ARL-TN-0950 • MAY 2019

DEVCOM ARMY RESEARCH LABORATORY

Application of Existing Uncertainty Quantification Capabilities to Modeling of Dynamic Penetration of Armor: A Follow-Up

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-TN-0950 • MAY 2019

Application of Existing Uncertainty Quantification Capabilities to Modeling of Dynamic Penetration of Armor: A Follow-Up

by James J Ramsey
Computational and Information Sciences Directorate,
CCDC Army Research Laboratory

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)**
May 2019	Technical Note	October 2018–September 2019

4. TITLE AND SUBTITLE
Application of Existing Uncertainty Quantification Capabilities to Modeling of Dynamic Penetration of Armor: A Follow-Up

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)
CCDC Army Research Laboratory
ATTN: FCDD-RLC-NB
Aberdeen Proving Ground, MD 21005-5066

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TN-0950

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 is a continuation of previous work discussed in the technical report *Survey of Existing Uncertainty Quantification Capabilities for Army-Relevant Problems* (ARL-TR-8218), in particular, the application of existing software tools for uncertainty quantification to modeling of penetration of rolled homogeneous armor. Whereas this previous work used only one plasticity model for the target and crude estimates for the material model's parameters, the current work uses two plasticity models, Johnson-Cook and Zerilli-Armstrong, and estimates the uncertainties in the parameters of these models by applying approximate interval predictor models to experimental stress-strain data of the target material. In addition, three software tools are applied to the penetration problem: Dakota, OpenTURNS, and UQPy. The two material models lead to largely overlapping uncertainty intervals for the penetration depth, and these intervals are less than half as wide as those of the previous work. The three software tools largely agree in their output but differ in their computational expense and ease of use.

15. SUBJECT TERMS
uncertainty quantification, armor, Johnson-Cook, Zerilli-Armstrong, interval predictor model

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	39	**19b. TELEPHONE NUMBER (Include area code)** 410-278-5614

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

List of Tables

1. Introduction

Previous work[1] has examined the application of existing uncertainty quantification (UQ) methodologies to the Army-relevant problem of dynamic penetration of armor. In particular, a modified version of Dakota[2] has been used to propagate input uncertainties through a computational simulation of penetration into a semi-infinite target of rolled homogeneous armor (RHA) in order to obtain estimates of the uncertainties in the penetration depth. However, this work has uncovered two issues. First, because of the lack of availability at the time of input uncertainties in the material parameters of RHA, the previous work has resorted to crude approximations of these uncertainties, which has led to a possibly unrealistic range of likely values for the penetration depth. Second, the material model used for RHA is itself an approximation, introducing uncertainty that can only be approximately taken into account by varying model parameters.[3] Accordingly, a new study has been done to address these issues. In addition to this, new software tools, OpenTURNS[4] and UQPy,[5] have been applied alongside Dakota, in order to assess the capabilities of these tools.

2. Description of Armor Penetration Problem

The armor penetration problem used in both this work and the previous work of Ramsey[1] is shown in Fig. 1. As before, the penetrator is a cylindrical 90%W–7%Fe–3%Ni tungsten heavy alloy (WHA) 131W tapered rod, with a diameter of 0.91 cm, a length of 13.1 cm, and an initial velocity of 1.28 km/s. The target is semi-infinite RHA.

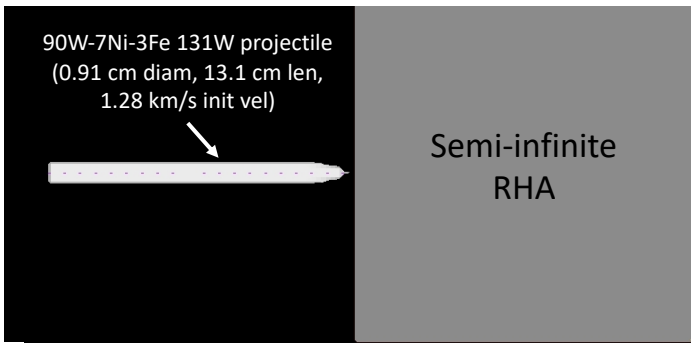

Fig. 1 Diagram of example dynamic armor penetration problem used in both the current and previous[1] work. The penetrator is a cylindrical 90%W–7%Fe–3%Ni tungsten alloy 131W rod, tapered at one end, with dimensions of 0.91 cm in diameter and 13.1 cm in length. The target is RHA, whose depth is taken to be effectively semi-infinite.

The Eulerian finite element code CTH[6] simulates the penetration. Each finite element is cubic with a side length of 0.05 cm. Each simulation has been run on 32 nodes of a computing cluster with 40 2.2-GHz Intel Xeon E5-2698v4 Broadwell cores per node. The median simulation time is 1.23 h, and the maximum simulation time is 3.78 h.

The Mie-Grüeneisen equation of state (EOS) is used for both penetrator and target. In the formulation of this EOS used in CTH, the relationship between the shock front velocity u_s and the velocity u_p of a point in the material is estimated to be

$$u_s = C_s + S_1 u_p + \frac{S_2}{C_S} u_p^2 \tag{1}$$

where C_S is the speed of sound in the material, and S_1 and S_2 are fitting parameters. However, here S_2 is taken to be zero, the specific heat capacity at constant volume C_v is taken to be constant with respect to temperature, and the initial pressure is taken to be negligible, so the EOS reduces to the following relationship between pressure P, density ρ, and temperature T[7]:

$$P = \frac{\rho_0 C_S^2 \chi}{(1 - \chi S_1)^2} \left(1 - \frac{\Gamma_0}{2}\chi\right) + \Gamma_0 \rho_0 C_v (T - T_0), \quad \chi = 1 - \frac{\rho_0}{\rho} \tag{2}$$

Here, ρ_0 and T_0 are the initial density and temperature, and Γ_0 is the Grüeneisen parameter. The values of the parameters of this EOS for the penetrator and target materials are taken from Hornbaker[7] and shown in Table 1.

Table 1 Parameters left fixed in CTH simulations

Parameter	WHA	RHA
ρ_0 (g/cm^3)	19.235	7.850
T_0 (K)	298	298
C_v (J/g · K)	0.138	0.446
Γ_0	1.72	1.67
C_S (m/s)	3980	4529
S_1	1.24	1.49
S_2 (m/s)	0	0
T_{melt} (K)	1723	1783
ν	0.310	0.294

The plasticity model used in previous work for both penetrator and target is the

Johnson-Cook model,[8] which takes the von Mises flow stress to be

$$\sigma_{\mathrm{JC}}(\epsilon_p, \dot{\epsilon}_p, T; \boldsymbol{\theta}_{\mathrm{JC}}) = \left(A + B\epsilon_p^n\right)\left(1 + C \ln \frac{\dot{\epsilon}_p}{\dot{\epsilon}_{p0}}\right)\left[1 - \left(\frac{T - T_{\mathrm{ref}}}{T_{\mathrm{melt}} - T_{\mathrm{ref}}}\right)^m\right] \qquad (3)$$

where ϵ_p and $\dot{\epsilon}_p$ are the equivalent plastic strain and plastic strain rate, $\boldsymbol{\theta}_{\mathrm{JC}} \equiv (A, B, n, C, m)$, and A, B, n, C, and m are fitting parameters. The fixed parameters in this model (i.e., *not* adjustable parameters fit to stress-strain data) are the nominal melting temperature of the material, T_{melt} (shown in Table 1), the reference temperature, $T_{\mathrm{ref}} = 298$ K, and the reference plastic strain rate $\dot{\epsilon}_{p0} = 1\ \mathrm{s}^{-1}$.

In order to examine the sensitivity of the simulation results on the plasticity model, the current work uses an alternative plasticity model for the target material, the Zerilli-Armstrong model for body-centered cubic (BCC) materials, in addition to the Johnson-Cook model. This model takes the von Mises flow stress to be

$$\sigma_{\mathrm{ZA,BCC}}(\epsilon_p, \dot{\epsilon}_p, T; \boldsymbol{\theta}_{\mathrm{ZA,BCC}}) = C_0 + C_1 \exp\left[\left(-C_3 + C_4 \ln \frac{\dot{\epsilon}_p}{\dot{\epsilon}_{p0}}\right) T\right] + C_5 \epsilon_p^n \qquad (4)$$

where $\boldsymbol{\theta}_{\mathrm{ZA,BCC}} \equiv (C_0, C_1, C_3, C_4, C_5, n)$, and C_0, C_1, C_3, C_4, C_5, and n are fitting parameters. (There is no parameter C_2; such a parameter belongs to the face-centered cubic version of the Zerilli-Armstrong model.[9]) In CTH, the Poisson's ratio ν of the material is considered part of the plasticity model. The values of this for the penetrator and target materials are in Table 1.

3. Reassessment of Parameter Uncertainties

Unfortunately, stress-strain data at high strain rates and temperatures does not appear to be available for the WHA used for the penetrator in the previous work, so one cannot refit a parameterized model to the data so as to obtain uncertainty measures on the fitted parameters. However, previous work[1] has indicated that the penetrations are largely insensitive to the material parameters of the penetrator, so the lower and upper bounds of those parameters from that previous work are reused and shown in Table 2. The bounds of A and B are roughly estimated to be ±10% of their nominal values shown in Table 2; the bounds of m are estimated to be ±15% of its nominal value; and the bounds of the other two parameters are based on an expert's opinion.[10]

Table 2 Lower and upper bounds of Johnson-Cook parameters for 90%W–7%Fe–3%Ni WHA, along with their nominal values

Parameter	Lower bound	Upper bound	Nominal value
A (MPa)	1356.30	1657.70	1507.0
B (MPa)	158.94	194.26	177.6
n	0.02	0.15	0.12
C	0.00	0.04	0.015
m	0.85	1.15	1.0

However, stress-strain data for RHA at several strain rates and temperatures has been obtained from MIDAS[11] and is shown in Fig. 2. Uncertainty measures thus can be estimated from this data. Ramsey[12] has discussed two approaches to obtaining uncertainty measures on material parameters for RHA. One of these, which is recommended by Smith,[13] is to use Bayesian methods such as Markov Chain Monte Carlo, which can be used to both fit a parameterized model and estimate a probability distribution for the values of the fitted parameters. Random samples from this distribution can be fed into another computational model in order to do uncertainty propagation analysis. Unfortunately, when a typical Bayesian analysis is performed, where the data is presumed to be a sum of the model prediction and random error, the joint probability density function (PDF) of the distribution of parameters becomes narrower and more sharply peaked as more experimental data is obtained, *regardless* of how accurately the parameterized model predicts the experimental data.[14,15] A measure of the misfit of the model can be obtained through a so-called nuisance parameter, but this cannot be propagated through another computational model.

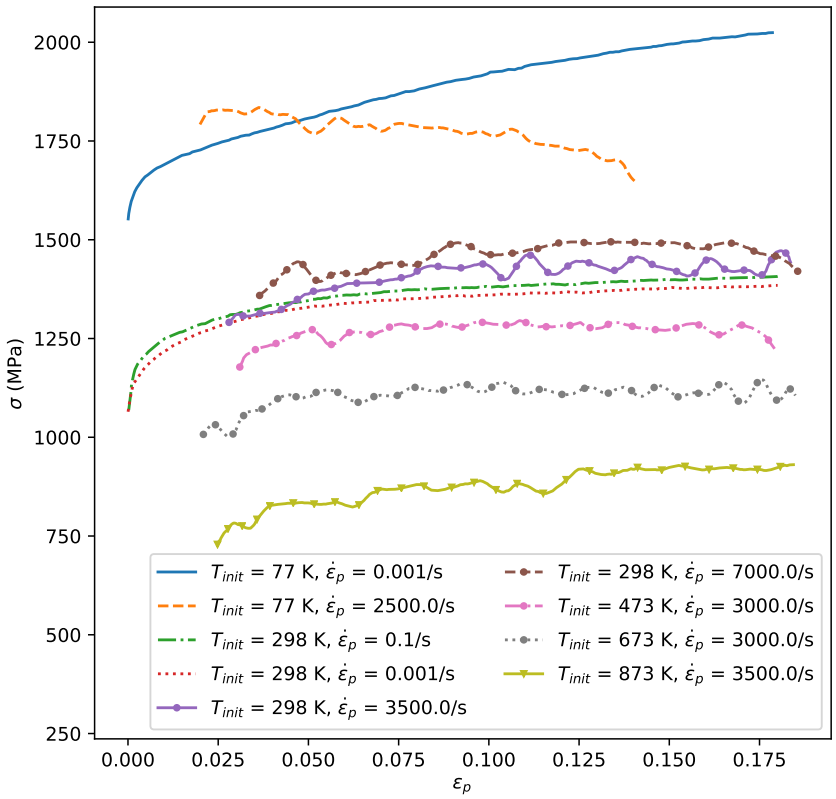

Fig. 2 Plots of flow stress σ vs. plastic strain ϵ_p for RHA from MIDAS, with the plastic strain rate denoted as $\dot{\epsilon}_p$ and the initial sample temperature T_{init}

Another approach involves approximating an interval predictor model (IPM), which is simply a function that returns an interval as its output rather than a single value. Typically, an IPM is built from a function $f(\mathbf{x};\boldsymbol{\theta})$ where $\boldsymbol{\theta}$ is a vector of parameters within the set $\boldsymbol{\Theta}$. The lower and upper bounds returned by the IPM for a given input $\mathbf{x}$, then, are[16]

$$y_{\min}(\mathbf{x};\boldsymbol{\Theta}) = \min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} f(\mathbf{x};\boldsymbol{\theta}) \tag{5}$$

$$y_{\max}(\mathbf{x};\boldsymbol{\Theta}) = \max_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} f(\mathbf{x};\boldsymbol{\theta}) \tag{6}$$

The set $\boldsymbol{\Theta}$ can be any shape, but it is typically taken to be one that can be readily parameterized, such as a hyperellipse[17] or a hyperrectangle.[16,18] In particular, the

latter can be parameterized by the corners of the hyperrectangle that represent lower and upper bounds for the elements of $\boldsymbol{\theta}$. The parameters of the set $\boldsymbol{\Theta}$ are chosen so that, given a data set $\{\mathbf{x}_i, y_i\}$, $i \in [1, N]$, they satisfy the following constrained minimization problem[16]:

$$\boldsymbol{\Theta} = \arg\min_{\boldsymbol{\Theta}^*} \frac{1}{N} \sum_{i=1}^{N} |y_{\max}(\mathbf{x}_i; \boldsymbol{\Theta}^*) - y_{\min}(\mathbf{x}_i; \boldsymbol{\Theta}^*)| \tag{7a}$$

$$y_{\min}(\mathbf{x}_i; \boldsymbol{\Theta}^*) \leq y_i \leq y_{\max}(\mathbf{x}_i; \boldsymbol{\Theta}^*), \quad \forall i \in [1, N] \tag{7b}$$

For a general $f(\mathbf{x}; \boldsymbol{\theta})$, the constrained minimization in Eq. 7 may entail a possibly unfeasible nested optimization problem. At a given iteration of the outer minimization, determining if Eq. 7b holds requires evaluating $y_{\min}(\mathbf{x}_i; \boldsymbol{\Theta}^*)$ and $y_{\max}(\mathbf{x}_i; \boldsymbol{\Theta}^*)$, and directly using Eqs. 5 and 6 to evaluate these quantities requires solving both a minimization and a maximization problem for each value of $\mathbf{x}_i$. However, if $\boldsymbol{\Theta}$ is a hyperrectangle and $f(\mathbf{x}; \boldsymbol{\theta})$ is a linear function of the elements of $\boldsymbol{\theta}$, then the solutions to the minimization in Eq. 5 and the maximization in Eq. 6 are linear functions of the lower and upper bounds of the elements of $\boldsymbol{\theta}$. No nested optimization is then required, and Eq. 7 becomes a linear programming problem.[18] To take advantage of this result even if $f(\mathbf{x}; \boldsymbol{\theta})$ is nonlinear in $\boldsymbol{\theta}$, one can take $\boldsymbol{\Theta}$ to be the hyperrectangular interval $[\boldsymbol{\theta}_0 - \Delta\boldsymbol{\theta}_{\min}, \boldsymbol{\theta}_0 + \Delta\boldsymbol{\theta}_{\max}]$ and approximate $f(\mathbf{x}; \boldsymbol{\theta})$ as a first-order Taylor expansion about $\boldsymbol{\theta}_0$. Then,[12]

$$y_{\min}(\mathbf{x}; \boldsymbol{\Theta}) = f(\mathrm{x}, \boldsymbol{\theta}_0) - \frac{1}{2} (\mathbf{g}(\mathbf{x}) + |\mathbf{g}(\mathbf{x})|)' \Delta\boldsymbol{\theta}_{\min} + \frac{1}{2} (\mathbf{g}(\mathbf{x}) - |\mathbf{g}(\mathbf{x})|)' \Delta\boldsymbol{\theta}_{\max} \tag{8}$$

$$y_{\max}(\mathbf{x}; \boldsymbol{\Theta}) = f(\mathrm{x}, \boldsymbol{\theta}_0) - \frac{1}{2} (\mathbf{g}(\mathbf{x}) - |\mathbf{g}(\mathbf{x})|)' \Delta\boldsymbol{\theta}_{\min} + \frac{1}{2} (\mathbf{g}(\mathbf{x}) + |\mathbf{g}(\mathbf{x})|)' \Delta\boldsymbol{\theta}_{\max} \tag{9}$$

where $\mathbf{g}(\mathbf{x})$ is the gradient of $f(\mathbf{x}; \boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$ evaluated at $\boldsymbol{\theta}_0$, $|\mathbf{g}(\mathbf{x})|$ is the *elementwise* absolute value of $\mathbf{g}(\mathbf{x})$, and the superscripted prime ("′") indicates the transpose. Equation 7a becomes

$$\Delta\boldsymbol{\theta}_{\min}, \Delta\boldsymbol{\theta}_{\max} = \arg\min_{\Delta\boldsymbol{\theta}^*_{\min}, \Delta\boldsymbol{\theta}^*_{\max}} \frac{1}{N} \left[\sum_{i=1}^{N} |\mathbf{g}(\mathbf{x})| \right]' (\Delta\boldsymbol{\theta}^*_{\min} + \Delta\boldsymbol{\theta}^*_{\max}) \tag{10}$$

Here, the elements of $\Delta\boldsymbol{\theta}_{\min}$ and $\Delta\boldsymbol{\theta}_{\max}$ are all positive.

The approximate IPM approach can be used to find bounds for the Johnson-Cook and Zerilli-Armstrong (BCC) parameters (i.e., $[\boldsymbol{\theta}_0 - \Delta\boldsymbol{\theta}_{\min}, \boldsymbol{\theta}_0 + \Delta\boldsymbol{\theta}_{\max}]$), with f

and $\boldsymbol{\theta}$ taken to be either σ_{JC} and $\boldsymbol{\theta}_{\text{JC}}$ or $\sigma_{\text{ZA,BCC}}$ and $\boldsymbol{\theta}_{\text{ZA,BCC}}$, and $\mathbf{x}$ taken to be $(\epsilon_p, \dot{\epsilon}_p, T)$. Some care, though, needs to be taken when evaluating the gradients of σ_{JC} or $\sigma_{\text{ZA,BCC}}$ at certain values. The analytical expressions for these gradients contain the expressions $\epsilon_p^n \ln \epsilon_p$ and $(T^*)^n \ln T^*$, where $T^* \equiv (T - T_{\text{ref}})(T_{\text{melt}} - T_{\text{ref}})$. These expressions are undefined when $\epsilon_p = 0$ or $T = T_{\text{ref}}$, but have a finite limit of zero as $\epsilon_p \to 0$ or $T \to T_{\text{ref}}$. Accordingly, when $\epsilon_p = 0$ or $T = T_{\text{ref}}$, the gradients of $\boldsymbol{\theta}_{\text{JC}}$ or $\sigma_{\text{ZA,BCC}}$ are taken to be their respective limiting values. In principle, $\boldsymbol{\theta}_0$ can be set to any "best fit" estimate of $\boldsymbol{\theta}$, such as one from a least-squares fit. Since Bayesian analyses of the Johnson-Cook and Zerilli-Armstrong models have been done prior to analyses with the approximate IPM approach, best fit estimates have been taken from these Bayesian analyses. For reasons discussed by Ramsey,[12] the best fit estimate for the Johnson-Cook model is taken to be the mean of the joint PDF of $\boldsymbol{\theta}_{\text{JC}}$ given a strong prior on A centered on an estimate of the yield stress of RHA based on data from Benck,[19] and the best fit estimate for the Zerilli-Armstrong (BCC) model is taken to be the mean of the joint PDF of $\boldsymbol{\theta}_{\text{ZA,BCC}}$ when the model is fit to the stress-strain data taken for initial sample temperatures of 298 K and above, ignoring the stress-strain curves shown in Fig. 2 for an initial temperature of 77 K. Tables 3 and 4 show the lower and upper bounds based off of these best fit estimates. For several parameters, namely A, B, C_0, C_1, C_3, and C_5, the lower and upper bounds are the same.

Table 3 Lower and upper bounds of Johnson-Cook parameters for RHA

Parameter	Lower bound	Upper bound
A (MPa)	700.618008	700.618008
B (MPa)	865.515315	865.515315
n	0.072011	0.123377
C	0.004541	0.007103
m	0.874260	1.051230

Table 4 Lower and upper bounds of Zerilli-Armstrong (BCC) parameters for RHA

Parameter	Lower bound	Upper bound
C_0 (MPa)	1.8678995	1.8678995
C_1 (MPa)	1535.4145842	1535.4145842
C_3 (K^{-1})	0.0013996	0.0013996
C_4 (K^{-1})	0.0000057	0.0000489
C_5 (MPa)	590.7882786	590.7882786
n	0.1783322	0.2162962

4. Estimating Penetration Depth Uncertainties

Three software tools have been used to estimate the uncertainties in the penetration depth: a modified version of Dakota 6.9,[2] OpenTURNS 1.12,[4] and UQPy 2.0.1.[5] Dakota has been modified to account for a gap in its functionality. While it is able to fit a metamodel to data (e.g., a Gaussian process) and save it to a file, it cannot read that file and use it for subsequent analyses. The modified version of Dakota, though, can do so. Dakota is an application that primarily works by reading in an input file and applying routines internal to the application to perform sampling, optimization, analysis, etc. On the other hand, OpenTURNS and UQPy are Python modules whose functionality can be accessed through Python scripts, which can make use of functionality (e.g., other Python modules or custom scripting) outside of OpenTURNS or UQPy. These tools have been used for the following tasks:

- Creating and cross-validating a Gaussian process metamodel from the results of armor penetration simulations, and then saving it to a file for later reuse.
- Interval analysis, that is, estimation of the lower and upper bounds on the penetration depth given the lower and upper bounds on input parameters.
- Performing Latin hypercube sampling (LHS) and obtaining moments (such as the mean, standard deviation, etc.) from the resulting samples.

4.1 Creating Metamodels from Armor Penetration Simulations

Section 2 describes the setup for the armor penetration simulations. As pointed out in that section, the plasticity model for the target may be either the Johnson-Cook or Zerilli-Armstrong (BCC) models. Accordingly, two sets of simulations have been done, one for each model. Each set consists of 128 individual CTH simulations. The simulations are identical except for their plasticity model parameters. Dakota's implementation of LHS is used to generate samples of these model parameters. In the LHS, the lower and upper bounds of the penetrator parameters are those in Table 2. When the Johnson-Cook model is used for the target, the lower bounds of the parameters of the target are 90% of the lower bounds in Table 3, and the upper bounds are 110% of the upper bounds shown in that table. Similarly, when the Zerilli-Armstrong (BCC) model is used for the target, the lower and upper bounds of the parameters of the target are 90% and 110%, respectively, of the lower and

upper bounds in Table 4. After each CTH simulation has finished, the penetration depth is extracted from the simulation output as described in Ramsey.[1] The output from each set of simulations, then, is 128 samples of input-output pairs, where the input consists of material parameters for both penetrator and target, and the output is the penetration depth.

For each set of CTH simulations, three Gaussian process metamodels are fit: one from Dakota, one from OpenTURNS, and one from UQPy. By default, Dakota takes the mean of the Gaussian process to be a quadratic polynomial of the inputs without interaction terms,[20] and it also requires the correlation function to be the Gaussian or squared exponential,[2]

$$R(\mathbf{x}, \mathbf{x}^*) = \exp\left[-\frac{1}{2}\sum_{i}^{D}\left(\frac{x_i - x_i^*}{L_i}\right)^2\right] \tag{11}$$

where $\mathbf{x}$ and $\mathbf{x}^*$ are inputs, D is the length of $\mathbf{x}$, and L_i is a correlation length. To be approximately consistent with these defaults, the same correlation function is used with both OpenTURNS and UQPy, and quadratic polynomials are used for the means (though neither OpenTURNS nor UQPy appears to be able to exclude interaction terms). The correlation lengths are determined automatically within Dakota, OpenTURNS, and UQPy. For the most part, Dakota is designed to create a metamodel (or "surrogate" as it is called in the Dakota example file and manuals), keep it in memory, and then apply it to some other analysis, all within the same run of the Dakota executable. Accordingly, just to create a metamodel in Dakota apparently requires specifying a "dummy" analysis, such as a random sampling where the number of samples is zero, in order to trigger building the metamodel. (An example of this is in the file `share/dakota/examples/eval_surrogate/dakota_surrogate.in` from the Dakota installation.) With OpenTURNS and UQPy, the creation of the metamodel is more straightforward: a Python script reads in data, feeds it to a Python object that fits a metamodel to the data (`Krig` in UQPy, `KrigingAlgorithm` in OpenTURNS), and then saves the metamodel object to a file. With OpenTURNS, though, there is an additional complication; normalization of inputs is the responsibility of the Python script that creates the metamodel.[21] By contrast, both UQPy and Dakota handle normalization transparently, internally operating on a normalized copy of the data. When using OpenTURNS to fit a metamodel to the simulation results, each input is normalized to the interval [0, 1]. Nei-

ther OpenTURNS nor UQPy has an *explicit* capability to save metamodels to files. Rather, for OpenTURNS, the metamodel object is saved using the `pickle` module that is a standard part of Python. For UQPy, the functionality of the `pickle` module is not sufficient to save a metamodel generated with it. Instead, the third-party module `dill`, version 0.2.9,[22] is used. The Python implementation used with both OpenTURNS and UQPy is version 3.6.8 from Anaconda, Inc.[23]

After each metamodel has been constructed, it is tested by k-fold cross-validation. The set of samples to which the metamodel has originally been fit is divided into k subsets. For $i \in [1, k]$, a new metamodel $f^{(-i)}(\mathbf{x})$ is fit to all subsets except for the i^{th} one. Then for each i, the following sums are calculated:

$$\Delta^i_{\text{sq,err}} = \sum_{\mathbf{x}\in I_i} \left[f^{\text{orig}}(\mathbf{x}) - f^{-i}(\mathbf{x}) \right]^2 \tag{12}$$

$$\Delta^i_{\text{abs,err}} = \sum_{\mathbf{x}\in I_i} \left| f^{\text{orig}}(\mathbf{x}) - f^{-i}(\mathbf{x}) \right| \tag{13}$$

Here, $f^{\text{orig}}(\mathbf{x})$ is the original metamodel, I_i is i^{th} subset of input samples, and $\mathbf{x}$ is one of the inputs from that subset. The root mean square (RMS) error, $\sqrt{\sum_{i=1}^{k} \Delta^i_{\text{sq,err}}/N_s}$, and mean absolute error, $\sum_{i=1}^{k} \Delta^i_{\text{abs,err}}/N_s$, where N_s is the total number of samples, can then be calculated. When k-fold cross-validation is done for the special case of $k = N_s$, it is also known as leave-one-out (LOO) cross-validation.[2] With Dakota, doing cross-validation is a matter of adding a few keywords to an input file. While neither OpenTURNS nor UQPy explicitly contain functionality to do cross-validation,* one can implement cross-validation with a custom Python script. When fitting the metamodel $f^{(-i)}(\mathbf{x})$ in OpenTURNS, the maximum possible size of the squared correlation lengths is set to 10^5 by setting the item in OpenTURNS' `ResourceMap` with key `GeneralLinearModel-Algorithm-DefaultOptimizationScaleFactor` to `"100000"`. Both 5-fold and LOO cross-validation have been done, and the results are shown in Tables 5 and 6.

*The OpenTURNS classes `KFold` and `CorrectedLeaveOneOut` are, in the words of the documentation of OpenTURNS, "not usable" because they only make sense within the implementation of a particular class within OpenTURNS.[24,25]

Table 5 Cross-validation results for the metamodels where the Johnson-Cook model is the plasticity model for the target

Software	5-fold		LOO	
	RMS error (cm)	Mean abs. error (cm)	RMS error (cm) (cm)	Mean abs. error (cm)
Dakota	0.0233313	0.0169573	0.0225512	0.0143692
OpenTURNS	0.0210885	0.0154039	0.0186467	0.0133622
UQPy	0.0399600	0.0294449	0.0353926	0.0260735

Table 6 Cross-validation results for the metamodels where the Zerilli-Armstrong (BCC) model is the plasticity model for the target

Software	5-fold		LOO	
	RMS error (cm)	Mean abs. error (cm)	RMS error (cm) (cm)	Mean abs. error (cm)
Dakota	0.0164004	0.0116212	0.0189751	0.0104531
OpenTURNS	0.0258654	0.0168240	0.0210045	0.0147231
UQPy	0.0280902	0.0186461	0.0222536	0.0154479

The central processing unit (CPU) times needed to create and validate the metamodels using the various software tools are shown in Tables 7 and 8. Here and in the rest of this work, CPU times are from a MacBook Pro running macOS 10.13.6 on a quad-core 2.9-GHz Intel Core i7 processor and 16 GB of memory. In Dakota, both creation and cross-validation were done together in a single run, so only the total CPU time is available.

Table 7 Metamodel creation and cross-validation approximate CPU times in seconds for the metamodels where the Johnson-Cook model is the plasticity model for the target

Software	Creation	5-fold	LOO	Total
Dakota	N/A	N/A	N/A	25.7
OpenTURNS	4.5	2.4	31.0	37.8
UQPy	4.1	15.0	535.3	555.4

Table 8 Metamodel creation and cross-validation approximate CPU times in seconds for the metamodels where the Zerilli-Armstrong (BCC) model is the plasticity model for the target

Software	Creation	5-fold	LOO	Total
Dakota	N/A	N/A	N/A	28.5
OpenTURNS	6.4	2.5	36.3	45.2
UQPy	4.0	15.8	491.8	511.5

4.2 Interval Analysis

Dakota has four built-in methods for interval analysis.[20] One of them is a simplistic, brute-force approach. LHS is used to generate a set of possible inputs, then each possible input is fed into a computational model (which here is the Gaussian process metamodel of armor penetration) to obtain its corresponding output (here, the penetration depth). The lower and upper bounds of the output are then taken to be the minimum and maximum of the generated outputs. As pointed out by Ramsey,[1] this approach tends to overestimate the lower bound and underestimate the upper bound, so it is not further pursued here.

The other three interval analysis methods in Dakota have the same overarching structure. The lower bound of the output (which again is the penetration depth) is found by doing a constrained minimization of the computational model (which again is the Gaussian process metamodel of armor penetration) where the constraints are the lower and upper bounds of the model inputs. The upper bound is found similarly, but with a constrained maximization instead of a minimization. Internally, the constrained maximization in Dakota is simply a constrained minimization of the negative of the computational model (i.e., $\max f(\mathbf{x}) = -\min[-f(\mathbf{x})]$). The difference between these other three methods is the choice of constrained minimization algorithm:

- *Efficient global optimization*. This algorithm[26] is designed to minimize the number of evaluations of the computational model (since in general it may be expensive to evaluate) and involves iteratively fitting Gaussian process metamodels to samples from the computational model. Since the computational model used in the current work is already a Gaussian process metamodel, this entails fitting a Gaussian process over another Gaussian process. Unfor-

tunately, the implementation of EGO in Dakota performs poorly with the armor penetration metamodel, reaching the maximum number of computational model evaluations or iterations before reaching the desired convergence tolerance. On the mailing list for Dakota users, it has been suggested that the problem may be that the samples from the computational model generated during minimization have all clustered together, causing numerical problems when fitting a Gaussian process to those samples.[27]

- *Evolutionary algorithm*. This algorithm is designed to resemble the process of Darwinial natural selection. The implementation in Dakota does not respect any convergence tolerance set in the input file. Rather, as indicated by the function `check_tolerance()` in the source code file `packages/external/acro/packages/colin/src/colin/solver/ColinSolver.h`, the algorithm terminates when a maximum wall-clock time, number of iterations, or number of computational model evaluations is reached, or when the output of the computational model is no greater than a predefined target value. Accordingly, one needs to assess convergence of this algorithm by monitoring if its output ceases to change as the number of iterations increases.

- *Surrogate-based optimization*. This algorithm applies the aforementioned evolutionary algorithm over a Gaussian process metamodel of the computational model. Again, since the computational model used in this current work is already a Gaussian process metamodel, this entails fitting a Gaussian process over another Gaussian process.

For the current work, the evolutionary algorithm appears to be the least problematic among the algorithms that Dakota makes available for interval analysis. This algorithm has been run for 129, 257, 513, 1025, 2049, 4093, and 8193 iterations, though the results did not change after 513 iterations.

Neither OpenTURNS nor UQPy explicitly has features for interval analysis. However, what is done implictly within Dakota can be done explicitly with custom Python scripting. That is, one can perform constrained minimization and maximization to obtain the lower and upper bounds on the penetration depth. As mentioned before, the Python scripts used with OpenTURNS or UQPy can readiliy access functionality outside of either of those modules. Here, the functionality is

a minimization routine from the `optimize` submodule of SciPy named `shgo`, which implements simplicial homology global optimization.[28] Again, maximization is simply minimization of the negative of the armor penetration metamodel.

Interval analysis has been done for the following four conditions: (1) Johnson-Cook model used for the target, lower and upper bounds for penetrator and target parameters set to those in Tables 2 and 3; (2) Johnson-Cook model used for the target, penetrator parameters fixed at nominal values in Table 2, lower and upper bounds for target parameters set to those in Table 3; (3) Zerilli-Armstrong (BCC) model used for the target, lower and upper bounds for penetrator and target parameters set to those in Tables 2 and 4; and (4) Zerilli-Armstrong (BCC) model used for the target, penetrator parameters fixed at nominal values in Table 2, lower and upper bounds for target parameters set to those in Table 4. Results are shown in Tables 9 and 10. CPU times to obtain the results are shown in Table 11.

Table 9 Lower and upper bounds for the penetration depth with the Johnson-Cook model used for the target, both for the case where the penetrator parameters are allowed to vary between lower and upper bounds, and for the case where the penetrator parameters are fixed at nominal values

Software	Penetrator parameters varied		Penetrator parameters fixed	
	Lower bound (cm)	Upper bound (cm)	Lower bound (cm)	Upper bound (cm)
Dakota	8.13	9.39	8.34	9.18
OpenTURNS	8.11	9.41	8.35	9.19
UQPy	8.13	9.44	8.35	9.17

Table 10 Lower and upper bounds for the penetration depth with the Zerilli-Armstrong (BCC) model used for the target, both for the case where the penetrator parameters are allowed to vary between lower and upper bounds, and for the case where the penetrator parameters are fixed at nominal values

Software	Penetrator parameters varied		Penetrator parameters fixed	
	Lower bound (cm)	Upper bound (cm)	Lower bound (cm)	Upper bound (cm)
Dakota	7.79	9.28	8.03	9.03
OpenTURNS	7.79	9.30	8.03	9.03
UQPy	7.77	9.28	8.04	9.03

Table 11 CPU times in seconds for interval analysis. For Dakota, "iters." refers to the number of iterations of the evolutionary algorithm

Software	Penetrator parameters varied		Penetrator parameters fixed	
	Target params. Johnson-Cook	Target params. Zerilli-Armstrong	Target params. Johnson-Cook	Target params. Zerilli-Armstrong
Dakota, 2049 iters.	353.9	320.0	223.6	222.2
Dakota, 8193 iters.	1297.3	1186.7	933.7	953.4
OpenTURNS	11.8	2.9	1.2	1.2
UQPy	4.9	1.3	0.9	0.9

4.3 Latin Hypercube Sampling

LHS samples have been generated for the same four conditions as the interval analysis. In Dakota, an LHS sample can be extended, doubling the number of samples while retaining the previous ones. This has been done for all LHS samples generated in Dakota that are larger than 128 samples. Neither OpenTURNS nor UQPy has this extension capability, so for example, the LHS with 256 samples does not share samples with the LHS with 128 samples. Moments of the LHS samples are shown in Tables 12–23. For the sake of brevity, only moments for sample sizes that are odd powers of 2 are shown. Dakota calculates skewness and kurtosis according to the formulas for G_1 and G_2 from Joanes and Gill,[29] and these formulas have been used to calculate the skewness and kurtosis from the LHS samples from OpenTURNS and UQPy as well.

Tables 24 and 25 show the CPU time and memory usage for parts of LHS. These tables show the CPU time to generate all the LHS samples using Dakota, the CPU time and memory to extend an LHS with 262144 to an LHS with 524288 samples using Dakota, the CPU time and memory to generate an LHS with 524288 samples using OpenTURNS, and the CPU time and memory to generate an LHS with 524288 samples using UQPy in two different ways. In one of these ways, the $N_s = 524288$ sample outputs are generated in a vectorized fashion, with the

Gaussian process metamodel taking in a whole $N_s \times D$ matrix of sample inputs and returning a $N_s \times 1$ column vector of sample outputs. The other way also creates a $N_s \times 1$ column vector of sample outputs, but the N_s sample inputs are fed into the metamodel one at a time in the body of a Python loop. With OpenTURNS, sample outputs are always generated in a vectorized fashion in this work.

Table 12 Moments from Latin hypercube samples from Dakota, with the Johnson-Cook model used for the target and penetrator parameters allowed to vary between lower and upper bounds

Number of samples	Mean (cm)	Standard deviation (cm)	Skewness	Kurtosis
128	8.7429	0.1681	−0.1336	−0.4807
512	8.7428	0.1691	0.0717	−0.5380
2048	8.7430	0.1676	0.0325	−0.2865
8192	8.7430	0.1676	0.0087	−0.3814
32768	8.7430	0.1673	0.0074	−0.3966
131072	8.7430	0.1671	0.0236	−0.3794
524288	8.7430	0.1671	0.0221	−0.3772

Table 13 Moments from Latin hypercube samples from OpenTURNS, with the Johnson-Cook model used for the target and penetrator parameters allowed to vary between lower and upper bounds

Number of samples	Mean (cm)	Standard deviation (cm)	Skewness	Kurtosis
128	8.7454	0.1569	−0.0581	0.0092
512	8.7454	0.1670	−0.0072	−0.3036
2048	8.7453	0.1668	0.0545	−0.3828
8192	8.7455	0.1689	0.0523	−0.3051
32768	8.7456	0.1689	0.0444	−0.3801
131072	8.7455	0.1691	0.0606	−0.3664
524288	8.7455	0.1689	0.0652	−0.3876

Table 14 Moments from Latin hypercube samples from UQPy, with the Johnson-Cook model used for the target and penetrator parameters allowed to vary between lower and upper bounds

Number of samples	Mean (cm)	Standard deviation (cm)	Skewness	Kurtosis
128	8.7373	0.1594	−0.1261	0.0898
512	8.7373	0.1761	0.1157	−0.4041
2048	8.7377	0.1688	0.0572	−0.3243
8192	8.7375	0.1697	0.0657	−0.3343
32768	8.7374	0.1708	0.0634	−0.3623
131072	8.7374	0.1699	0.0975	−0.3456
524288	8.7374	0.1704	0.0856	−0.3491

Table 15 Moments from Latin hypercube samples from Dakota, with the Johnson-Cook model used for the target and penetrator parameters fixed at nominal values

Number of samples	Mean (cm)	Standard deviation (cm)	Skewness	Kurtosis
128	8.7417	0.1439	0.1141	−0.6550
512	8.7416	0.1444	0.0566	−0.4313
2048	8.7415	0.1446	0.0848	−0.4739
8192	8.7415	0.1444	0.0703	−0.4803
32768	8.7416	0.1447	0.0527	−0.4989
131072	8.7416	0.1449	0.0462	−0.5167
524288	8.7416	0.1450	0.0473	−0.5172

Table 16 Moments from Latin hypercube samples from OpenTURNS, with the Johnson-Cook model used for the target and penetrator parameters fixed at nominal values

Number of samples	Mean (cm)	Standard deviation (cm)	Skewness	Kurtosis
128	8.7470	0.1625	0.2071	−0.7301
512	8.7464	0.1569	0.0474	−0.6562
2048	8.7465	0.1492	0.1021	−0.4865
8192	8.7464	0.1485	0.0757	−0.5267
32768	8.7465	0.1498	0.1009	−0.5466
131072	8.7464	0.1493	0.0757	−0.5638
524288	8.7465	0.1494	0.0835	−0.5610

Table 17 Moments from Latin hypercube samples from UQPy, with the Johnson-Cook model used for the target and penetrator parameters fixed at nominal values

Number of samples	Mean (cm)	Standard deviation (cm)	Skewness	Kurtosis
128	8.7338	0.1424	−0.1026	−0.3977
512	8.7340	0.1469	0.0993	−0.5988
2048	8.7339	0.1439	0.1386	−0.5757
8192	8.7340	0.1459	0.1155	−0.5623
32768	8.7340	0.1460	0.0861	−0.5890
131072	8.7340	0.1453	0.0879	−0.5485
524288	8.7340	0.1453	0.0930	−0.5399

Table 18 Moments from Latin hypercube samples from Dakota, with the Zerilli-Armstrong (BCC) model used for the target and penetrator parameters allowed to vary between lower and upper bounds

Number of samples	Mean (cm)	Standard deviation (cm)	Skewness	Kurtosis
128	8.5313	0.2629	0.1388	−0.9024
512	8.5310	0.2618	0.0687	−0.9112
2048	8.5309	0.2615	0.0454	−0.9434
8192	8.5309	0.2614	0.0330	−0.9320
32768	8.5309	0.2612	0.0214	−0.9185
131072	8.5308	0.2612	0.0186	−0.9258
524288	8.5308	0.2612	0.0190	−0.9238

Table 19 Moments from Latin hypercube samples from OpenTURNS, with the Zerilli-Armstrong (BCC) model used for the target and penetrator parameters allowed to vary between lower and upper bounds

Number of samples	Mean (cm)	Standard deviation (cm)	Skewness	Kurtosis
128	8.5317	0.2646	0.1062	−1.1761
512	8.5315	0.2599	−0.0446	−0.9843
2048	8.5310	0.2631	0.0079	−0.9269
8192	8.5312	0.2601	−0.0011	−0.9344
32768	8.5313	0.2603	0.0305	−0.8960
131072	8.5313	0.2610	0.0238	−0.9089
524288	8.5313	0.2607	0.0232	−0.9136

Table 20 Moments from Latin hypercube samples from UQPy, with the Zerilli-Armstrong (BCC) model used for the target and penetrator parameters allowed to vary between lower and upper bounds

Number of samples	Mean (cm)	Standard deviation (cm)	Skewness	Kurtosis
128	8.5286	0.2669	0.0716	−0.8340
512	8.5315	0.2675	0.0003	−0.9030
2048	8.5309	0.2620	0.0252	−0.9238
8192	8.5304	0.2616	0.0063	−0.9080
32768	8.5306	0.2625	0.0136	−0.9107
131072	8.5307	0.2627	0.0130	−0.9236
524288	8.5306	0.2621	0.0140	−0.9206

Table 21 Moments from Latin hypercube samples from Dakota, with the Zerilli-Armstrong (BCC) model used for the target and penetrator parameters fixed at nominal values

Number of samples	Mean (cm)	Standard deviation (cm)	Skewness	Kurtosis
128	8.5269	0.2503	0.0119	−1.2284
512	8.5272	0.2479	−0.0089	−1.1509
2048	8.5272	0.2474	−0.0052	−1.1225
8192	8.5272	0.2475	0.0024	−1.1222
32768	8.5272	0.2478	0.0053	−1.1277
131072	8.5272	0.2477	0.0006	−1.1274
524288	8.5272	0.2477	0.0015	−1.1278

Table 22 Moments from Latin hypercube samples from OpenTURNS, with the Zerilli-Armstrong (BCC) model used for the target and penetrator parameters fixed at nominal values

Number of samples	Mean (cm)	Standard deviation (cm)	Skewness	Kurtosis
128	8.5266	0.2462	−0.0931	−1.1977
512	8.5268	0.2493	−0.0177	−1.1350
2048	8.5267	0.2478	0.0021	−1.1346
8192	8.5267	0.2477	0.0029	−1.1134
32768	8.5268	0.2483	0.0013	−1.1291
131072	8.5268	0.2483	−0.0021	−1.1263
524288	8.5268	0.2482	−0.0010	−1.1289

Table 23 Moments from Latin hypercube samples from UQPy, with the Zerilli-Armstrong (BCC) model used for the target and penetrator parameters fixed at nominal values

Number of samples	Mean (cm)	Standard deviation (cm)	Skewness	Kurtosis
128	8.5306	0.2531	−0.0059	−1.2088
512	8.5300	0.2456	−0.0056	−1.1148
2048	8.5303	0.2484	−0.0168	−1.1578
8192	8.5303	0.2486	−0.0083	−1.1350
32768	8.5303	0.2481	−0.0049	−1.1321
131072	8.5303	0.2483	−0.0041	−1.1282
524288	8.5303	0.2487	−0.0052	−1.1297

Table 24 CPU times in seconds for LHS

Software	Penetrator parameters varied		Penetrator parameters fixed	
	Target params. Johnson-Cook	Target params. Zerilli-Armstrong	Target params. Johnson-Cook	Target params. Zerilli-Armstrong
Dakota, all samples	434.3	434.8	331.1	331.3
Dakota, last LHS extension	222.7	226.2	170.4	168.1
OpenTURNS, largest sample	117.6	117.5	106.9	113.5
UQPy, largest sample, vectorized evaluation	145.9	160.9	143.6	162.6
UQPy, largest sample, loop evaluation	155.4	172.5	159.8	168.6

Table 25 Memory usage in gigabytes for LHS

Software	Penetrator parameters varied		Penetrator parameters fixed	
	Target params. Johnson-Cook	**Target params. Zerilli-Armstrong**	**Target params. Johnson-Cook**	**Target params. Zerilli-Armstrong**
Dakota, last LHS extension	2.43	2.41	2.28	2.27
OpenTURNS, largest sample	0.48	0.47	0.34	0.33
UQPy, largest sample, vectorized evaluation	10.46	11.19	10.57	11.19
UQPy, largest sample, loop evaluation	0.21	0.20	0.16	0.15

5. Discussion and Conclusions

In the previous work by Ramsey,[1] when examining the problem described in Section 2 (where the element size in CTH is still 0.05 cm), the narrowest interval from interval analysis—where only the four most sensitive parameters A, B, C, and n of the target vary—predicts that the penetration depth could range from 5.9 to 10.1 cm, while the moments generated from LHS samples indicate that the true value of the penetration depth has at least an 88.8% probability of being within the interval [5.9 cm, 9.8 cm], according to Chebyshev's inequality.[30] As can be seen in Tables 9 and 10, interval analysis predicts significantly narrow intervals, with widths roughly around 1.5 cm rather than approximately 4 cm as in the previous work. Similarly, depending on whether the Johnson-Cook model or the Zerilli-Armstrong (BCC) model is used for the target, the moments generated from LHS samples in Tables 12–14 or Tables 18–20, where the values of the penetrator are allowed to vary, indicate that the true value of the penetration depth has at least an 88.8% probability of being within the interval [8.2 cm, 9.3 cm] or [7.7 cm, 9.3 cm]. Fixing the penetrator's material parameters at nominal values only slightly narrows the intervals, to [8.3 cm, 9.1 cm] when the Johnson-Cook model is used for the target, or to [7.8 cm, 9.3 cm] when the Zerilli-Armstrong (BCC) model is used for the target. This slight difference is due to the mean being largely unaffected by variations in the penetrator parameters, while the standard deviation narrows slightly when the penetrator parameters are fixed. This trend has also appeared in the previous work. Changing the material model of the target has relatively small effects on the mean, shifting it by about 2 mm, and shifts the standard deviation by about 1 mm. As shown in Tables 9 and 10, the intervals predicted by the different material models largely overlap. This suggests that predictions of the penetration depth are insensitive to inaccuracies in these two models.

The different software tools yield largely the same results, so the question of which one is the "best" among them largely comes down to computational costs, availability of features, ease of use, and so on. Dakota has two major detriments. First, it has to be modified in order to take full advantage of the metamodels that it produces. Second, its choice of optimization algorithms for interval analysis is limited. Strictly speaking, one could say that since neither OpenTURNS nor UQPy implement interval analysis themselves, they offer no choice of algorithms for such analysis, but because they can be used within a larger Python ecosystem, they can

easily be used with any third-party algorithm with a Python interface. Furthermore, as can be seen in Table 11, interval analysis with them is far less costly than it is with Dakota in practice. Dakota has a few advantages, though. One need not explicitly choose a mean or correlation function; sensible defaults can be used. It is trivial to cross-validate a metamodel created by Dakota. It also has the capability to extend a previously available LHS sample, something lacking in both OpenTURNS and UQPy. That said, as indicated by Tables 24 and 25, the cost of generating even a large LHS need not be that high to begin with, though, as seen in Table 25, one should avoid inputting too many samples at once into a metamodel generated with UQPy. Overall, if one needs to use a purely off-the-shelf solution that requires no modifications, OpenTURNS appears to be the best of the software tools discussed in this work, having CPU times and memory usage that are at least competitive with Dakota.

The current work improves over the previous work applying UQ software tools to an armor penetration problem. In particular, two additional practices are introduced in this work. First, where feasible, find uncertainty measures for model parameters that have been obtained from the original experimental data, rather than make crude judgments based on rules of thumb such as taking the upper and lower bounds to be $\pm 10\%$ of available point estimates. Provided that the models being used are reasonable approximations to begin with, this should lead to tighter uncertainty measures (e.g., a narrower interval, smaller standard deviation) for the desired quantity of interest. Second, where feasible, apply multiple models. If different empirical models approximately agree even if they share little in common, this is a sign that they produce reasonable results over the range of input values that they encounter during an analysis, thus bolstering confidence in one's assessments of uncertainty of quantities of interest.

6. References

1. Ramsey JJ. Survey of existing uncertainty quantification capabilities for Army-relevant problems. Aberdeen Proving Ground (MD): Army Research Laboratory (US); 2017 Nov. Report No.: ARL-TR-8218.

2. Adams BM et al. Dakota, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: Version 6.9 user's manual. Albuquerque (NM): Sandia National Laboratories; 2018 Nov.

3. Pernot P. The parameter uncertainty inflation fallacy. The Journal of Chemical Physics. 2017;147(10):104102.

4. Baudin M, Dutfoy A, Iooss B, Popelin AL. OpenTURNS: An industrial software for uncertainty quantification in simulation. In: Ghanem R, Higdon D, Owhadi H, editors. Handbook of uncertainty quantification. Cham (Switzerland): Springer International Publishing; 2017. p. 2001–2038.

5. Shields MD, Giovanis DG, Satish AB, Chauhan MS, Olivier A, Vandanapu L, Zhang J. Uncertainty quantification with python (UQpy). c2019 [accessed 2019 Apr]. `https://github.com/SURGroup/UQpy`.

6. Hertel ES, 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: Brun R, Dumitrescu LZ, editors. Shock waves at Marseille I; Berlin (Heidelberg): Springer Berlin Heidelberg; 1995. p. 377–382.

7. 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.

8. 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; 1983. p. 541–547.

9. Gray GT III, Chen SR, Wright W, Lopez MF. Constitutive equations for annealed metals under compression at high strain rates and high temperatures.

Los Alamos (NM): Los Alamos National Laboratory; 1994 Jan. Report No.: LA-12669-MS.

10. Becker R. CCDC Army Research Laboratory, Aberdeen Proving Ground, MD. Personal Communication, 2017 Jan 12.

11. Lawrence Livermore National Laboratory. MIDAS: Material implementation, database, and analysis source. c2018 [accessed 2018 Mar]. https://pls.llnl.gov/people/divisions/physics-division/condensed-matter-science-section/eos-and-materials-theory-group/projects/midas-material-implementation-database-and-analysis-source.

12. Ramsey JJ. Quantifying uncertainties in parameterizations of strength models of rolled homogeneous armor: part 1, overview. Aberdeen Proving Ground (MD):CCDC Army Research Laboratory; Forthcoming 2019 Sep.

13. Smith RC. Uncertainty quantification: Theory, implementation, and applications. Philadelphia (PA): Society for Industrial and Applied Mathematics; 2014.

14. Brynjarsdóttir J, O'Hagan A. Learning about physical parameters: the importance of model discrepancy. Inverse Problems. 2014;30(11):114007.

15. Sargsyan K, Najm HN, Ghanem R. On the statistical calibration of physical models. International Journal of Chemical Kinetics. 47(4):246–276.

16. Crespo LG, Kenny SP, Giesy DP. Calibration of predictor models using multiple validation experiments. In: 17th AIAA Non-Deterministic Approaches Conference; American Institute of Aeronautics and Astronautics; 2015. p. 2015–0659.

17. Campi MC, Calafiore G, Garatti S. Interval predictor models: Identification and reliability. Automatica. 2009;45(2):382–392.

18. Crespo LG, Kenny SP, Giesy DP. Interval predictor models with a linear parameter dependency. Journal of Verification, Validation and Uncertainty Quantification. 2016;1(2):021007.

19. Benck RF. Quasi-static tensile stress strain curves: II, rolled homogeneous armor. Aberdeen Proving Ground (MD): Ballistic Research Laboratories (US); 1976 Nov. Report No.: 2703.

20. Adams BM et al. Dakota, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: Version 6.9 reference manual. Albuquerque (NM): Sandia National Laboratories; 2018 Nov.

21. Haddad S. Comment on Kriging optimization enhancements #728. c2018 [accessed 2019 Apr]. https://github.com/openturns/openturns/pull/728#issuecomment-376941796.

22. McKerns M. dill 0.2.9. c2019 [accessed 2019 Apr]. https://pypi.org/project/dill/.

23. Anaconda, Inc. Anaconda. c2018 [accessed 2018 Mar]. https://anaconda.com.

24. Dutfoy A, Dumas A, Ladier A, Barbier D, Martinez FA, Dutka-Malen I, Schueller J, Couplet M, Lapointe M, Souchaud M, Lebrun R, Conty R, Haddad S, Dubourg V. KFold. c2018 [accessed 2019 Apr]. http://openturns.github.io/openturns/latest/user_manual/response_surface/_generated/openturns.KFold.html.

25. Dutfoy A, Dumas A, Ladier A, Barbier D, Martinez FA, Dutka-Malen I, Schueller J, Couplet M, Lapointe M, Souchaud M, Lebrun R, Conty R, Haddad S, Dubourg V. CorrectedLeaveOneOut. c2018 [accessed 2019 Apr]. http://openturns.github.io/openturns/latest/user_manual/response_surface/_generated/openturns.CorrectedLeaveOneOut.html.

26. Jones DR, Schonlau M, Welch WJ. Efficient global optimization of expensive black-box functions. Journal of Global Optimization. 13:455–492.

27. Dalbey K. Sandia National Laboratories, Albuquerque, NM. Personal Communication, 2019 Apr 9.

28. SciPy community. scipy.optimize.shgo. c2019 [accessed 2019 Apr]. https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.shgo.html.

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

30. NIST. NIST/SEMATECH e-handbook of statistical methods: Approximate intervals that contain most of the population values. c2018 [accessed 2019 Apr]. https://www.itl.nist.gov/div898/handbook/prc/section2/prc261.htm.

$\boldsymbol{\theta}_{\text{JC}}$	fitting parameters of Johnson-Cook model
$\boldsymbol{\theta}_{\text{ZA,BCC}}$	fitting parameters of Zerilli-Armstrong (BCC) model
$\dot{\epsilon}_p$	plastic strain rate
$\dot{\epsilon}_{p0}$	reference plastic strain rate, $1/\text{s}$
ϵ_p	plastic strain
Γ_0	Grüeneisen parameter
ν	Poisson's ratio
ρ	density
ρ_0	initial density
σ_{JC}	flow stress according to Johnson-Cook model
$\sigma_{\text{ZA,BCC}}$	flow stress according to Zerilli-Armstrong (BCC) model
A	fitting parameter of Johnson-Cook model that represents yield strength at reference strain rate and room temperature
B	fitting parameter of Johnson-Cook model that represents strain hardening prefactor at reference strain rate and room temperature
C	fitting parameter of Johnson-Cook model that represents strain hardening effects due to strain rate
C_i	fitting parameter of Zerilli-Armstrong (BCC) model, where $i \in \{0, 1, 3, 4, 5\}$
C_S	speed of sound in a material, and denominator of quadratic coefficient of u_s-u_p curve
C_v	specific heat capacity at constant volume
D	length of the vector input to a Gaussian process metamodel

m	fitting parameter of Johnson-Cook model that represents thermal softening exponent
n	fitting parameter of Johnson-Cook and Zerilli-Armstrong models that represents strain hardening exponent
N_s	number of samples
P	pressure
S_1	linear coefficient of u_s-u_p curve
S_2	numerator of quadratic coefficient of u_s-u_p curve
T	temperature
T_0	initial temperature
T_{melt}	melting temperature
T_{ref}	reference temperature, typically room temperature
u_p	velocity of point in material
u_s	shock front velocity
BCC	body-centered cubic
CPU	central processing unit
EOS	equation of state
IPM	interval predictor model
LHS	Latin hypercube sampling
LOO	leave-one-out
OpenTURNS	Open Treatment of Uncertainties, Risks, 'N Statistics
PDF	probability density function
RHA	rolled homogeneous armor
RMS	root mean square

UQ	uncertainty quantification
UQPy	Uncertainty Quantification with Python
WHA	tungsten heavy alloy

1 (PDF) DEFENSE TECHNICAL INFORMATION CTR DTIC OCA

1 (PDF) GOVT PRINTG OFC A MALHOTRA

2 (PDF) DIR CCDC ARL IMAL HRA RECORDS MGMT FCDD RLD CL TECH LIB

1 (PDF) CCDC ARL FCDD RLC NB JJ RAMSEY