion. However, there is also a tool that constructs model emulators via Gaussian process regression, a tool that performs Bayesian model calibration via MCMC, and a tool that generates sensitivity indices from Monte Carlo sampling. UQTk support for Monte Carlo sampling includes both ordinary random sampling and LHS, and outside of the supplied command-line tools it can be applied to more than just SA.

Different aspects of the functionality of the UQTk libraries have different means of interfacing with external software, and the command-line tools—which are thin wrappers around the functionality of the UQTk libraries—reflect this. For example, the tools for polynomial chaos entail the launchless approach, since the tools take as input a file containing sample responses from a computational model. On the other hand, the tool that performs MCMC has to employ the launch-and-use approach,

due to the nature of the computational method that it implements.

5. Application of Uncertainty Quantification to Modeling of Dynamic Penetration of Armor

As an example of how well the current state of the art in UQ can address Army-relevant problems, existing UQ software is applied to a model case of dynamic penetration of armor, illustrated in Fig. 2. The UQ software used for most of the following analyses is Dakota,[4] since it is widely used and has a broad range of well-documented features. Penetration simulations are done with CTH,[70] an Eulerian finite element code, and the setup for these simulations largely follows that of Hornbaker.[71] The penetrator is a cylindrical 90%W–7%Fe–3%Ni tungsten alloy 131W rod, tapered at one end, with a diameter of 0.91 cm and a length of 13.1 cm. The initial velocity of the penetrator is 1.28 km/s. The witness (i.e., the target) is rolled homogeneous armor (RHA), whose depth is taken to be effectively semi-infinite. To track the QoI, that is, the penetration depth of the projectile, 20 tracers are placed along the length of the projectile, and 4 tracers are placed on a plane 15 cm from the front of the target. The latter 4 tracers are evenly spaced from each other and approximately 10.5 cm away from the z-axis, which is also the center line along the length of the projectile. The projectile moves in the positive z-direction. The penetration depth, then, is estimated according to the following formula,

$$PD = 15 \text{ cm} - \left[\max(\{Z_1^p, Z_2^p, \dots\}) - \frac{Z_1^w + Z_2^w + Z_3^w + Z_4^w}{4} \right], \qquad (1)$$

where Z_i^p is the z-coordinate of the i^{th} tracer on the penetrator, and Z_i^w is the z-coordinate of the i^{th} tracer on the witness. The Johnson–Cook model[72] is used for both the penetrator and the witness. In this model, the von Mises flow stress is taken to be

$$\sigma_{\text{flow}} = (A + B\epsilon^n)\left(1 + C \ln \frac{\dot{\epsilon}}{\dot{\epsilon}_0}\right)\left[1 - \left(\frac{T - T_{\text{room}}}{T_{\text{melt}} - T_{\text{room}}}\right)^m\right], \qquad (2)$$

where ϵ, $\dot{\epsilon}$, and T are the equivalent plastic strain, the plastic strain rate, and the temperature, and A, B, n, C, and m are fitting parameters. Parameter T_{melt} is the melting temperature of the material. The room temperature, T_{room}, is taken to be 298 K, and $\dot{\epsilon}_0$ is 1 s^{-1}. The baseline material parameters for the penetrator and witness materials are taken from Hornbaker[71] and shown in Table 1. The calculated penetration depth and wall clock time for a given simulation depend upon the size of

the finite elements in the regular mesh used in CTH, as shown in Table 2. The results in that table are from simulations executed on 2 nodes of a Cray XC40 computing cluster. Each node of the cluster has 32 Intel Xeon E5-2698 v3 cores, each with a clock speed of 2.3 GHz. The material parameters for these simulations are those in Table 1. Using a coarse mesh with an element size of 0.2 cm leads to a wall clock time about 175 times smaller than the one for the finer mesh with elements of size 0.05 cm, while underestimating the penetration depth by about 15%.

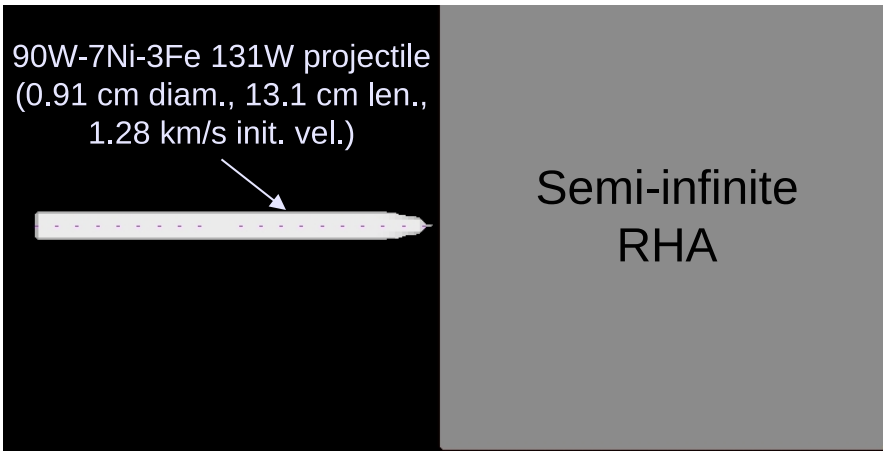

Fig. 2 Diagram of example dynamic armor penetration problem. The penetrator is a cylindrical 90%W–7%Fe–3%Ni tungsten alloy 131W rod, tapered at one end, with dimensions of 0.91 cm in diameter and 13.1 cm in length. The target is rolled homogeneous armor (RHA), whose depth is taken to be effectively semi-infinite.

Table 1 Baseline Johnson–Cook plasticity parameters and Poisson's ratio (ν) for the penetrator and witness materials

Parameter	Penetrator	Witness
A (MPa)	1507	780
B (MPa)	176.6	780
C	0.016	0.004
n	0.12	0.106
m	1.00	1.00
T_{melt} (K)	1723	1783
ν	0.310	0.294

Table 2 Calculated penetration depth and wall clock time versus element size, given baseline material parameters

Element size (cm)	Penetration depth (cm)	Wall clock time
0.05	8.7	17 h, 30 min
0.1	8.2	1 h, 20 min
0.2	7.5	6 min

5.1 Sensitivity Analysis

The modified MOAT algorithm,[33] as implemented within Dakota,[4] is used to compute sensitivity indices. This algorithm takes the following as input: (1) the desired number of samples of the computational model and (2) the number of levels, that is, the number of discrete values that each parameter can take on. For each parameter, the algorithm computes 2 indices, μ^*, a first-order sensitivity index that indicates how the response of the model is directly affected by changes to an input parameter, and σ, a second-order sensitivity index that indicates how much interaction a parameter has with other parameters.

A coarse mesh (0.2-cm element size) is used here for SA, since the values of the penetration depth are of less importance than how they vary with the model parameters. While there is wide range of model parameters that could potentially be taken as uncertain,[71] for the purposes of this example study, only the parameters in Table 1 are taken as uncertain here. Initially, the values of the uncertain parameters are taken to be within $\pm 10\%$ of their baseline values, and MOAT runs are performed for several numbers of samples and levels. For the MOAT run with 105 samples, 4 levels are used. The MOAT runs with 1500 and 3000 samples use 4, 8, 16, and 32 levels. The results of these runs are summarized in Fig. 3, where the error bars are used to indicate the minimum and maximum values of μ^* and σ for each pa-

rameter across all the runs. According to this MOAT analysis, the penetration depth is most sensitive to the Johnson–Cook parameters A and B of the witness, though the melting temperature of the penetrator appears to have significant second-order sensitivities.

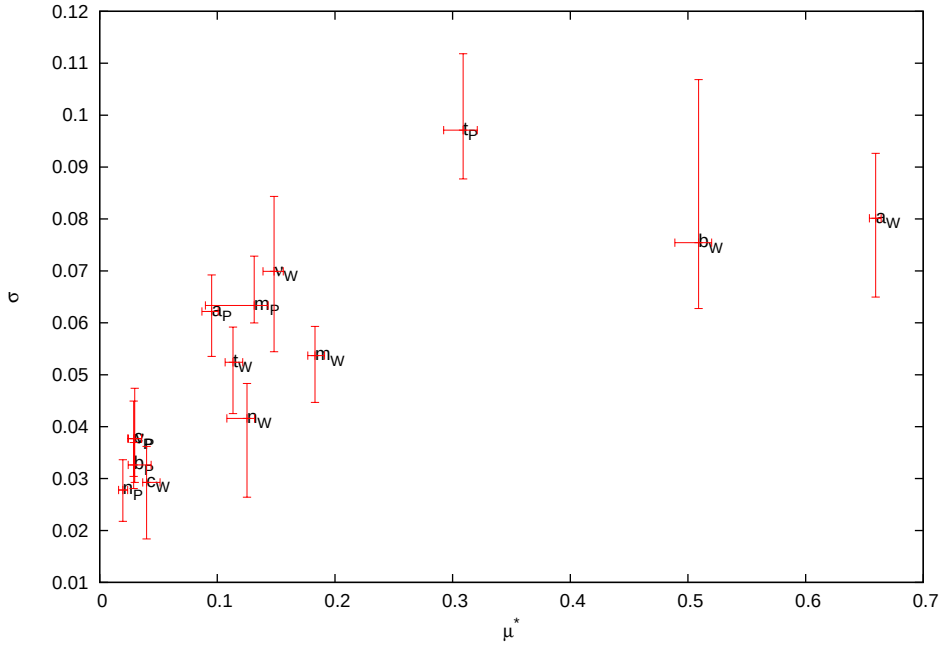

Fig. 3 MOAT results, assuming that all uncertain parameters are within ±10% of their baseline values

To corroborate the results from the MOAT runs, a second SA method is used, the FAST algorithm.[37] Since this algorithm is not available in the version of Dakota used for these analyses, the implementation of FAST from PSUADE[22] is used here. The results from applying the FAST method are to be compared to those of the MOAT analyses employing 3000 samples. However, implementation of the FAST method restricts the number of samples used to be certain integer values, so the number of samples used with this method is not 3000 as in MOAT, but rather 3657. A comparison of the sensitivity indices $\{\mu_i^*\}$ from the MOAT method with sensitivity indices from the FAST method is shown in Fig. 4, and the results from the 2 methods appear to confirm each other. The approximate computational costs of running sensitivity analyses with MOAT and FAST are shown in Table 3.

Table 3 Approximate computational costs of sensitivity analysis

Method	Number of samples	Estimated CPU hours
MOAT	3000	19,000
FAST	3657	23,000

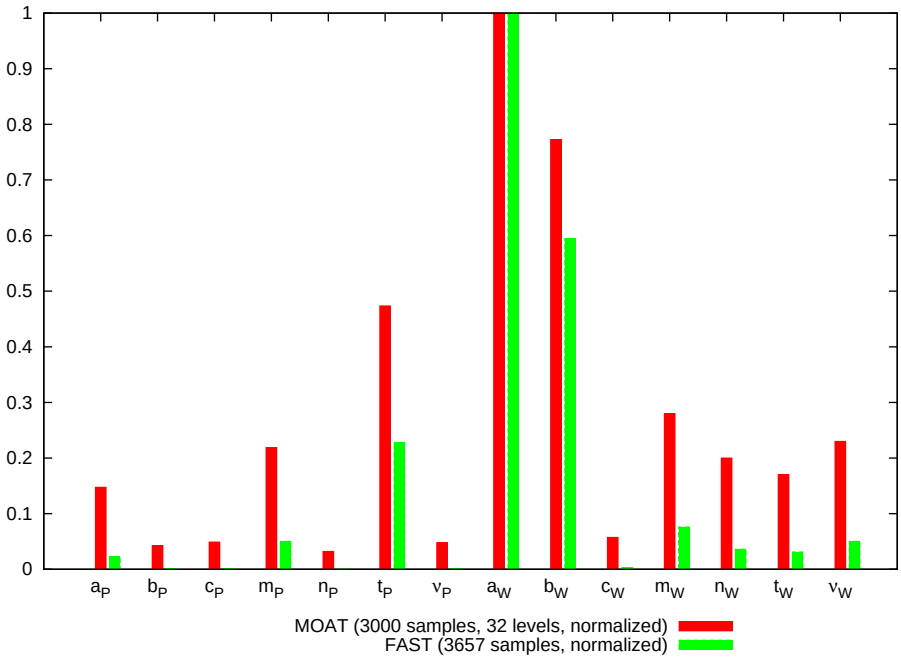

Fig. 4 Comparison of sensitivity indices $\{\mu_i^*\}$ from the MOAT method with sensitivity indices from the FAST method, assuming that all uncertain parameters are within $\pm$10% of their baseline values. Sensitivity indices have been normalized so that the largest index equals one.

As mentioned before, in the previous analysis, the values of the uncertain parameters are taken to be within $\pm 10\%$ of their baseline values. However, an expert opinion[73] has indicated that the uncertainty bounds for Johnson–Cook parameters C, n, and m should be revised, and recommends that C be taken to be within $[0, 0.04]$, n be taken to be within $[0.02, 0.15]$, and m be within $[0.85, 1.15]$, due to problems in how the Johnson–Cook functional form accounts for the strain rate and temperature sensitivity. These new bounds have been applied to the Johnson–Cook parameters of both penetrator and witness, leading to the SA results shown in Fig. 5, where the penetration depth is most sensitive to the Johnson–Cook parameters C and n of the witness. Taking into account both sets of sensitivity analyses, the 4 most sensitive parameters are found to be A, B, C, and n of the witness.

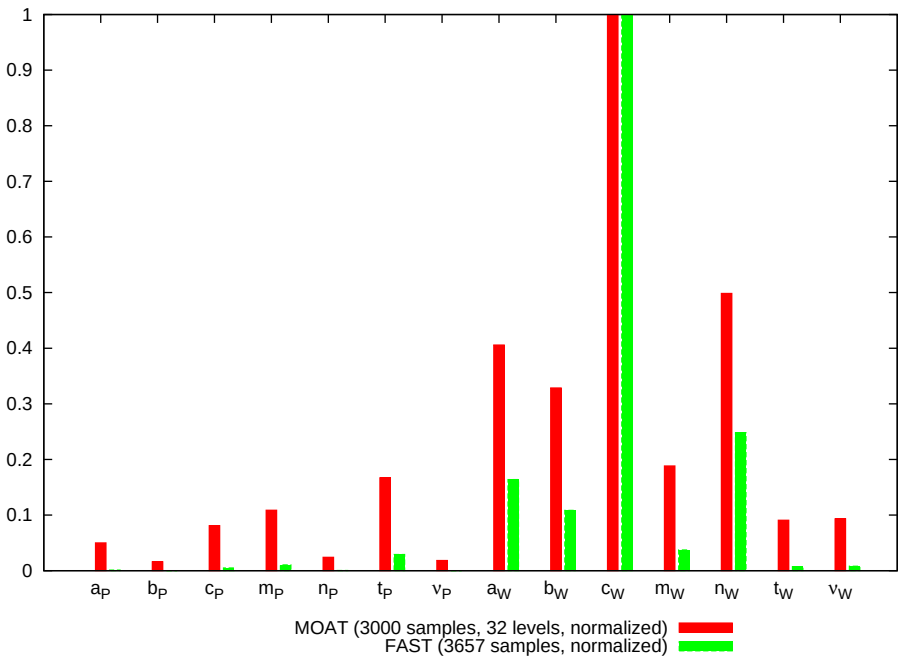

Fig. 5 Comparison of sensitivity indices $\{\mu_i^*\}$ from the MOAT method with sensitivity indices from the FAST method, using revised uncertain bounds. Sensitivity indices have been normalized so that the largest index equals one.

5.2 Uncertainty Propagation through Coarse Computational Model

The initial rounds of uncertainty propagation analysis have been applied to a coarse discretization of the dynamic penetration model, with an element size of 0.2 cm. Simulations have been performed for three sets of uncertain parameters: all the Johnson–Cook parameters and Poisson's ratios for both penetrator and witness, 14 parameters in total; the Johnson–Cook parameters and Poisson's ratio for the witness only, 7 in total; and the 4 most sensitive parameters as determined from the previously done sensitivity analyses (i.e., A, B, C, and n of the witness). The bounds assumed for any uncertain parameters are shown in Table 4. These bounds are the same as those used in the second round of sensitivity analyses, where most of the bounds are taken to be $\pm 10\%$ of their baseline values, but the bounds for C, n, and m are those recommended by an expert opinion.[73] Parameters not treated as uncertain are assigned the baseline values shown in Table 1.

Table 4 Assumed bounds for the uncertain parameters in uncertainty propagation simulations

Parameter	Penetrator	Witness
A (MPa)	1356.3–1657.7	702.0–858.0
B (MPa)	158.94–194.26	702.0–858.0
C	0–0.04	0–0.04
n	0.02–0.15	0.02–0.15
m	0.85–1.15	0.85–1.15
T_{melt} (K)	1550.7–1895.3	1604.7–1961.3
ν	0.279–0.341	0.2646–0.3234

5.2.1 Interval Analysis

Since probability distributions for the uncertain parameters are not known, interval analyses are performed via Dakota. Two methods are used to estimate the minimum and maximum values of the penetration depth. One method is crude but easily parallelized. It involves taking an LHS of the computational model (i.e., the CTH simulation of dynamic penetration). For each set of sampled model parameters, a value for the penetration depth is determined, and the smallest and largest of these values are taken to be the minimum and maximum values of the penetration depth. One may obtain increasingly accurate estimates of the minimum and maximum by extending the previous LHS (i.e., keeping the original samples from LHS and adding additional samples, which in Dakota requires doubling the number of samples[4]) rather than generating a new set of samples afresh. The advantage of this method is that the evaluations of the computational model for each LHS are independent and thus can be run in parallel. The disadvantage is that the samples are spread evenly over the parameter space rather than concentrated in the regions of the space that would be more likely to yield minima and maxima of the computational model. The other method, efficient global optimization (EGO)[74], samples the computational model where a so-called "expected improvement" function is maximal, so it generally requires far fewer evaluations of the model than the previous method. However, at each iteration of the EGO algorithm, the determination of where to next sample the computational model depends upon the evaluations of the model that have been done in previous iterations. This forces evaluations of the computational model to be done in sequence rather than in parallel.

Figures 6 and 7 show the results of interval analyses that have been done using LHS and EGO. The abscissas in these figures are for the number of samples used in LHS *only*. (The numbers of samples used for the EGO runs are shown in Table 5.) For

both LHS and EGO, reducing the number of uncertain parameters leads to overestimating the minimum and underestimating the maximum values of the penetration depth. The figures also show that LHS, even with a sample size of over 500,000, consistently overestimates the minimum and almost consistently underestimates the maximum. Furthermore, it does this while having a far greater computational cost than EGO, as shown in Table 5. The computational cost of EGO does depends, of course, on the number of iterations needed to reach convergence within a certain tolerance (and this tolerance is hard-coded in Dakota's implementation of EGO). It also depends strongly on the number of uncertain parameters, in contrast to the interval analyses using LHS.

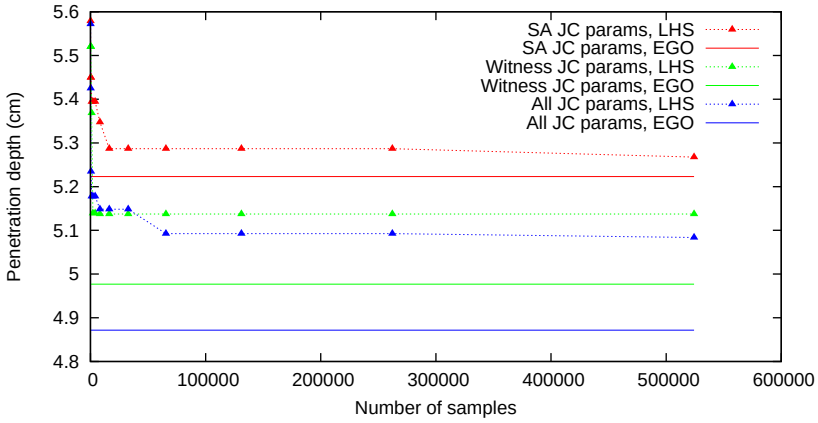

Fig. 6 Minimum penetration depth as a function of the number of samples used in LHS, using coarse computational model

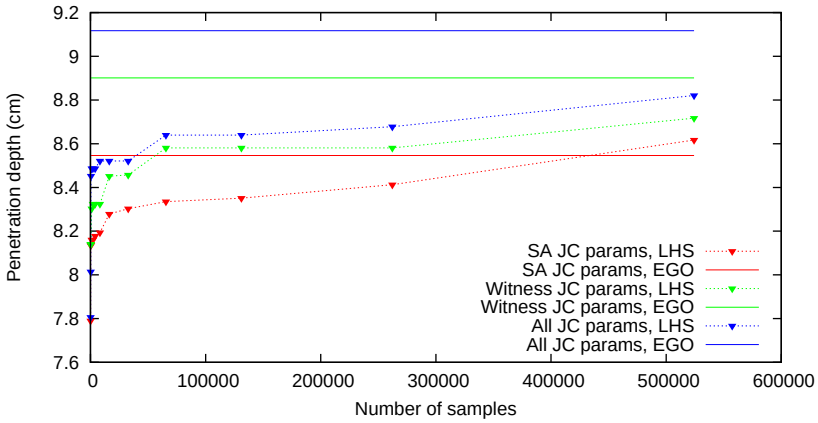

Fig. 7 Maximum penetration depth as a function of the number of samples used in LHS, using coarse computational model

Table 5 Interval analysis results with approximate computational cost, using coarse computational model

Method	Number of uncertain parameters	Number of samples	Minimum (cm)	Maximum (cm)	Estimated CPU hours
LHS	4	524,288	5.26757	8.61692	3×10^6
	7	524,288	5.13744	8.71716	3×10^6
	14	524,288	5.08354	8.8207	3×10^6
EGO	4	25	5.22289	8.54611	160
	7	45	4.97696	8.90132	290
	14	126	4.87181	9.11674	800

5.2.2 Aleatoric Uncertainty Propagation

In addition to the interval analysis, uncertainty propagation methods suited for aleatoric uncertainties are also applied to the dynamic penetration problem. However, since probability distributions for the uncertain parameters are not known, the results of these analyses should be treated with caution. In order to apply the methods for aleatoric uncertainties, the probability density functions of the uncertain parameters are taken to be uniform, with the bounds as shown in Table 4. Two methods are used to estimate the moments of the probability density function of the penetration depth: Monte Carlo using LHS, and stochastic collocation using Smolyak sparse grids. Both of these are done with Dakota.

Tables 6 through 8 show the mean, standard deviation, skewness, and kurtosis of the Latin hypercube samples that have been used in the interval analysis discussed in Section 5.2.1. In the source code of Dakota, the calculated skewness and kurtosis are equivalent to the following formulas found in Joanes and Gill.[75]

$$G_1 = \frac{\sqrt{M(M-1)}}{M-2} \frac{m_3}{m_2^{3/2}}, \tag{3}$$

$$G_2 = \frac{M-1}{(M-2)(M-3)} \left\{ (M+1) \left[\frac{m_4}{m_2^2} - 3 \right] + 6 \right\}, \tag{4}$$

where M is the number of samples and

$$m_r = \frac{1}{M} \sum_{i=1}^{M} (x_i - \bar{x})^r, \tag{5}$$

where x_i is one of the M samples of the QoI, and $\bar{x}$ is the mean of the QoI. The num-

ber of uncertain parameters changes the values to which the moments of the samples appear to have converged, and this effect is stronger for higher order moments. The mean varies from about 6.63 to 6.67 cm, a relative difference of about 0.6%. The standard deviation varies from about 0.55 to 0.57 cm, a relative difference of about 4%. For the kurtosis in particular, if only the 4 most sensitive parameters are taken as uncertain, it is approximately -0.72, but if all 14 of the parameters in Table 1 are taken as uncertain, it becomes closer to -0.56, a relative difference of 28%.

Table 6 Moments of estimated probability distribution of penetration depth, determined via LHS applied to a coarse model, where the 4 most sensitive parameters are taken as uncertain

Number of samples	Mean (cm)	Standard deviation (cm)	Skewness	Kurtosis
128	6.62668	0.557501	0.189498	−0.961469
256	6.62592	0.553948	0.295482	−0.729879
512	6.62629	0.547239	0.205096	−0.726831
1,024	6.62706	0.547204	0.20364	−0.702608
2,048	6.62856	0.548484	0.208008	−0.752213
4,096	6.62902	0.550569	0.242908	−0.744212
8,192	6.62906	0.549154	0.237961	−0.729982
16,384	6.62874	0.548376	0.231745	−0.72422
32,768	6.62888	0.548611	0.233957	−0.723087
65,536	6.62905	0.548475	0.232121	−0.727582
131,072	6.6293	0.548469	0.226063	−0.73075
262,144	6.62938	0.548561	0.224258	−0.726403
524,288	6.63129	0.549032	0.225786	−0.721545

Table 7 Moments of estimated probability distribution of penetration depth, determined via LHS applied to a coarse model, where the Johnson–Cook parameters and Poisson's ratio of the witness are taken as uncertain

Number of samples	Mean (cm)	Standard deviation (cm)	Skewness	Kurtosis
128	6.65118	0.574996	0.052879	−0.822637
256	6.64969	0.568117	0.105518	−0.862397
512	6.64967	0.561954	0.166596	−0.72365
1,024	6.64932	0.56002	0.212136	−0.658223
2,048	6.64954	0.559955	0.2174	−0.653254
4,096	6.64999	0.560162	0.192965	−0.648086
8,192	6.65021	0.560195	0.162296	−0.693831
16,384	6.65009	0.560443	0.187207	−0.636809
32,768	6.65016	0.560721	0.189291	−0.658717
65,536	6.64997	0.560595	0.190039	−0.663245
131,072	6.64995	0.560444	0.188385	−0.673174
262,144	6.64991	0.560693	0.191832	−0.676633
524,288	6.64988	0.560611	0.192973	−0.672354

Table 8 Moments of estimated probability distribution of penetration depth, determined via LHS applied to a coarse model, where all Johnson–Cook parameters and Poisson's ratios are taken as uncertain

Number of samples	Mean (cm)	Standard deviation (cm)	Skewness	Kurtosis
128	6.6701	0.557533	0.017908	−0.832316
256	6.66744	0.555064	0.056735	−0.741171
512	6.66678	0.561462	0.209637	−0.481117
1,024	6.66813	0.566678	0.238629	−0.457562
2,048	6.66784	0.564511	0.209327	−0.48841
4,096	6.66838	0.567589	0.184448	−0.574621
8,192	6.66883	0.567524	0.211078	−0.555243
16,384	6.66835	0.568057	0.210512	−0.564959
32,768	6.66838	0.568521	0.21297	−0.565021
65,536	6.66835	0.568256	0.212324	−0.545928
131,072	6.66817	0.567895	0.212661	−0.546767
262,144	6.66814	0.568206	0.21089	−0.555352
524,288	6.66817	0.568183	0.208243	−0.557963

Tables 9 through 11 show the moments calculated in the stochastic collocation runs. These moments are determined in Dakota by numerical integration, using the points of the sparse grid as quadrature points.[8] The number of samples (i.e., the number of grid points) is determined from the sparse grid level and the number of uncertain parameters. A comparison of Tables 6 and 9 shows that for the cases where the four most sensitive parameters are taken as uncertain, the results from LHS and stochastic collocation are largely consistent. However, as the number of uncertain parameters increases, the variation seen in the values of the moments also increases. This is starkly apparent in Fig. 8, where stochastic collocation appears particularly ill-behaved. Further ill behavior can be seen in the negative variance shown in Table 11. Since the standard deviation is the square root of the variance, such a negative value is obviously incorrect. Similar ill behavior from stochastic collocation has been observed by Tsuji et al.,[76] who attributed a calculated negative variance to the negative weights in numerical quadrature applied to a highly oscillatory computational model (in this case, scattering of high-frequency waves). Here, though, the computational model is dynamic penetration, which is not necessarily oscillatory per se. The model entails discontinuities in displacement fields, since it involves fracture, but this may not necessarily entail discontinuities or other non-smoothness in the QoI (i.e., the penetration depth). However, the coarse discretization employed by the computational model used here could introduce non-smoothness in the QoI, and Fig. 9 indicates that this is in fact the case. This figure shows the calculated values of the penetration depth as a function of the Johnson–Cook parameter C of the witness, at varying values of mesh element size, with the Johnson–Cook parameter n of the witness held constant at $n = 0.11$ and all other parameters are held constant at their baseline values. Coarsening the mesh does more than just lead to underestimation of the penetration depth; it also introduces artifacts that artificially roughen the solution for it. This would indicate that given a properly refined mesh, stochastic collocation would be more well-behaved.

Table 9 Moments of estimated probability distribution of penetration depth, determined via stochastic collocation applied to a coarse model, where the 4 most sensitive parameters are taken as uncertain

Sparse grid level	Number of samples	Mean (cm)	Standard deviation (cm)	Skewness	Kurtosis
1	9	6.6372	0.571718	−0.003583	−2.07818
2	49	6.62832	0.543583	0.214062	−0.752549
3	209	6.62883	0.552371	0.291884	−0.767651
4	705	6.63528	0.542335	0.14579	−0.731395
5	1,985	6.64836	0.62377	0.398902	−0.790707
6	4,865	6.62821	0.553691	0.225577	−0.775033
7	10,369	6.62333	0.536665	0.258248	−0.584716
8	19,841	6.63093	0.550355	0.231678	−0.748186
9	35,201	6.63016	0.551521	0.208562	−0.760429
10	57,729	6.62727	0.546689	0.232657	−0.734721
11	90,497	6.63014	0.549362	0.229378	−0.729081

Table 10 Moments of estimated probability distribution of penetration depth, determined via stochastic collocation applied to a coarse model, where the Johnson–Cook parameters and Poisson's ratio of the witness are taken as uncertain

Sparse grid level	Number of samples	Mean (cm)	Standard deviation (cm)	Skewness	Kurtosis
1	15	6.6229	0.579427	0.062287	−2.12148
2	127	6.75694	0.544227	−0.390457	−0.577686
3	799	6.60178	0.57202	0.520205	−0.571376
4	4,047	6.60138	0.533004	0.407697	−0.141323
5	17,263	6.70644	0.577721	−0.108599	−0.785255
6	63,967	6.61358	0.556794	0.404215	−0.712062

Table 11 Moments of estimated probability distribution of penetration depth, determined via stochastic collocation applied to a coarse model, where all Johnson–Cook parameters and Poisson's ratios are taken as uncertain. Red text is used to highlight an obviously faulty value.

Sparse grid level	Number of samples	Mean (cm)	Variance (cm^2)	Skewness	Kurtosis
1	29	6.83208	0.292281	−1.11537	−0.986829
2	449	6.55093	0.295054	0.65924	−0.712058
3	4,929	6.71718	0.451761	0.313725	−1.18332
4	42,785	5.84996	−0.777013	NA	NA

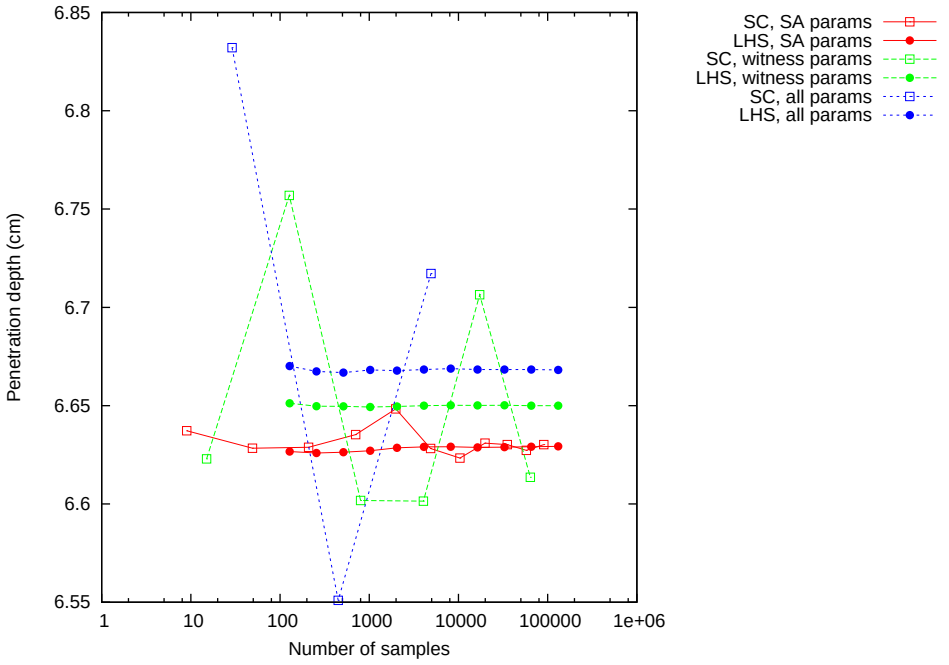

Fig. 8 Calculated values of the mean as a function of the number of samples, for both LHS and stochastic collocation applied to a coarse model

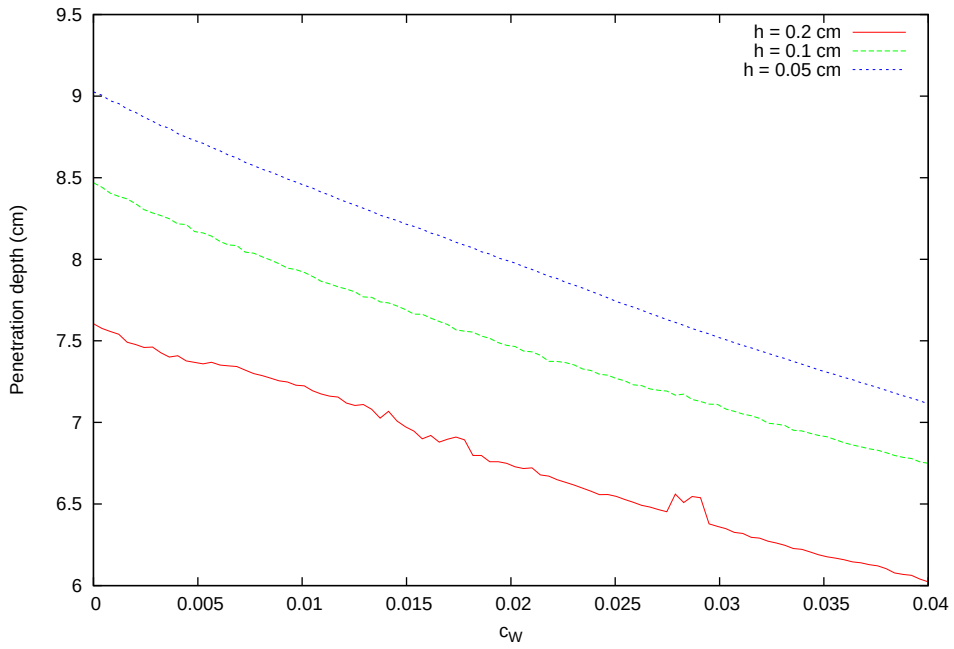

Fig. 9 Calculated penetration depth as a function of Johnson–Cook parameter C of the witness, at varying values of mesh element size h. Johnson–Cook parameter n of the witness is held constant at $n = 0.11$. All other parameters are held constant at their baseline values.

5.3 Uncertainty Propagation through Emulator of Refined Computational Model

The length of time needed to run a refined computational model of dynamic penetration, that is, one with an element size of at least 0.05 cm, makes certain computational analyses intractable. An interval analysis with EGO would be an example of this. Even if only the 4 most sensitive parameters are taken as uncertain, then an interval analysis will likely require about 25 samples, given the results shown in Table 5. Now the first few samples may be done in parallel. By default, in Dakota, there would be $(P + 1)(P + 2)/2$ of these samples, where P is the number of uncertain parameters, so for $P = 4$, the first 15 samples may be done in parallel, leaving the next 10 to be done in sequence. As indicated in Table 2, each of these samples takes about 17.5 h (on 64 processing cores), so the expected wall clock time would be about 8 days, which is longer than allowed wall clock time for a batch job on several computing clusters.[77–80] Another example of a difficult analysis would be stochastic collocation with 14 parameters. To determine convergence in practice, collocation would need to be done for 4 sparse grid levels, but at the fourth level, 42,785 samples are required, leading to a computational cost of about

48 million CPU hours for that grid level alone. Because of this, emulators of the refined computational model of dynamic penetration are employed.

The Surfpack functionality in Dakota has been used to create emulators for 3 sets of uncertain parameters: all the Johnson–Cook parameters and Poisson's ratios for both penetrator and witness, 14 parameters in total; the Johnson–Cook parameters and Poisson's ratio for the witness only, 7 in total; and the 4 most sensitive parameters as determined from the previously done sensitivity analysis (i.e., A, B, C, and n of the witness). As in the analyses described in Section 5.2, the bounds assumed for any uncertain parameters are shown in Table 4, while parameters not treated as uncertain are assigned the baseline values shown in Table 1. Kriging methods (with a reduced quadratic trend) are used to build the emulators. Each emulator is fit to 128 samples of the refined model (with element size 0.05 cm). The placement of samples is determined via LHS. Dakota is used to perform 2 forms of cross-validation on the emulators: k-fold (with 5 folds) and leave-one-out. For each cross-validation, the root mean square (RMS) error and mean absolute error are calculated, and the results of these tests are shown in Table 12.

Table 12 Cross-validation test results

	5-fold		Leave-one-out	
Number of parameters	**RMS error (cm)**	**Mean abs. error (cm)**	**RMS error (cm)**	**Mean abs. error (cm)**
4	0.0161882	0.0101982	0.0128969	0.0093863
7	0.0176734	0.0132631	0.0178954	0.0136456
14	0.0372402	0.0266779	0.0286012	0.0215368

While Dakota is capable of creating model emulators and saving them to files, at the time of this writing it is not capable of reading back in those saved emulators and using them for further analysis. Accordingly, we have modified our copy of Dakota to add this functionality, and this copy is what has been used in the analyses discussed below. The patches for this functionality have been submitted to the Dakota developers.

5.3.1 Interval Analysis

In order to compare the results from the emulator of the fine model with the previous results from the coarse model, 2 methods are initially used to estimate the minimum and maximum values of the penetration depth: EGO and LHS. Again,

Dakota's implementations of these methods, which are discussed in more detail in Section 5.2, have been used here. However, convergence problems have occurred in the EGO implementation in Dakota, especially for the emulators with over 4 parameters, where the implementation terminates because it reaches the maximum number of samplings of the computational model (which has a hard-coded cap of 89,980 within Dakota), instead of terminating due to finding an extremum. Because of this, EGO is replaced with an evolutionary algorithm (EA) in subsequent interval analyses. Unlike EGO, where the number of samplings is dependent on a convergence tolerance, the EA implementation in Dakota proceeds until either the number of iterations or the number of samplings of the computational model reaches a user-specified maximum value.[4] Hence, in subsequent figures, the extrema determined from EA are plotted as a function of the number of samplings of the computational model. Like EGO, EA forces a serialization of the sampling of the computation model. That is, at each iteration of EA, the determination of where to next sample the computational model depends upon the evaluations of the model that have been done in previous iterations.

Table 13 and Figs. 10 and 11 show the results of interval analyses that have been done using LHS and EA. For both LHS and EA, reducing the number of uncertain parameters leads to overestimating the minimum and underestimating the maximum values of the penetration depth. The figures also show that LHS, even with a sample size of over 500,000, consistently overestimates the minimum and underestimates the maximum. These results are similar to the ones in Section 5.2.1. However, most of these results have been obtained on a Linux workstation rather than a computational cluster, because they have been determined from computationally cheap emulators, so the overall computation time is usually on the order of minutes. The computational costs of EA are shown in Table 14, and it is apparent that the cost for EA here is not strongly dependent on the number of parameters.

Table 13 Interval analysis results using emulator of refined computational model

Method	Number of uncertain parameters	Number of samples to determine minimum	Number of samples to determine maximum	Minimum (cm)	Maximum (cm)
LHS	4	524,288	524,288	5.9974	9.99367
	7	524,288	524,288	5.8905	10.50262
	14	524,288	524,288	5.82182	10.40303
EA	4	6,000	6,000	5.93536	10.07932
	7	65,561	65,555	5.63299	10.76256
	14	524,373	524,326	5.33994	10.95350

Table 14 Computational costs of EA using emulator of refined computational model

Number of uncertain parameters	Number of samples to determine minimum	Number of samples to determine maximum	CPU time (s)
4	6,000	6,000	3617.11
7	65,561	65,555	35.34
14	524,373	524,326	248.72

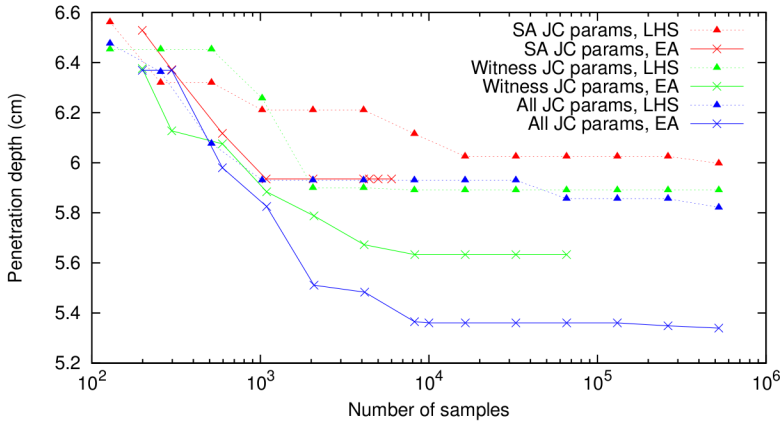

Fig. 10 Minimum penetration depth as a function of the number of samples used in LHS, using emulator of refined computational model

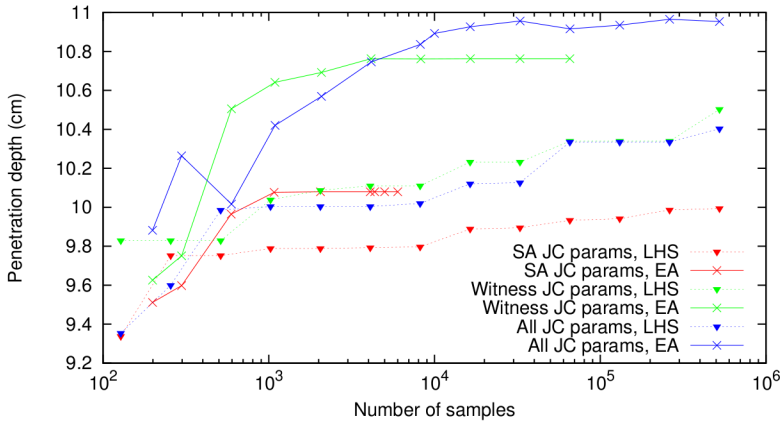

Fig. 11 Maximum penetration depth as a function of the number of samples used in LHS, using emulator of refined computational model

5.3.2 Aleatoric Uncertainty Propagation

In addition to the interval analysis, uncertainty propagation methods suited for aleatoric uncertainties are also applied to the dynamic penetration problem. The methods used to estimate the moments of the probability density function of the penetration depth, as well as the implementations of these methods, are the same ones that have been applied to the coarse computational model in Section 5.2.2: Monte Carlo using LHS and stochastic collocation using Smolyak sparse grids. Again, since probability distributions for the uncertain parameters are not known, the results of these analyses should be treated with caution. In order to apply the methods for aleatoric uncertainties, the probability density functions of the uncertain parameters are taken to be uniform, with the bounds as shown in Table 4.

Tables 15 through 17 show the mean, standard deviation, skewness, and kurtosis of the Latin hypercube samples that have been used in the interval analysis discussed in Section 5.3.1. Skewness and kurtosis have been calculated using the formulas shown in Eqs. 3 and 4. Again, the number of uncertain parameters changes the values to which the moments of the samples appear to have converged, and this effect is stronger for higher order moments. The mean varies from about 7.86 to 7.89 cm, a relative difference of about 0.4%. The standard deviation varies from about 0.66 to 0.68 cm, a relative difference of about 3%. For the kurtosis in particular, if only the 4 most sensitive parameters are taken as uncertain, it is approximately -0.49, but if all 14 of the parameters in Table 1 are taken as uncertain, it becomes closer to -0.44, a relative difference of 11%.

Tables 18 through 20 show the moments calculated in the stochastic collocation runs. These results are consistent with the LHS results from Tables 15 through 17, and they show none of the ill behavior that is seen in the stochastic collocation results for the coarse model. Furthermore, unlike the case with the coarse model, these stochastic collocation results converge more quickly than the corresponding LHS results. Figure 12 illustrates this by plotting the estimated relative error in the mean versus the number of samples.

These results, much like the results from Section 5.3.1, usually take only minutes because they have been determined from computationally cheap emulators. The computational costs of stochastic collocation in particular are shown in Table 21.

Table 15 Moments of estimated probability distribution of penetration depth, determined via LHS applied to an emulator of a refined model, where the 4 most sensitive parameters are taken as uncertain

Number of samples	Mean (cm)	Standard deviation (cm)	Skewness	Kurtosis
128	7.860942	0.671751	0.214417	−0.780524
256	7.861443	0.666484	0.305348	−0.506901
512	7.861390	0.657371	0.191849	−0.504726
1,024	7.861731	0.657898	0.190248	−0.469401
2,048	7.862261	0.658083	0.202693	−0.518278
4,096	7.862414	0.659850	0.243553	−0.506483
8,192	7.862314	0.658545	0.235649	−0.490406
16,384	7.862266	0.657494	0.224716	−0.489922
32,768	7.862431	0.657890	0.226145	−0.489605
65,536	7.862375	0.657438	0.224686	−0.495513
131,072	7.862323	0.657298	0.219203	−0.497210
262,144	7.862369	0.657512	0.216640	−0.492003
524,288	7.862408	0.657653	0.219913	−0.490080

Table 16 Moments of estimated probability distribution of penetration depth, determined via LHS applied to an emulator of a refined model, where the Johnson–Cook parameters and Poisson's ratio of the witness are taken as uncertain

Number of samples	Mean (cm)	Standard deviation (cm)	Skewness	Kurtosis
128	7.878764	0.693531	0.093858	−0.572949
256	7.877735	0.685709	0.143252	−0.644030
512	7.878390	0.680854	0.215643	−0.464405
1,024	7.878854	0.679304	0.259640	−0.376152
2,048	7.878791	0.678602	0.269414	−0.365374
4,096	7.878548	0.678936	0.242118	−0.363966
8,192	7.878413	0.678761	0.204815	−0.423998
16,384	7.878558	0.679683	0.235691	−0.350584
32,768	7.878527	0.679589	0.238237	−0.377591
65,536	7.878521	0.679489	0.239495	−0.381770
131,072	7.878470	0.679332	0.236238	−0.397288
262,144	7.878483	0.679660	0.239759	−0.401954
524,288	7.878492	0.679633	0.241657	−0.396144

Table 17 Moments of estimated probability distribution of penetration depth, determined via LHS applied to an emulator of a refined model, where all Johnson–Cook parameters and Poisson's ratios are taken as uncertain

Number of samples	Mean (cm)	Standard deviation (cm)	Skewness	Kurtosis
128	7.887400	0.670496	0.078593	−0.628344
256	7.886947	0.668399	0.086020	−0.553926
512	7.888593	0.674503	0.219055	−0.361933
1,024	7.889483	0.679516	0.225833	−0.366981
2,048	7.889293	0.677243	0.185826	−0.378712
4,096	7.889428	0.680186	0.165514	−0.464165
8,192	7.889521	0.679740	0.205251	−0.440576
16,384	7.889599	0.680939	0.208647	−0.451748
32,768	7.889653	0.681484	0.210668	−0.452663
65,536	7.889630	0.681646	0.211492	−0.429366
131,072	7.889645	0.681223	0.212860	−0.429911
262,144	7.889667	0.681586	0.211442	−0.438300
524,288	7.889664	0.681610	0.208760	−0.439043

Table 18 Moments of estimated probability distribution of penetration depth, determined via stochastic collocation applied to an emulator of a refined model, where the 4 most sensitive parameters are taken as uncertain

Sparse grid level	Number of samples	Mean (cm)	Standard deviation (cm)	Skewness	Kurtosis
1	9	7.857600	0.645780	0.027160	−2.058740
2	49	7.864780	0.657830	0.222844	−0.589014
3	209	7.862480	0.657247	0.222194	−0.461234
4	705	7.862320	0.657826	0.219210	−0.497957
5	1,985	7.862380	0.657763	0.220081	−0.493212
6	4,865	7.862370	0.657759	0.220166	−0.493194
7	10,369	7.862370	0.657759	0.220171	−0.493193

Table 19 Moments of estimated probability distribution of penetration depth, determined via stochastic collocation applied to an emulator of a refined model, where the Johnson–Cook parameters and Poisson's ratio of the witness are taken as uncertain

Sparse grid level	Number of samples	Mean (cm)	Standard deviation (cm)	Skewness	Kurtosis
1	15	7.873470	0.665098	−0.027559	−2.116560
2	127	7.879140	0.681261	0.229545	−0.566297
3	799	7.878530	0.679870	0.242216	−0.389906
4	4,047	7.878540	0.679926	0.240589	−0.406856
5	17,263	7.878540	0.679927	0.240684	−0.405990
6	63,967	7.878540	0.679927	0.240676	−0.406026

Table 20 Moments of estimated probability distribution of penetration depth, determined via stochastic collocation applied to an emulator of a refined model, where all Johnson–Cook parameters and Poisson's ratios are taken as uncertain

Sparse grid level	Number of samples	Mean (cm)	Standard deviation (cm)	Skewness	Kurtosis
1	29	7.892810	0.660604	−0.119872	−2.124830
2	449	7.889430	0.684504	0.217672	−0.636388
3	4,929	7.889680	0.681388	0.207447	−0.415551
4	42,785	7.889670	0.681634	0.206738	−0.441398
5	310,689	7.889670	0.681621	0.206899	−0.439740

Table 21 Computational costs of stochastic collocation at the highest sparse grid level used, given emulator of refined computational model

Number of uncertain parameters	Sparse grid leven	Number of samples	CPU time (s)
4	7	10,369	10.26
7	6	63,967	121.83
14	5	310,689	2130.9

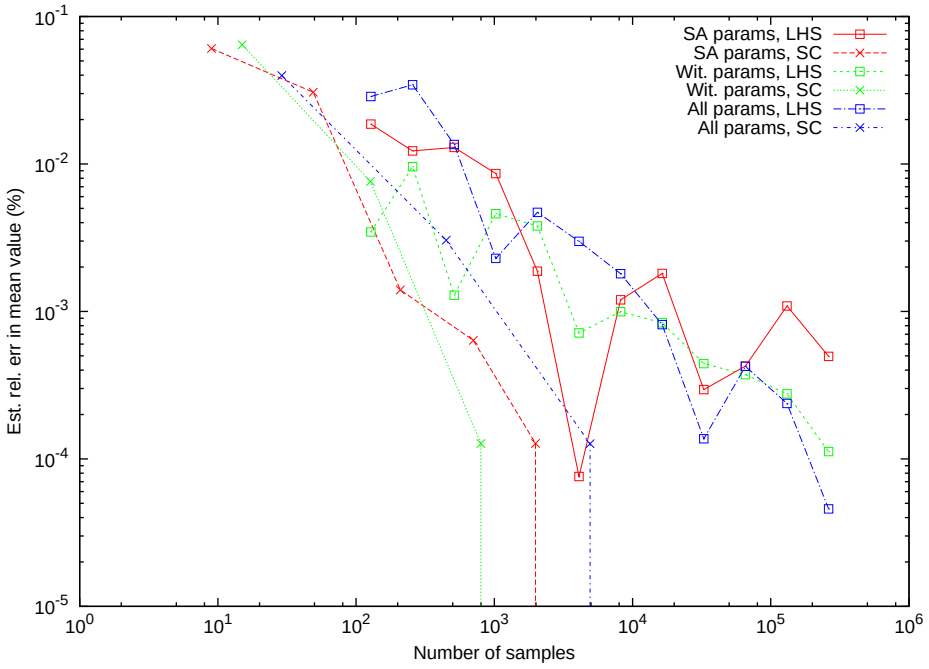

Fig. 12 Estimated relative error in mean value as a function of the number of samples, for both LHS and stochastic collocation applied to an emulator of a refined model

6. Discussion and Conclusions

The survey of software discussed in Section 4 indicates the range of UQ software that is available. Some of these works of software, such as Chaospy or QUESO, have a relatively narrow focus, such as polynomial chaos or MCMC, while others, like Dakota or PSUADE, are intended to offer a broad range of functionality. Some of these, such as UQ-PyL, appear to be maintained by one or two developers, while others, such as Dakota, are supported by a team of developers at an institution, such as a national laboratory. So far, Dakota appears to be the most promising UQ software out of what is currently available, at least with regard to uncertainty propagation. However, even it was not entirely sufficient for the analyses described in Section 5. It lacked a method of SA (that was supplied by PSUADE), and it required modification to read back in the very saved model emulators that it created.

In the uncertainty propagation analyses of dynamic penetration of armor discussed in Section 5, the intervals from the interval analyses are far from sharp. For the coarse model, the width of the interval is about 64% of the estimated mean penetration depth, and for the emulator of the refined model, the width is about 71% of the mean. This lack of sharpness is also reflected in the results from the propagation of aleatoric uncertainty through the dynamic penetration simulations. Based on Chebyshev's inequality,[81] the "true" value of the penetration depth has at least an 88.8% probability of being within 3 standard deviations from the mean.* If one were to assume the coarse computational model is correct, this would indicate that the true value is likely within an interval of about $[4.96, 8.37]$ cm. For the fine model, this would indicate that it is likely within an interval of about $[5.84, 9.93]$ cm. In principle, this lack of sharpness may be improved by having better estimates of the uncertainties of the parameters. However, the uncertainty of at least one of the parameters, the Johnson–Cook parameter C, is a reflection of how poorly the Johnson–Cook strength model reflects how the stress-strain relationship in a metal is affected by the stress rate,[73] and Fig. 9 indicates that variation in this parameter is responsible for a significant portion of the variability in the penetration depth. Reducing the uncertainty due to this issue may be a matter of replacing the Johnson–Cook strength model with a more accurate one.

*For a *Gaussian* probability distribution, the true value would have about a 99.7% probability of being within 3 standard deviations from the mean.[81] However, since the probability distributions of the model parameters are crudely estimated as uniform, the shape of the probability distribution of the penetration depth is also a crude estimate and likely should not be treated as close to Gaussian.

One can also see the so-called curse of dimensionality have some effect on the uncertainty propagation analyses of dynamic penetration of armor. In the interval analyses using EGO and EA, the number of samples needed to reach a converged value for the extrema increases with the number of uncertain parameters. As pointed out in Section 2.2, stochastic collocation methods are also affected by the curse, and how this manifests itself in practice can be seen in how, for a given sparse level, the number of samples needed increases with the number of parameters. For 4 parameters, 705 samples are needed; for 7 parameters, 4,047 samples are needed, and for all 14 parameters, 42,785 samples are needed. However, the practical effect of this is mitigated through the use of emulators. For example, given a sparse grid level of 4 and the coarse model using 14 parameters, the number of CPU hours for stochastic collocation is about $42,785 \times 64 \times 6$ min $\approx 270,000$ h. As seen in Table 21, for an even higher sparse grid level and the same number of parameters, the CPU time for stochastic collocation with an emulator is about 36 min. In short, because the emulators are so computationally inexpensive, even if the relative computational cost explodes as the number of parameters increases, the absolute cost for a large number of parameters can still be manageable. As pointed out in Section 2.4.2, though, the construction of the emulators themselves can be subject to the curse of dimensionality, which may limit the degree to which they can mitigate the practical costs of the curse.

Future work on UQ with armor may expand the number of uncertain model parameters, use more precise characterizations of the uncertainties in those parameters, and possibly use more accurate strength models.

7. References

1. National Academies of Sciences, Engineering, and Medicine. 2015–2016 Assessment of the Army Research Laboratory. Washington (DC): The National Academies Press; 2017.

2. Smith RC. Uncertainty quantification: theory, implementation, and applications. Philadelphia (PA): Society for Industrial and Applied Mathematics; 2013.

3. Oberkampf WL, Helton JC. Evidence theory for engineering applications. In: Engineering Design Reliability Handbook; Nikolaidis E, Ghiocel DM, Singhal S, editors. Boca Raton (FL): CRC Press; 2004 Dec; Chapter 10.

4. Adams BM et al. Dakota, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: version 6.5 user's manual. Albuquerque (NM): Sandia National Laboratories; 2016 Nov.

5. O'Hagan A, Oakley JE. Probability is perfect, but we can't elicit it perfectly. Reliability Engineering & System Safety. 2004;85(1):239–248.

6. Roy CJ, Oberkampf WL. A comprehensive framework for verification, validation, and uncertainty quantification in scientific computing. Computer Methods in Applied Mechanics and Engineering. 2011;200(25–28):2131–2144.

7. Chaospy documentation. 2016 [accessed 2017 Feb 1]. `http://chaospy.readthedocs.io/en/master/index.html`.

8. Adams BM et al. Dakota, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: version 6.5 theory manual. Albuquerque (NM): Sandia National Laboratories; 2016 Nov.

9. McKay MD, Beckman RJ, Conover WJ. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. Technometrics. 1979;21(2):239–245.

10. Denny M. Introduction to importance sampling in rare-event simulations. European Journal of Physics. 2001;22(4):403.

11. Eldred M. Recent advances in non-intrusive polynomial chaos and stochastic collocation methods for uncertainty analysis and design. In: 50th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference; 2009 May; Palm Springs (CA). (AIAA; no. 2009–2274) American Institute of Aeronautics and Astronautics; c2009.

12. Crespo LG, Giesy DP, Kenny SP. Reliability-based analysis and design via failure domain bounding. Structural Safety. 2009;31(4):306–315.

13. Adams BM et al. Dakota, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: version 6.5 reference manual. Albuquerque (NM): Sandia National Laboratories; 2016 Nov.

14. Jones DR. Direct global optimization algorithm. In: Encyclopedia of Optimization; Floudas CA, Pardalos PM, editors. Boston (MA): Springer US; 2009. p. 725–735.

15. Suh D, Yook J. A method to determine basic probability assignment in context awareness of a moving object. International Journal of Distributed Sensor Networks. 2013;9(8).

16. Xu P, Deng Y, Su X, Mahadevan S. A new method to determine basic probability assignment from training data. Knowledge-Based Systems. 2013;46:69–80.

17. Kennedy MC, O'Hagan A. Bayesian calibration of computer models. Journal of the Royal Statistical Society: Series B (Statistical Methodology). 2001;63(3):425–464.

18. Brooks S, Gelman A, Jones G, Meng XL, editors. Handbook of Markov Chain Monte Carlo. Boca Raton (FL): CRC Press; 2011.

19. Helton JC, Johnson JD, Sallaberry CJ, Storlie CB. Survey of sampling-based methods for uncertainty and sensitivity analysis. Reliability Engineering & System Safety. 2006;91(10–11):1175–1209.

20. Saltelli A, Chan K, Scott EM, editors. Sensitivity analysis. Chichester (England): John Wiley & Sons, Ltd.; 2000.

21. Gan Y, Duan Q, Gong W, Tong C, Sun Y, Chu W, Ye A, Miao C, Di Z. A comprehensive evaluation of various sensitivity analysis methods: a case study with a hydrological model. Environmental Modelling & Software. 2014;51:269–285.

22. Tong C. PSUADE Short Manual (Version 1.7). Livermore (CA): Lawrence Livermore National Laboratory; 2017 May.

23. Tripathy R, Bilionis I, Gonzalez M. Gaussian processes with built-in dimensionality reduction: applications to high-dimensional uncertainty propagation. Journal of Computational Physics. 2016;321:191–223.

24. Chouchoulas A, Shen Q. Rough Set-Based Dimensionality Reduction for Multivariate Adaptive Regression Splines. In: Rough Sets and Current Trends in Computing: Second International Conference, RSCTC 2000 Banff, Canada, October 16–19, 2000 Revised Papers; Ziarko W, Yao Y, editors. Berlin (Heidelberg): Springer Berlin Heidelberg; 2001. p. 144–151.

25. Diwekar U. Optimization Under Uncertainty. In: Introduction to Applied Optimization; 2nd ed.; New York (NY): Springer; 2008 ; Chapter 5.

26. Giunta AA, Eldred MS, Swiler LP, Trucano TG. Perspectives on optimization under uncertainty: algorithms and applications. In: 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference; 2004 Aug; Albany (NY). (AIAA; no. 2004–4451) American Institute of Aeronautics and Astronautics; c2004.

27. High Performance Computing Centers @ DoD HPCMP. 2017 [accessed 2017 Jan 20]. https://centers.hpc.mil/.

28. OpenTURNS. 2016 [accessed 2016 Dec 22]. http://www.openturns.org/.

29. Hunt M. PUQ: PRISM uncertainty quantification framework. c2010–2012 [accessed 2016 Dec 22]. http://c-primed.github.io/puq/.

30. Login vs. Service vs. Compute Nodes. 2013 [accessed 2017 Jan 18]. https://www.olcf.ornl.gov/kb_articles/login-vs-service-vs-compute-nodes/.

31. Stephens JA. Sandia National Laboratories, Albuquerque, NM. Personal communication, 2016 Dec 2.

32. Dakota GUI Version 0.2 User Manual. 2016 [accessed 2017 Jan 23]. https://dakota.sandia.gov/content/dakota-gui-version-02-user-manual.

33. Campolongo F, Cariboni J, Saltelli A. An effective screening design for sensitivity analysis of large models. Environmental Modelling & Software. 2007;22(10):1509–1518.

34. Morris MD. Factorial sampling plans for preliminary computational experiments. Technometrics. 1991;33(2):161–174.

35. Foundation FS. Frequently Asked Questions about the GNU Licenses. 2017 [accessed 2017 Jan 24]. https://www.gnu.org/licenses/gpl-faq.html.

36. Foundation FS. GNU Lesser General Public License, version 2.1. 1999 [accessed 2017 Jan 24]. https://www.gnu.org/licenses/old-licenses/lgpl-2.1.html.

37. Cukier RI, Fortuin CM, Shuler KE, Petschek AG, Schaibly JH. Study of the sensitivity of coupled reaction systems to uncertainties in rate coefficients. I Theory. The Journal of Chemical Physics. 1973;59(8):3873–3878.

38. Pi H, Peterson C. Finding the embedding dimension and variable dependencies in time series. Neural Computation. 1994;6(3):509–520.

39. Breiman L, Friedman J, Stone CJ, Olshen RA. Classification and regression trees. Boca Raton (FL): CRC Press; 1984.

40. Baudin M, Dutfoy A, Iooss B, Popelin AL. OpenTURNS: An Industrial Software for Uncertainty Quantification in Simulation. In: Handbook of Uncertainty Quantification; Ghanem R, Higdon D, Owhadi H, editors. Cham (Switzerland): Springer International Publishing; 2016. p. 1–38.

41. Feinberg J, Langtangen HP. Chaospy: An open source tool for designing methods of uncertainty quantification. Journal of Computational Science. 2015;11:46–57.

42. Chaospy 2.2.0: Python Package Index. 2017 [accessed 2017 Feb 1]. https://pypi.python.org/pypi/chaospy.

43. The MIT License. 2017 [accessed 2017 Jan 25]. https://opensource.org/licenses/MIT.

44. Estacio-Hiroms KC, Prudencio EE. QUESO User's Manual. Austin (TX): Center for Predictive Engineering and Computational Sciences, University of Texas; 2013.

45. Prudencio EE, Schulz KW. The parallel C++ statistical library 'QUESO': Quantification of uncertainty for estimation, simulation and optimization. In: Euro-Par 2011: Parallel Processing Workshops; 2011 Aug; Bordeaux (France). Springer; c2012. p. 398–407.

46. Foundation FS. Why you shouldn't use the Lesser GPL for your next library. 2016 [accessed 2017 Jan 24]. https://www.gnu.org/licenses/why-not-lgpl.html.

47. Question about QUESO's formulation of forward problems. 2017 [accessed 2017 Jan 30]. https://groups.google.com/d/msg/queso-users/ptkxrqG2JHk/rBMiEF8tBwAJ.

48. Patil A, Huard D, Fonnesbeck C. PyMC: Bayesian stochastic modelling in Python. Journal of Statistical Software. 2010;35(1):1–81.

49. Salvatier J, Wiecki TV, Fonnesbeck C. Probabilistic programming in Python using PyMC3. PeerJ Computer Science. 2016;2:e55.

50. PyMC User's Guide. 2017 [accessed 2017 Feb 2]. http://pymcmc.readthedocs.io/en/latest/.

51. Stan. 2017 [accessed 2017 Apr 5]. http://mc-stan.org/.

52. The 3-Clause BSD License. 2017 [accessed 2017 Feb 3]. https://opensource.org/licenses/BSD-3-Clause.

53. Betancourt M. Re: Using external C++ code in Stan. 2017 [accessed 2017 Jul 14]. https://groups.google.com/d/msg/stan-users/vs3A1uVmr1g/oA--zKaFCwAJ.

54. Kenny SP, Crespo LG, Giesy DP. UQTools: The Uncertainty Quantification Toolbox—Introduction and Tutorial. Hampton (VA): National Aeronautics and Space Administration (NASA); 2012 Apr.

55. The Uncertainty Quantification Toolbox. 2016 [accessed 2017 Feb 3]. https://uqtools.larc.nasa.gov/.

56. Crespo LG, Kenny SP, Giesy DP. Sampling-based strategies for the estimation of probabilistic sensitivities. In: 50th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference; 2009 May; Palm Springs (CA). (AIAA; no. 2009–2283) American Institute of Aeronautics and Astronautics; c2009.

57. Wang C, Duan Q, Tong CH, Di Z, Gong W. A GUI platform for uncertainty quantification of complex dynamical models. Environmental Modelling & Software. 2016;76:1–12.

58. Wang C, Duan Q. UQ-PyL User Manual (Version 1.1). Beijing (China): Beijing Normal University; 2015.

59. Foundation FS. GNU General Public License, version 3. 2007 [accessed 2017 Feb 3]. https://www.gnu.org/licenses/gpl.html.

60. Burges CJ. A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge Discovery. 1998;2(2):121–167.

61. Madsen H, Thyregod P. Introduction to general and generalized linear models. Boca Raton (FL): CRC Press; 2010.

62. SmartUQ – Quantify Every Uncertainty. 2017 [accessed 2017 Jan 31]. https://www.smartuq.com/.

63. SmartUQ's Uncertainty Quantification Software Features. 2017 [accessed 2017 Feb 1]. https://www.youtube.com/watch?v=eQ8u58D-MR0.

64. COSSAN software. 2017 [accessed 2017 Feb 8]. http://www.cossan.co.uk/.

65. Patelli E. An open computational framework for reliability based optimization: presentation slides from Eleventh International Conference on

Computational Structures Technology, Dubrovnik (Croatia). 2012 [accessed 2017 Feb 8]. https://www.academia.edu/3151038/An_open_computational_framework_for_reliability_based_optimization.

66. MUQ: MIT Uncertainty Quantification Library. 2015 [accessed 2016 Dec 22]. http://muq.mit.edu/.

67. Parno M, Davis A, Wang Z, Feng C, Marzouk Y, Conrad P. MUQ: The MIT Uncertainty Quantification Library Version 0.1 User's Manual. Cambridge (MA): Massachusetts Institute of Technology; 2014 Dec.

68. van Heesch D. Doxygen. 2016 [accessed 2017 Aug 1]. http://www.doxygen.org.

69. Sargsyan K, Safta C, Chowdhary K, Castorena S, de Bord S, Debusschere B. UQTk version 3.0.1 user manual. Sandia National Laboratories; 2016 Sep. Report No.: SAND2016-9215.

70. Hertel ES, Jr., Bell RL, Elrick MG, Farnsworth AV, Kerley GI, Mcglaun JM, Petney SV, Silling SA, Taylor PA, Yarrington L. CTH: A software family for multi-dimensional shock physics analysis. In: Shock Waves @ Marseille I: Hypersonics, Shock Tube & Shock Tunnel Flow; 1993 June; Marseille (France). Springer; c1995. p. 377–382.

71. Hornbaker DJ. Quantifying uncertainty from computational factors in simulations of a model ballistic system. Aberdeen Proving Ground (MD): Army Research Laboratory (US); 2017 Aug. Report No.: ARL–TR–8074.

72. Johnson GR, Cook WH. A constitutive model and data for metals subjected to large strains, high strain rates and high temperatures. In: Seventh International Symposium on Ballistics: Proceedings; 1983 Apr; The Hague (Netherlands). American Defense Preparedness Association; c1983. p. 541–547.

73. Becker R. US Army Research Laboratory, Aberdeen Proving Ground, MD. Personal communication, 2017 Jan 9.

74. Jones DR, Schonlau M, Welch WJ. Efficient global optimization of expensive black-box functions. Journal of Global Optimization. 1998;13(4):455–492.

75. Joanes DN, Gill CA. Comparing measures of sample skewness and kurtosis. Journal of the Royal Statistical Society: Series D (The Statistician). 1998;47(1):183–189.

76. Xiu D, Tsuji P, Ying L. Fast method for high-frequency acoustic scattering from random scatterers. International Journal for Uncertainty Quantification. 2011;1(2):99–117.

77. Cray XC40 (Conrad) User Guide. 2017 [accessed 2017 Jun 30]. https://www.navydsrc.hpc.mil/docs/conradUserGuide.html#queueInfo.

78. SGI ICE X (Topaz) User Guide. 2017 [accessed 2017 Jun 30]. https://www.erdc.hpc.mil/docs/topazUserGuide.html#queueInfo.

79. Cray XC40 (Excalibur) User Guide. 2017 [accessed 2017 Jun 30]. https://www.arl.hpc.mil/docs/excaliburUserGuide.html#queueInfo.

80. SGI Ice X (Thunder) User Guide. 2017 [accessed 2017 Jun 30]. https://www.afrl.hpc.mil/docs/thunderUserGuide.html#queueInfo.

81. NIST/SEMATECH e-Handbook of Statistical Methods: Approximate intervals that contain most of the population values. 2017 [accessed 2017 Aug 9]. http://www.itl.nist.gov/div898/handbook/prc/section2/prc261.htm.

List of Symbols, Abbreviations, and Acronyms

ARL US Army Research Laboratory

BPA basic probability assignment

EA evolutionary algorithm

EGO efficient global optimization

FAST Fourier amplitude sampling test

GPL (GNU) General Public License

HPC high-performance computing

LGPL (GNU) Lesser General Public License

LHS Latin hypercube sampling

MARS multivariate adaptive regression splines

MCMC Markov chain Monte Carlo

MOAT Morris one-at-a-time

NASA National Aeronautics and Space Administration

OLS ordinary least-squares

OOU optimization under uncertainty

PBS Portable Batch System

QoI quantities of interest

RHA rolled homogeneous armor

RMS root mean square

SA sensitivity analysis

UQ uncertainty quantification

INTENTIONALLY LEFT BLANK.

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
2 (PDF)	DIR ARL IMAL HRA RECORDS MGMT RDRL DCL TECH LIB
1 (PDF)	GOVT PRINTG OFC A MALHOTRA
1 (PDF)	DIR USARL RDRL CIH C J J RAMSEY

INTENTIONALLY LEFT BLANK.