ARL-TR-8826 • SEP 2019

Quantifying Uncertainties in Parameterizations of Strength Models of Rolled Homogeneous Armor: Part 1, Overview

by JJ Ramsey

ARL-TR-8826 • SEP 2019

Quantifying Uncertainties in Parameterizations of Strength Models of Rolled Homogeneous Armor: Part 1, Overview

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

REPORT DOCUMENTATION PAGE	***Form Approved*** ***OMB No. 0704-0188***

1. REPORT DATE *(DD-MM-YYYY)* September 2019	**2. REPORT TYPE** Technical Report	**3. DATES COVERED (From - To)** October 2017-September 2019
4. TITLE AND SUBTITLE Quantifying Uncertainties in Parameterizations of Strength Models of Rolled Homogeneous Armor: Part 1, Overview		**5a. CONTRACT NUMBER**
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) JJ Ramsey		**5d. PROJECT NUMBER**
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) US Army Combat Capabilities Development Command Army Research Laboratory ATTN: FCDD-RLC-NB Aberdeen Proving Ground, MD 21005-5066		**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-TR-8826
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
Guidance is provided on how to obtain uncertainties in parameters of material strength models of rolled homogeneous armor, along with point estimates for those parameters, using existing software tools to implement two different approaches: Bayesian regression and the interval predictor model (IPM) approach. This report shows how to mathematically describe a Bayesian model associated with a material strength model and related experimental data, and how to express this Bayesian model in forms that the aforementioned tools can process. It also describes how the IPM approach can be implemented in Python. The report shows how the model parameter uncertainties can be visualized and how they may be presented in a form suitable for input to software tools for uncertainty propagation analysis, such as Dakota. Finally, the report shows how Bayesian analysis may be used to evaluate the quality of the fit of a strength model to experimental data.

15. SUBJECT TERMS
uncertainty quantification, Bayesian analysis, Johnson-Cook, Zerilli-Armstrong, Stan, PyMC3, interval predictor model

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** James J Ramsey
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified	UU	140	**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

When it comes to uncertainty quantification, point estimates of model parameters are not good enough. One needs to have estimates of the uncertainties in the parameters of a model if one is going to determine uncertainties in the quantities of interest produced by that model. Yet in spite of this, published model parameters all too often lack associated uncertainties; for example, they are missing in parameterizations of strength models for rolled homogeneous armor (RHA).[1–3] Future researchers who do uncertainty propagation calculations may then be left to make educated guesses about the parameter uncertainties.[4] The point of this report and its two companion reports is to try to reverse this state of affairs by showing current and future researchers, especially at the CCDC Army Research Laboratory (ARL), how to use existing software tools to obtain parameter estimates *that include uncertainty*. These tools are used, not on a "toy" problem, but rather on the realistic problem of finding fitting parameters to strength models for RHA. Two approaches are used: Bayesian analysis via the packages Stan[5] and PyMC3,[6] and an approximate interval predictor model (IPM) approach[7,8] that can be solved using off-the-shelf linear programming tools from, for example, SciPy.[9]

This report is meant as an overview of how to use the aforementioned two approaches to fit a parameterized model to data, to ascertain the uncertainties in the model parameters and evaluate the quality of the fit of the model to the data. Its coverage of Bayesian software tools is mostly limited to a discussion of how to express a Bayesian model (discussed in Sections 2 and 5) in forms that these tools can accept. However, there are a few brief mentions of certain functions of these tools, and one may use them to assist in finding relevant documentation for how to execute a Bayesian analysis with said tools. A discussion of how to implement the approximate IPM approach is in Section 7. Those who want more details of how to implement these analyses may wish to consult at least one of the two companion technical reports[10,11]: one covers step by step a workflow using the R language[12] and relevant packages that that interface with it, such as the Bayesian tool RStan[13] and the linear programming package lpSolve[14] used to implement the IPM approach; and the other covers step by step a workflow that uses the Python language,[15] two Python-based Bayesian tools, PyStan[16] and PyMC3,[6] and the aforementioned SciPy to implement the IPM approach.

Excerpts of program code, variables, functions, and filenames are written in a fixed-width font `like this`. The lines in excerpts of program code files are also numbered.

2. Overview of Bayesian Analysis

When employing Bayesian analysis to estimate the uncertainties in strength model parameters, one treats the parameters as random variables and seeks the probability density function (PDF) of these parameters, given the model and the data at hand. (For more details on PDFs, see Appendix A.) To obtain the PDF of these model parameters, one uses Bayes' rule, which, for continuous random variables, takes the following form[17]:

$$p(\boldsymbol{\theta}|\mathbf{D}) = \frac{p(\mathbf{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{\int_{\mathbb{R}^{n_p}} p(\mathbf{D}|\boldsymbol{\theta}^*)p(\boldsymbol{\theta}^*)d\boldsymbol{\theta}^*} \tag{1}$$

Here, $\boldsymbol{\theta}$ represents a vector of n_p model parameters, where $n_p \geq 1$, and $\mathbf{D}$ represents a known quantity on which $\boldsymbol{\theta}$ is supposed to depend, usually some set of experimental data. The symbol $\mathbb{R}^{n_p}$ represents all possible vectors of length n_p. Three particular expressions merit particular mention:

- $p(\boldsymbol{\theta})$, the PDF of what is called the *prior distribution*, or often just the prior, which represents a possibly rough estimate of what model parameter values may be more or less likely, without taking $\mathbf{D}$ fully into account. Whether the expression $p(x)$ indicates a prior for x or just a PDF of x in general depends on context.

- $p(\mathbf{D}|\boldsymbol{\theta})$, the PDF of the *likelihood*, which represents how likely $\mathbf{D}$ would be what it is, given a particular value for the model parameter vector $\boldsymbol{\theta}$.

- $p(\boldsymbol{\theta}|\mathbf{D})$, the PDF of what is called the *posterior distribution*, or often just the posterior, which is the PDF of $\boldsymbol{\theta}$ once $\mathbf{D}$ has been taken into account. This is the primary output of Bayesian analysis.

In this report, the prior and likelihood together are described as a *Bayesian model*. This is distinct from a model used for prediction, such as a strength model, though such a model is necessarily a part of a Bayesian model. Indeed, the model parameter vector $\boldsymbol{\theta}$ is technically a vector of the parameters of the overall Bayesian model, of which the parameters of the predictive part of the model (e.g., a strength model) are a subset. This is elaborated in the discussion of the likelihood later on.

The prior PDF is often assumed to be the product of its marginal PDFs[*], that is, $p(\boldsymbol{\theta}) = \prod_{i=1}^{n_p} p(\theta_i)$. The marginal PDF of the prior for a particular model parameter may be *noninformative*, indicating complete or nearly complete ignorance of the likely values of a model parameter. Such a prior is typically flat, that is, $p(\theta_i)$ is the same for all possible values of θ_i, and also improper (i.e., $\int_{-\infty}^{\infty} p(\theta_i^*) d\theta_i^* \to \infty$).[18] Alternatively, the marginal prior PDF $p(\theta_i)$ may also be *weakly informative*. By definition, such a prior PDF is proper (i.e., $\int_{-\infty}^{\infty} p(\theta_i^*) d\theta_i^* = 1$), and it also tends to be wide or fat, indicating that it represents a rough order-of-magnitude estimate of a parameter's possible values.[18] A sufficiently narrow marginal prior PDF may be considered *strongly informative*, with the sharp peak of the PDF being indicative of a parameter's likely value given previous experiments, theory, and so on. The overall prior $p(\boldsymbol{\theta})$ may be strongly informative for a particular parameter, while being weakly informative for the rest of the model parameters. However, if the predictive part of the model imposes correlations among the model parameters, then a strong marginal prior for one parameter may not only affect the locations of the peaks of the marginal posterior PDFs of the other parameters, but may lead to narrower peaks for those PDFs as well. Strongly informative priors should be used with caution. If they are based on misinformation, and there is not enough data in **D** to contradict this misinformation, then the resulting posterior will be misleading.

The likelihood, in practice, often accounts for the discrepancy between the predictions of a model and the experimental data. For example, there may be a strength model $\sigma_{mdl}(\mathbf{e}, \boldsymbol{\theta}_{mdl})$ that predicts the flow stress given a material state $\mathbf{e}$ (which may be a combination of, for instance, plastic strain, strain rate, and temperature) and a parameter vector $\boldsymbol{\theta}_{mdl}$, whose components are a subset of the components of $\boldsymbol{\theta}$. This model is such that when **D** consists of a set of N experimental inputs $\{\mathbf{e}_i\}$, $i \in [1, N]$, along with the corresponding set of experimental outputs $\{\sigma_i\}$, then $\sigma_{mdl}(\mathbf{e}_i, \boldsymbol{\theta}_{mdl})$ is an imperfect predictor of σ_i. The experimental outputs may be thought of as samples of random variables represented by the following sampling statement:

$$\{\sigma_i\} \sim \mathcal{D}_{lik}(\sigma_{mdl}, \{\mathbf{e}_i\}, \boldsymbol{\theta}_{mdl}, \boldsymbol{\theta}_{err}) \tag{2}$$

[*]Appendix A defines what a marginal PDF is; see Eq. A-3.

The "$\sim$" operator indicates that the set $\{\sigma_i\}$ is assumed to have been drawn from a random distribution $\mathcal{D}_{lik}$ that models the analyst's assumptions of how the set of model predictions $\{\sigma_{mdl}(\mathbf{e}_i, \boldsymbol{\theta}_{mdl})\}$ departs from reality. This distribution may depend not only on the model predictions but directly upon $\{\mathbf{e}_i\}$ itself (as, for example, in the work of Kennedy and O'Hagan[19]). The vector $\boldsymbol{\theta}_{err}$ contains any parameters of the distribution $\mathcal{D}_{lik}$ that characterize noise and errors in the experiment and possibly errors in the model as well.* The components of this vector are the components of $\boldsymbol{\theta}$ that are not components of $\boldsymbol{\theta}_{mdl}$. If $p_{\mathcal{D}_{lik}}$ is the PDF of $\mathcal{D}_{lik}$, then

$$p(\mathbf{D}|\boldsymbol{\theta}) = p(\{\mathbf{e}_i\}, \{\sigma_i\} \mid \boldsymbol{\theta}_{mdl}, \boldsymbol{\theta}_{err}) = p_{\mathcal{D}_{lik}}(\{\sigma_i\} \mid \sigma_{mdl}, \{\mathbf{e}_i\}, \boldsymbol{\theta}_{mdl}, \boldsymbol{\theta}_{err}) \tag{3}$$

where the arguments of $p_{\mathcal{D}_{lik}}$ after "|" are parameters of $\mathcal{D}_{lik}$. The components of $\boldsymbol{\theta}_{err}$ are often called *nuisance parameters*, since they are needed for analysis but are not inputs to the predictive model itself.[18]

From the posterior distribution itself, one can obtain both point estimates of strength model parameters and measures of their uncertainties. From a combination of the posterior and the likelihood, one may obtain the *posterior predictive distribution* (PPD), which can be used to check how well a model's predictions agree with the data.[18] The PPD may be summarized by the following sampling statement,

$$\boldsymbol{\Sigma}^{pred}(\{\mathbf{e}_i\}) \sim \mathcal{D}_{lik}(\sigma_{mdl}, \{\mathbf{e}_i\}, \boldsymbol{\theta}_{mdl}, \boldsymbol{\theta}_{err}), \text{ if } \boldsymbol{\theta} \sim \mathcal{D}_{post} \tag{4}$$

where $\boldsymbol{\Sigma}^{pred}(\{\mathbf{e}_i\})$ is a random set of possible experimental outputs for a given set of experimental inputs $\{\mathbf{e}_i\}$, provided that σ_{mdl} represents the actual behavior of the experimental system; and $\mathcal{D}_{post}$ is the posterior distribution, which again has PDF $p(\boldsymbol{\theta}|\mathbf{D})$. Equation 4 implies that a sample from the PPD is obtained by sampling $\boldsymbol{\theta}$ (i.e., $\boldsymbol{\theta}_{mdl}$ and $\boldsymbol{\theta}_{err}$) from the posterior distribution, substituting that into the likelihood $\mathcal{D}_{lik}(\ldots)$, and then sampling from the likelihood. The PPD includes the effects of nuisance parameters. However, when one inputs PDFs of parameters into tools for uncertainty propagation analyses, one generally does not input the PDFs of the nuisance parameters. To see the effects of the uncertainties in the predictive model parameters alone, one may use the *pushed forward posterior* (PFP),[21] which

*It is possible to construct $\mathcal{D}_{lik}$ such that error is taken into account through $\boldsymbol{\theta}_{mdl}$ alone.[20] In that case, $\boldsymbol{\theta}_{err}$ is an empty, zero-length vector.

may be represented by the sampling statement

$$\mathbf{\Sigma}^{pfp}(\mathbf{e}_i) \sim \sigma_{mdl}(\mathbf{e}_i, \boldsymbol{\theta}_{mdl}), \text{ if } \boldsymbol{\theta}_{mdl} \sim \mathcal{D}_{post} \tag{5}$$

This statement implies that a sample of the PFP is obtained by sampling the predictive model parameters $\boldsymbol{\theta}_{mdl}$ from the posterior distribution, and substituting that into the predictive model $\sigma_{mdl}(\mathbf{e}_i, \boldsymbol{\theta}_{mdl})$. Ideally, a sufficiently large number of samples from the PPD should form a pattern that resembles the experimental outputs $\{\sigma_i\}$. Similarly, a sufficiently large number of samples from the PFP ideally should resemble what the experimental outputs would look like if there were no error. The extent to which these ideals hold indicates the level of accuracy of the assumptions used to construct the likelihood, such as the choice of σ_{mdl} and $\mathcal{D}_{lik}$.

The posterior PDF is often difficult or impossible to obtain analytically, so in practice it is estimated numerically via various algorithms collectively called Markov Chain Monte Carlo (MCMC). Briefly, these algorithms take the likelihood as input, may either take priors as input or assume noninformative priors, and produce as output what are called *chains*, sequences of random samples from what is supposed to be the posterior distribution. These samples can then be postprocessed to obtain information about the posterior and related quantities such as the PPD and PFP. The particular software tools mentioned in Section 1, Stan and PyMC3, implement a form of MCMC called Hamiltonian Monte Carlo (HMC),[22] which uses gradients of the logarithms of the likelihood and prior to sample posterior distributions that would be more difficult for MCMC methods that do not use gradients to sample, and the no U-turn sampler (NUTS),[23] which automatically and adaptively sets the parameters for HMC that would otherwise need to be manually set by the software user. For those interested in more details of MCMC algorithms, one may consult Kruschke,[17] Smith,[24] Gelman et al.,[18] and Betancourt,[22] as well as other works.

3. Overview of Approximate Interval Predictor Model Approach

An IPM[7,8] is simply a function that returns an interval as its output rather than a single value. For example, given a function to predict the flow stress, $\sigma_{mdl}(\mathbf{e}, \boldsymbol{\theta}_{mdl})$, and a set $\mathbf{\Theta}$, the interval within which the flow stress is estimated to lie is

$[\sigma_{min}(\mathbf{e};\boldsymbol{\Theta}), \sigma_{max}(\mathbf{e};\boldsymbol{\Theta})]$, where

$$\sigma_{min}(\mathbf{e};\boldsymbol{\Theta}) = \min_{\boldsymbol{\theta}_{mdl}\in\boldsymbol{\Theta}} \sigma_{mdl}(\mathbf{e},\boldsymbol{\theta}_{mdl}) \tag{6}$$

$$\sigma_{max}(\mathbf{e};\boldsymbol{\Theta}) = \max_{\boldsymbol{\theta}_{mdl}\in\boldsymbol{\Theta}} \sigma_{mdl}(\mathbf{e},\boldsymbol{\theta}_{mdl}) \tag{7}$$

The set $\boldsymbol{\Theta}$ is chosen so as to keep the intervals from the IPM reasonably tight, given known data points $\{\mathbf{e}_i, \sigma_i\}$. For example, $\boldsymbol{\Theta}$ may be chosen such that

$$\boldsymbol{\Theta} = \arg\min_{\boldsymbol{\Theta}'} \frac{1}{N}\sum_{i=1}^{N} [\sigma_{max}(\mathbf{e}_i;\boldsymbol{\Theta}') - \sigma_{min}(\mathbf{e}_i;\boldsymbol{\Theta}')] \tag{8}$$

The minimization of Eq. 8 under the constraint

$$\sigma_{min}(\mathbf{e}_i;\boldsymbol{\Theta}) \le \sigma_i \le \sigma_{max}(\mathbf{e}_i;\boldsymbol{\Theta}), \forall i \in [1, N] \tag{9}$$

may not be tractable, especially if there is no analytical solution to Eqs. 6 and 7, thus requiring a nested optimization (i.e., at each iteration to solve Eq. 8, optimization routines would need to be used to estimate σ_{min} and σ_{max} for each data point). However, one may obtain a more tractable problem by approximating $\sigma_{mdl}(\mathbf{e},\boldsymbol{\theta}_{mdl})$ with a first-order Taylor expansion about a point estimate of $\boldsymbol{\theta}_{mdl}$, $\boldsymbol{\theta}_0$, and taking $\boldsymbol{\Theta}$ to be a hyperrectangle with corners $\boldsymbol{\theta}_0 - \Delta\boldsymbol{\theta}_{min}$ and $\boldsymbol{\theta}_0 + \Delta\boldsymbol{\theta}_{max}$. If $\mathbf{g}_{\sigma_{mdl}}(\mathbf{e})$ is the gradient of $\sigma_{mdl}(\dots)$ with respect to $\boldsymbol{\theta}_{mdl}$ evaluated at $\mathbf{e}$ and $\boldsymbol{\theta}_0$, and $|\mathbf{g}_{\sigma_{mdl}}(\mathbf{e})|$ is the *elementwise* absolute value of $\mathbf{g}_{\sigma_{mdl}}(\mathbf{e})$, then Eqs. 6 and 7 can be approximated as follows:

$$\begin{aligned}\sigma_{min}(\mathbf{e};\boldsymbol{\Theta}) \approx \sigma_{mdl}(\mathbf{e},\boldsymbol{\theta}_0) &- \frac{1}{2}\left(\mathbf{g}_{\sigma_{mdl}}(\mathbf{e}) + |\mathbf{g}_{\sigma_{mdl}}(\mathbf{e})|\right)^T \Delta\boldsymbol{\theta}_{min} \\ &+ \frac{1}{2}\left(\mathbf{g}_{\sigma_{mdl}}(\mathbf{e}) - |\mathbf{g}_{\sigma_{mdl}}(\mathbf{e})|\right)^T \Delta\boldsymbol{\theta}_{max}\end{aligned} \tag{10}$$

$$\begin{aligned}\sigma_{max}(\mathbf{e};\boldsymbol{\Theta}) \approx \sigma_{mdl}(\mathbf{e},\boldsymbol{\theta}_0) &- \frac{1}{2}\left(\mathbf{g}_{\sigma_{mdl}}(\mathbf{e}) - |\mathbf{g}_{\sigma_{mdl}}(\mathbf{e})|\right)^T \Delta\boldsymbol{\theta}_{min} \\ &+ \frac{1}{2}\left(\mathbf{g}_{\sigma_{mdl}}(\mathbf{e}) + |\mathbf{g}_{\sigma_{mdl}}(\mathbf{e})|\right)^T \Delta\boldsymbol{\theta}_{max}\end{aligned} \tag{11}$$

Here, a superscript T indicates the transpose. Given Eqs. 10 and 11 along with a fixed $\boldsymbol{\theta}_0$, Eq. 8 becomes

$$\Delta\boldsymbol{\theta}_{min}, \Delta\boldsymbol{\theta}_{max} = \underset{\Delta\boldsymbol{\theta}'_{min}, \Delta\boldsymbol{\theta}'_{max}}{\arg\min} \frac{1}{N}\left[\sum_{i=1}^{N} |\mathbf{g}_{\sigma_{mdl}}(\mathbf{e}_i)|\right]^T (\Delta\boldsymbol{\theta}'_{min} + \Delta\boldsymbol{\theta}'_{max}) \tag{12}$$

Together, Eqs. 9–12 form a constrained minimization problem that can be solved through linear programming.

The previously described approach for estimating $\boldsymbol{\Theta}$ does not guarantee that it does not contain invalid values of $\boldsymbol{\theta}_{mdl}$. For example, even though the elements of $\boldsymbol{\theta}_{mdl}$ must be nonnegative, $\boldsymbol{\theta}_0 - \Delta\boldsymbol{\theta}_{min}$ may have negative elements. There are two possible approaches to remedy this. One of these is to use transformed parameters, so for example, if a strength model takes nonnegative parameters, then $\boldsymbol{\theta}_{mdl}$ represents the natural logarithm of the parameters, and $\sigma_{mdl}(\mathbf{e}, \boldsymbol{\theta}_{mdl}) = \sigma'_{mdl}(\mathbf{e}, \exp(\boldsymbol{\theta}_{mdl}))$, where the exponential function is taken to operate elementwise. Another approach is to simply truncate the bounds of $\boldsymbol{\Theta}$, so for example, if the estimated value of $\Delta\boldsymbol{\theta}_{min}$ is $\mathbf{a} + \boldsymbol{\delta}$, where $\boldsymbol{\delta}$ causes $\boldsymbol{\Theta}$ to contain invalid values, then $\Delta\boldsymbol{\theta}_{min}$ can be taken to be just $\mathbf{a}$. While this approach is more simplistic, it can still lead to reasonable results if the elements of $\boldsymbol{\delta}$ are small.

4. Data for Analyses

The stress-strain data to be used in Bayesian analysis of strength models comes from the Material Implementation, Database, and Analysis Source (MIDAS).[25] Tables of this stress-strain data, along with some of the details of how they have been obtained, are in Appendix B. The data can be divided into n_c subsets, where subset i_c ($i_c \in [1, n_c]$) is associated with a plastic strain rate $\dot{\epsilon}_p^{i_c}$ and an initial temperature $T_{init}^{i_c}$ of the sample being deformed. A graph of the data is shown in Fig. 1, with each subset plotted as a curve. Examination of these curves indicates that they can be grouped into two categories, one for relatively smooth curves, which are associated with strain rates no greater than 1/s, and one for rougher, wavier curves, associated with higher strain rates. As pointed out in Appendix B, the low-strain-rate data and high-strain-rate data have been obtained with different instruments, which may be taken into account when fitting strength models to them.

At first, it may seem that when the stress-strain data are input to a computational Bayesian analysis, far less storage should be needed for the strain rate and temper-

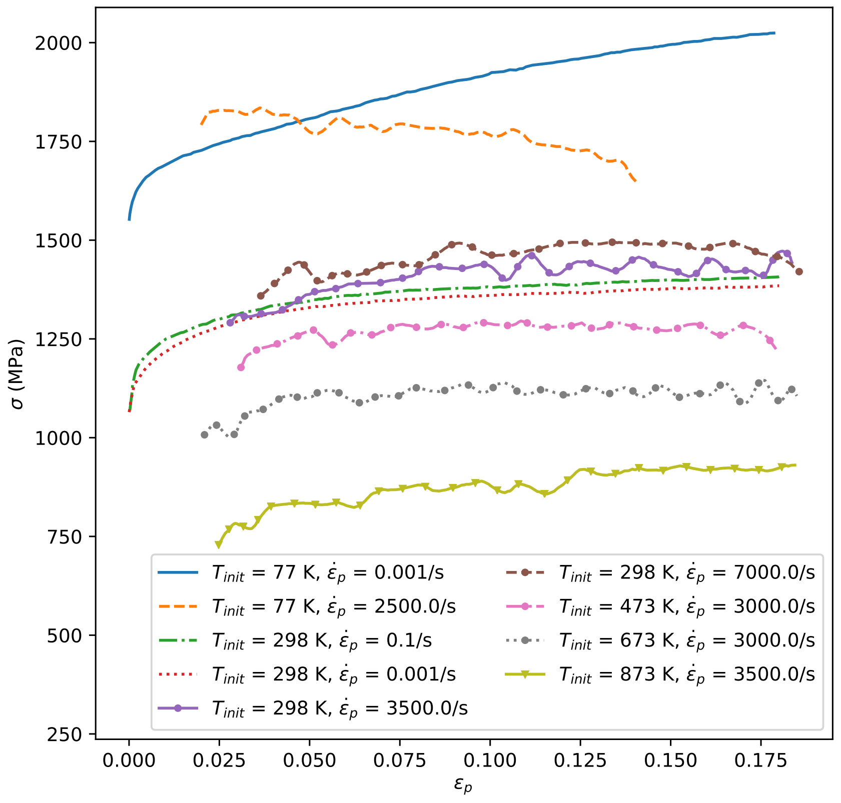


Fig. 1 Plots of flow stress σ vs. plastic strain ϵ_p for RHA from MIDAS, with the plastic strain rate denoted as $\dot{\epsilon}_p$ and the initial sample temperature T_{init}

ature than for the stresses and strains, since only one strain rate and temperature would be needed for each stress-strain curve. However, when a sample is deformed at a high strain rate, there is no time for the heat generated from plastic work to dissipate from the sample while it deforms. Accordingly, during deformation, the temperature of the sample rises from its initial value. Ideally, the best way to account for this temperature rise due to the buildup of heat would be to measure the temperature of the sample as it deforms (as done, for example, in Walley et al.[26]). However, this has not been done for the stress-strain data under consideration here. Instead, the temperature rise ΔT as the sample deforms is estimated as follows[27–29]:

$$\Delta T = T_j^{i_c} - T_{j-1}^{i_c} \approx \frac{\beta_{TQ}}{\rho c(T_{j-1}^{i_c})} \int_{\epsilon_{p,j-1}^{i_c}}^{\epsilon_{p,j}^{i_c}} \sigma d\epsilon_p \tag{13}$$

Here, $T_j^{i_c}$ and $\epsilon_{p,j}^{i_c}$ are the temperature and plastic strain of data point j in subset i_c, β_{TQ} is the Taylor-Quinney coefficient, ρ is the density, and $c(T)$ is the specific heat, which is a function of temperature T. The integral in Eq. 13 is the area under the portion of stress-strain curve i_c that is over the strain interval $[\epsilon_{p,j-1}^{i_c}, \epsilon_{p,j}^{i_c}]$. In this equation, β_{TQ} indicates the fraction of plastic work converted to heat, so its maximum value is 1.[29] The density is taken to be 7840 kg/m^3.[30] Specific heat values of RHA do not appear to be readily available, but since the equations of state for RHA and iron do not appear to be significantly different,[31] specific heat values of body-centered cubic (BCC) iron,[32] shown in Appendix B, are used instead. Linear interpolation is used to estimate $c(T)$ for temperature values not given in Table B-1. Equation 13 treats the specific heat as approximately constant over the temperature rise ΔT.

If the temperature associated with the first point of curve i_c, $T_1^{i_c}$, were equal to $T_{init}^{i_c}$, calculating the set of temperatures $\{T_j^{i_c}\}$ from the temperature rise would be straightforward. However, the initial temperature is for an *unstrained* sample, and for high strain rates, the plastic strain $\epsilon_{p,1}^{i_c}$ is not 0 but some small finite value, usually around 2.5%. At this finite strain, the temperature has already increased from $T_{init}^{i_c}$. To approximately account for this, one can note two things illustrated in Fig. 2, which shows an example stress-strain curve starting at $(\epsilon_{p,1}^{i_c}, \sigma_1^{i_c})$. The part of the curve over the interval $[0, \epsilon_{p,1}^{i_c}]$ is essentially missing. It is unlikely that the curve is above the horizontal line $\sigma = \sigma_1^{i_c}$, so the area of the shaded rectangle in the figure, $\sigma_1^{i_c} \epsilon_{p,1}^{i_c}$, is a likely overestimate of the area under the missing part of the stress-strain curve. On the other hand, the area under the line from the origin to $(\epsilon_{p,1}^{i_c}, \sigma_1^{i_c})$, $0.5\sigma_1^{i_c}\epsilon_{p,1}^{i_c}$, is a likely underestimate, because the intercept of the stress-strain curve with the σ-axis is not zero but rather the initial yield stress for the temperature and strain rate that is associated with the curve. Accordingly, an approximation of the temperature rise due to this missing part of the curve then is

$$T_1^{i_c} - T_{init}^{i_c} \approx \frac{\beta_{TQ}}{\rho c(T_{init}^{i_c})} f_{area}\sigma_1^{i_c}\epsilon_{p,1}^{i_c}, f_{area} \in [0.5, 1] \tag{14}$$

assuming that $c(T)$ is approximately constant over the temperature range $[T_{init}^{i_c}, T_1^{i_c}]$.

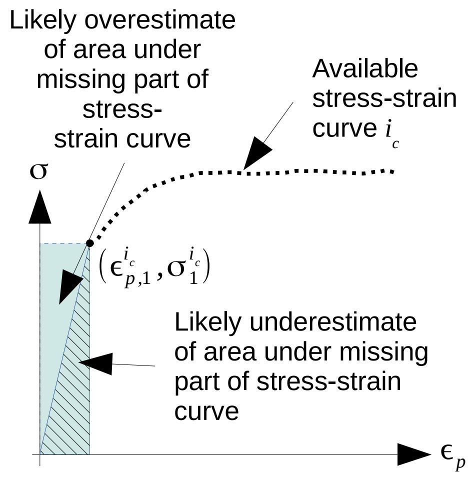

Likely overestimate of area under missing part of stress-strain curve
Available stress-strain curve i_c
σ
$(\epsilon_{p,1}^{i_c}, \sigma_1^{i_c})$
Likely underestimate of area under missing part of stress-strain curve
ϵ_p

Fig. 2 Example stress-strain curve where the available data points start at $(\epsilon_{p,1}^{i_c}, \sigma_1^{i_c})$, so that the part of the curve over the interval $[0, \epsilon_{p,1}^{i_c}]$ is missing. The shaded rectangle, with area $\sigma_1^{i_c}\epsilon_{p,1}^{i_c}$, is a likely overestimate of the area under the missing part of the stress-strain curve. The hatched triangle, with area $\sigma_1^{i_c}\epsilon_{p,1}^{i_c}/2$, is a likely underestimate of the area.

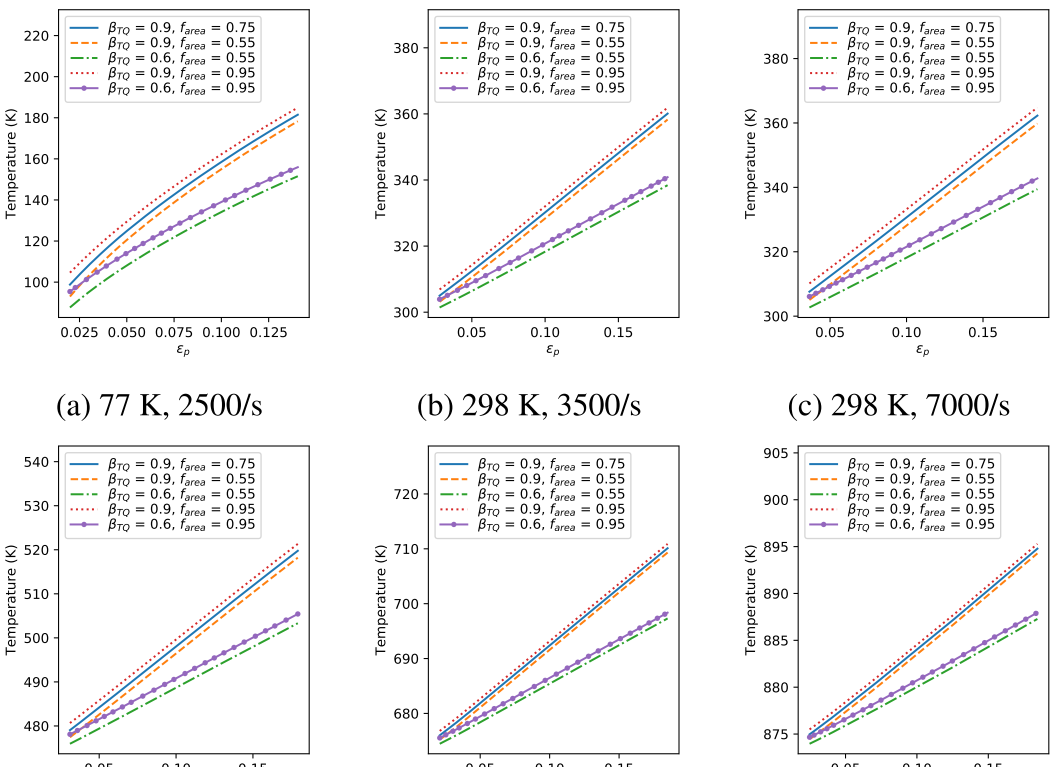
While β_{TQ} is often taken to be equal to 0.9 for metals, there is a wide spread of values found in the literature, with β_{TQ} sometimes found to be as low as 0.4.[29] Estimation of f_{area} amounts to educated guesswork. Accordingly, temperatures are estimated for a few combinations of reasonable estimates of β_{TQ} and f_{area}, shown in Table 1. Plots of these estimated temperatures are shown in Fig. 3.

5. Constructing Bayesian Models

To construct an Bayesian model to fit a strength model, one needs a likelihood and a prior. To determine a prior, one of course needs to know the plausible ranges of values of strength model parameters, while the likelihood needs the predictions of a

Table 1 Possible combinations of values of β_{TQ} and f_{area} used in temperature estimation

β_{TQ}	f_{area}
0.9	0.75
0.9	0.55
0.6	0.55
0.9	0.95
0.6	0.95



(a) 77 K, 2500/s (b) 298 K, 3500/s (c) 298 K, 7000/s

(c) 473 K, 3000/s (d) 673 K, 3000/s (e) 873 K, 3500/s

Fig. 3 Estimated temperatures along stress-strain curves with the initial temperatures and strain rates shown, given the values of β_{TQ} and f_{area} in Table 1

strength model as one of its inputs. Many strength models may be used to model the plastic behavior of RHA, but here, the focus is on two of them: one by Johnson and Cook[33] and one by Zerilli and Armstrong[34] that is specific to BCC materials. These two models are chosen because they have relatively simple closed forms and are available in Army-relevant codes such as CTH.[35] The Johnson-Cook model may be written as follows:

$$\sigma_{JC}(\epsilon_p, \dot{\epsilon}_p, T^*; \boldsymbol{\theta}_{JC}) = (A + B\epsilon_p^n)[1 + C\ln(\dot{\epsilon}_p/\dot{\epsilon}_{p0})][1 - (T^*)^m] \tag{15}$$

$$T^* = (T - T_{room})/(T_{melt} - T_{room}) \tag{16}$$

Here, σ_{JC} is the flow stress according to the Johnson-Cook model, ϵ_p is the plastic strain, $\dot{\epsilon}_p$ is the plastic strain rate, $\dot{\epsilon}_{p0} = 1/\mathrm{s}$, T is the temperature, T_{room} is the room temperature, T_{melt} is the melting temperature, A, B, n, C, and m are fitting parameters, and $\boldsymbol{\theta}_{JC} = (A, B, n, C, m)$. Following Gray et al.,[1] T_{room} and T_{melt} are taken to be 298 and 1783 K, respectively. Since the Johnson-Cook model cannot be applied where the temperature is below T_{room}, only stress-strain data for temperatures of T_{room} and above are used with it.

The Zerilli-Armstrong model for BCC materials may be written as follows:

$$\sigma_{ZA,BCC}(\epsilon_p, \dot{\epsilon}_p, T; \boldsymbol{\theta}_{ZA,BCC}) = C_0 + C_1 \exp[(-C_3 + C_4 \ln(\dot{\epsilon}_p/\dot{\epsilon}_{p0}))T] + C_5\epsilon_p^n \tag{17}$$

Here, $\sigma_{ZA,BCC}$ is the flow stress according to the Zerilli-Armstrong (BCC) model; ϵ_p, $\dot{\epsilon}_p$, and T are again the plastic strain, plastic strain rate, and temperature, respectively; $\dot{\epsilon}_{p0} = 1/\mathrm{s}$; C_0, C_1, C_3, C_4, C_5, and n are fitting parameters; and $\boldsymbol{\theta}_{ZA,BCC} = (C_0, C_1, C_3, C_4, C_5, n)$. (There is no parameter C_2; such a parameter belongs to the face-centered cubic version of the Zerilli-Armstrong model.[1])

To construct a likelihood, one needs a model of the discrepancy between the model predictions and the experimental data. Conceptually, this may be modeled as follows[19]:

$$\{\sigma_j^{i_c}\} = \sigma_{mdl}(\{\mathbf{e}_j^{i_c}\}, \boldsymbol{\theta}_{mdl}) + e(\mathbf{D}, \boldsymbol{\theta}_{err}^e) + \delta(\mathbf{D}, \boldsymbol{\theta}_{err}^\delta) \tag{18}$$

Following similar notation in Section 4, the vector of experimental stress values is written as $\{\sigma_j^{i_c}\}$, where $j \in [1, N_{i_c}]$ and N_{i_c} is the number of data points in the subset associated with stress-strain curve i_c. Given the constitutive models described in Eqs. 15 and 17, "mdl" now stands in for "JC" or "ZA, BCC," and $\mathbf{e}_j^{i_c}$ represents the combination of plastic strain $\epsilon_{p,j}^{i_c}$, plastic strain rate $\dot{\epsilon}_p^{i_c}$, and temperature $T_j^{i_c}$. The vector $\sigma_{mdl}(\{\mathbf{e}_j^{i_c}\}, \boldsymbol{\theta}_{mdl})$ consists of predicted stress values, one for each element of $\{\mathbf{e}_j^{i_c}\}$. Concatenating vectors $\boldsymbol{\theta}_{err}^e$ and $\boldsymbol{\theta}_{err}^\delta$ yields $\boldsymbol{\theta}_{err}$. Here, $e(\mathbf{D}, \boldsymbol{\theta}_{err}^e)$ is a random function that represents noise from experimental instruments, while $\delta(\mathbf{D}, \boldsymbol{\theta}_{err}^\delta)$ is a smoother random function that represents more systematic departures between the model predictions and real-world behavior, i.e., model inadequacy. While this may be a reasonable conceptual depiction of model error, it can have significant practical problems, because for almost any value of $\boldsymbol{\theta}_{mdl}$, one can construct a function $\delta(\mathbf{D}, \boldsymbol{\theta}_{err}^e)$ that makes up the difference between $\{\sigma_j^{i_c}\}$ and $\sigma_{mdl}(\{\mathbf{e}_j^{i_c}\}, \boldsymbol{\theta}_{mdl})$. Furthermore, estimating $\boldsymbol{\theta}_{err}^\delta$ can be expensive. For example, if $\delta(\dots)$ is taken to be a Gaussian process, then it has a covariance matrix of size $N \times N$, where $N = \prod_{i_c=1}^{n_c} N_{i_c}$. This matrix requires $O(N^2)$ storage and requires $O(N^3)$ time to invert, and the inversion has to be done at least once per MCMC iteration. Because of these issues, an explicit representation of model inadequacy is avoided in the rest of this report. A likelihood that neglects model inadequacy is at least a useful starting point, and the results of proceeding with it can be used to indicate if some alternate likelihood may be advisable. Interested readers who wish to delve more deeply into representations of model inadequacy may wish to consult Kennedy and O'Hagan,[19] Ling et al.,[36] or Sargsyan et al.[20]

When model inadequacy is neglected and discrepancies between model predictions and data are attributed solely to measurement noise, the likelihood PDF may be simplified. If the noise in the measurement of one flow stress value is independent of the noise in the measurement of another, then the PDF of the set of stress values is the product of the PDFs of each individual value, so the overall likelihood PDF may be decomposed as follows:

$$\begin{aligned} p(\mathbf{D}|\boldsymbol{\theta}) &= p_{\mathcal{D}_{lik}}(\{\sigma_j^{i_c}\} \mid \sigma_{mdl}, \{\mathbf{e}_j^{i_c}\}, \boldsymbol{\theta}_{mdl}, \boldsymbol{\theta}_{err}) \\ &= \prod_{i_c=1}^{n_c} \prod_{j=1}^{N_{i_c}} p(\sigma_j^{i_c} | \sigma_{mdl}, \mathbf{e}_j^{i_c}, \boldsymbol{\theta}_{mdl}, \boldsymbol{\theta}_{err}) \end{aligned} \tag{19}$$

The random noise in an experimental output may be approximated as being normally distributed and centered about the model prediction for that output, such that

$$p(\sigma_j^{i_c} | \sigma_{mdl}, \mathbf{e}_j^{i_c}, \boldsymbol{\theta}_{mdl}, \boldsymbol{\theta}_{err}) = p_{\text{normal}}(\sigma_j^{i_c} | \sigma_{mdl}(\mathbf{e}_j^{i_c}, \boldsymbol{\theta}_{mdl}), SD_\sigma) \tag{20}$$

where $p_{\text{normal}}(\dots)$ is the PDF of a normal distribution with mean $\sigma_{mdl}(\mathbf{e}_j^{i_c}, \boldsymbol{\theta}_{mdl})$ and standard deviation SD_σ. (See Appendix A for more discussion of normal distributions.) Of course, SD_σ is meant to indicate how much noise is in a measurement source. For this particular problem, the choice of measurement source depends on the strain rate, so SD_σ depends on it as well. Here, SD_σ is assumed to be a constant for each source, so that

$$SD_\sigma(\dot{\epsilon}_p^{i_c}) = \begin{cases} SD_{\sigma,1} & \dot{\epsilon}_p^{i_c} \leq 1.0/\text{s} \\ SD_{\sigma,2} & \text{otherwise} \end{cases} \tag{21}$$

If the noise in the measurement sources were well known, then $SD_{\sigma,1}$ and $SD_{\sigma,2}$ could be treated as known quantities, and $\boldsymbol{\theta}_{err}$ would be an empty, zero-length vector. However, here they are treated as nuisance parameters, that is, $\boldsymbol{\theta}_{err} = (SD_{\sigma,1}, SD_{\sigma,2})$.

If the assumption of negligible model inadequacy does not hold, then the resulting PDF of model parameters, $p(\boldsymbol{\theta}|\mathbf{D})$, will still tend to center around a value of $\boldsymbol{\theta}$ that minimizes the least-squares error (weighted by $SD_{\sigma,1}$ and $SD_{\sigma,2}$) as well as the discrepancy (or more precisely, the Kullback-Leibler divergence) between the approximate constructed likelihood and the actual random distribution that generated the experimental data.[37] Furthermore, if more experimental data were to be obtained, the PDFs of the strength model parameters would become narrower, regardless of how well the model tracks the trends in the experimental data. Accordingly, the width of these PDFs *cannot* be used to gauge the accuracy of the strength model. However, the average values of the nuisance parameters $SD_{\sigma,1}$ and $SD_{\sigma,2}$ do increase to accommodate discrepancies between model predictions and experimental results, so they do provide a rough estimate of the accuracy of the model.

The PDF of all the priors here is taken to be the product of the prior PDFs of the individual parameters, including nuisance parameters. For example, for the Johnson-Cook model, this is

$$p(\boldsymbol{\theta}_{JC}, SD_{\sigma,1}, SD_{\sigma,2}) = p(A)p(B)p(n)p(C)p(m)p(SD_{\sigma,1})p(SD_{\sigma,2}) \tag{22}$$

Since noninformative priors may lead to both instability in numerical analyses[38] and improper posteriors, they are not used in any analysis in this report. Instead, priors are mostly weakly informative.

For both the Johnson-Cook and Zerilli-Armstrong (BCC) models, the parameter n is the exponent for a power-law curve whose slope decreases with increasing ϵ_p; therefore, n is in the interval $[0, 1]$. The simplest choice of prior distribution for n, then, would be a uniform prior over that interval. However, for more flexibility, $p(n)$ is taken to be $p_{\text{beta}}(n|n_\alpha, n_\beta)$, the PDF of a beta probability distribution (described in Appendix A), which is also 0 outside of $[0, 1]$. The parameters of this distribution are taken to be $n_\alpha = n_\beta = 1.1$, leading to the PDF shown in Fig. 4a. This PDF distribution is still nearly as flat as a uniform prior, but is slightly more informative, indicating a random variable that is highly unlikely to have values at the extremes of the interval $[0, 1]$ but could readily be almost anything else in that interval.

The other model parameters are positive but have no other hard limits on their range of values. For simplicity, the prior distribution used for each of these parameters is a modified normal distribution that is centered around a point estimate of the parameter—which may be little more than a rough guess—and truncated so that its PDF is zero for negative parameter values, that is,[39]

$$\begin{aligned} p(a) &= p_{\text{normal}}^{T[0,\infty)}(a|a_{guess\ mean}, a_{guess\ sd}) \\ &= \begin{cases} \dfrac{p_{\text{normal}}(a|a_{guess\ mean}, a_{guess\ sd})}{\int_0^\infty p_{\text{normal}}(a^*|a_{guess\ mean}, a_{guess\ sd})da^*} & a \geq 0 \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{23}$$

where a stands in for any model parameter except n, and the expression $p_{\text{normal}}(a|a_{guess\ mean}, a_{guess\ sd})$ indicates the PDF of a normal distribution (described in Appendix A) with mean $a_{guess\ mean}$ and standard deviation $a_{guess\ sd}$. The superscript $T[0, \infty)$ indicates that the domain of the PDF is truncated to the interval $[0, \infty)$. The integral in Eq. 23 is a normalization factor that ensures that the PDF remains proper. When the prior for a is weakly informative, $a_{guess\ mean}$ is set to an order-of-magnitude estimate of that parameter. Most of these estimates are from previous model fits in Gray et al.[1] Initial order-of-magnitude estimates of parameters are shown in Table 2. For the prior to be weakly informative, $a_{guess\ sd}$ should be at least the same order of magnitude as $a_{guess\ mean}$. However, since the model parameters are presumed to be nonzero, $p(0)$ needs to at least be approximately 0. Accordingly, in a weakly informative prior for a, $a_{guess\ sd}$ is set to $a_{guess\ mean}/3$. A plot of the prior PDF $p(A)$, where $A_{guess\ mean}$ is set to the initial estimate for A in the aforementioned table and $A_{guess\ sd}$ is set to $A_{guess\ mean}/3 \approx 333$ MPa, is shown in

Fig. 4b.

Table 2 Initial rough estimates of model parameters

Parameter	Estimate
A (MPa)	1000
B (MPa)	1000
n	0.5
C	0.001
m	1
C_0 (MPa)	100
C_1 (MPa)	1000
C_3 (K^{-1})	0.001
C_4 (K^{-1})	0.00001
C_5 (MPa)	1000
$SD_{\sigma,1}$, $SD_{\sigma,2}$ (MPa)	100

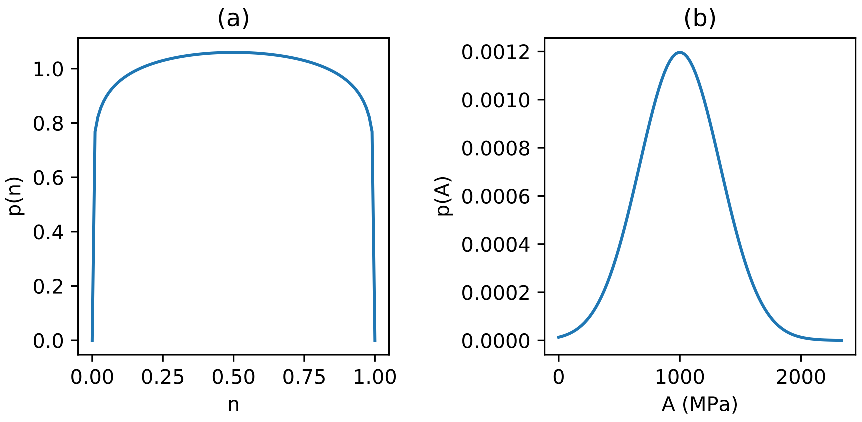


Fig. 4 Marginal prior PDFs for parameters n and A, where a) $p(n) = p_{\text{beta}}(n|1.1, 1.1)$ and b) $p(A) = p_{\text{normal}}^{T[0,\infty)}(A|1000, 1000/3)$

For parameter A of the Johnson-Cook model, a strongly informative prior is also available, because it represents an experimentally obtainable quasi-static yield stress. In Gray et al.,[1] the original source for the stress-strain data to which the Johnson-Cook and Zerilli-Armstrong (BCC) models are to be fit, samples of 5-inch-thick* RHA are taken perpendicular and parallel to the rolling direction of the armor plate. In Benck,[30] four samples of 4-inch RHA, also tested perpendicular and parallel

*According to Meyer and Kleponis,[2] the RHA used by Gray et al.[1] is 2 inches thick, but this appears to be a misreading of the test certificate in Fig. B-2 in Gray et al.,[1] which describes the plates as "`2 ⌐ 5 ⌐X⌐ 74 ⌐X⌐ 74`", indicating two 5-inch-thick square plates with a side length of 74 inches. A 2-inch thickness is inconsistent with the weight of each plate, 7765 lb, and a steel density of about 0.284 lb/in^2.

to the rolling direction, have measured yield strengths of 700.0, 702.0, 704.0, and 723.0 MPa. According to the specification for RHA,[40] plates of 4 to 6 inches should have the same range of hardness values, which suggests that the yield strengths of 4 and 5-inch plates should be about the same. This in turn suggests a value for $A_{guess\ mean}$ of 707.25 MPa, the mean of the measured yield strengths for the 4-inch-thick RHA, and a value for $A_{guess\ sd}$ of 10.63 MPa, the standard deviation of those measured yield strengths. The narrowness of this standard deviation, as compared with the standard deviation of about 333 MPa for the weakly informative prior for A, is what makes this prior strong.

A Bayesian model may be expressed in terms of sampling statements, and doing so simplifies the process of inputting the model into Bayesian software tools.[17,39,41] Sampling statements for the priors are as follows:

$$n \sim \text{beta}(n_\alpha, n_\beta) \tag{24}$$

$$A \sim \text{normal}(A_{guess\ mean}, A_{guess\ sd})_{T[0,\infty)} \tag{25}$$

$$B \sim \text{normal}(B_{guess\ mean}, B_{guess\ sd})_{T[0,\infty)} \tag{26}$$

$$C \sim \text{normal}(C_{guess\ mean}, C_{guess\ sd})_{T[0,\infty)} \tag{27}$$

$$m \sim \text{normal}(m_{guess\ mean}, m_{guess\ sd})_{T[0,\infty)} \tag{28}$$

$$C_k \sim \text{normal}(C_{k,guess\ mean}, C_{k,guess\ sd})_{T[0,\infty)} \tag{29}$$

$$SD_{\sigma,1} \sim \text{normal}(SD_{\sigma,1,guess\ mean}, SD_{\sigma,1,guess\ sd})_{T[0,\infty)} \tag{30}$$

$$SD_{\sigma,2} \sim \text{normal}(SD_{\sigma,2,guess\ mean}, SD_{\sigma,2,guess\ sd})_{T[0,\infty)} \tag{31}$$

where $k \in \{0, 1, 3, 4, 5\}$, $\text{beta}(n_\alpha, n_\beta)$ is the beta distribution with PDF $p(n) = p_{\text{beta}}(n|n_\alpha, n_\beta)$, $\text{normal}(a_{guess\ mean}, a_{guess\ sd})_{T[0,\infty)}$ is the truncated normal distribution with the PDF in Eq. 23, and the values of n_α, n_β, $A_{guess\ mean}$, $A_{guess\ sd}$, and so on, are as previously described. Because the likelihood PDF is assumed to decompose into a product of PDFs, one for each data point, a sampling statement can be associated with each data point (rather than having a sampling statement for the set of data points as a whole as in Eq. 2). Following Eq. 20, each of these sampling statements may be expressed as

$$\sigma_j^{i_c} \sim \text{normal}(\sigma_{mdl}(\mathbf{e}_j^{i_c}, \boldsymbol{\theta}_{mdl}), SD_{\sigma,k}) \tag{32}$$

where $k = 1$ if $\dot{\epsilon}_p^{i_c} \leq 1/\text{s}$, and $k = 2$ otherwise. For the Johnson-Cook and Zerilli-

Armstrong (BCC) models specifically, the sampling statements corresponding to their respective likelihoods are

$$\sigma_j^{i_c} \sim \text{normal}(\sigma_{JC}(\epsilon_{p,j}^{i_c}, \dot{\epsilon}_p^{i_c}, (T_j^{i_c})^*, \boldsymbol{\theta}_{JC}), SD_{\sigma,k}) \tag{33}$$

and

$$\sigma_j^{i_c} \sim \text{normal}(\sigma_{ZA,BCC}(\epsilon_{p,j}^{i_c}, \dot{\epsilon}_p^{i_c}, T_j^{i_c}, \boldsymbol{\theta}_{ZA,BCC}), SD_{\sigma,k}) \tag{34}$$

6. Specifying Bayesian Models for Software Tools

To use existing software tools to analyze Bayesian models, these models need to be translated from the mathematical notation seen in Sections 2 and 5 to a form that these software tools can use. The following sections discuss how this translation is done for Stan[39] and PyMC3.[6]

6.1 Specifying Models With Stan Specification Files

In order to use the various interfaces of Stan (such as RStan,[13] PyStan,[16] or CmdStan[42]), a specification of a Bayesian model should be written in plain text, preferably in a separate file,[43,44] using the syntax of the Stan language. This language currently consists of several *program blocks*,[*] each of which begins with a keyword or keyphrase followed by statements enclosed in braces.[39] These are the program blocks of the Stan specification files used in the analyses of this report:

- the `functions` block, which contains the definitions of one or more functions to be used in subsequent program blocks;
- the `data` block, which contains declarations of the input variables needed for the Bayesian model;
- the `transformed data` block, which contains declarations of variables that are functions of the variables from the `data` block;
- the `parameters` block, which contains declarations of the unknown model parameters to be found; and

[*]This may change in the future; see Carpenter.[45]

- the `model` block, which contains descriptions of the priors and likelihood of the model, in a syntax that resembles the sampling statements of Sections 2 and 5.

As an example, the program blocks of the specification of the Johnson-Cook model (shown in Appendix C) are shown and discussed in more detail.

The `functions` block is as follows.

```
 2  functions {
 3    vector jc(vector epsilon_p, real log_epsilon_p_dot, vector T_star,
 4              real A, real B, real n, real C, real m) {
 5
 6      int length_epsilon_p = num_elements(epsilon_p);
 7      vector[length_epsilon_p] sigma;
 8
 9      real edot_factor = (1.0 + C*log_epsilon_p_dot);
10
11      // The exponentiation operator "^" doesn't vectorize, so I need a
12      // "for" loop here.
13      for (i in 1:length_epsilon_p) {
14        sigma[i] = (A + B*(epsilon_p[i])^n)*edot_factor*
15          (1.0 - (T_star[i])^m);
16      }
17
18      return sigma;
19    }
20  }
```

This block defines the function `jc`, which specifies the Johnson-Cook strength model, σ_{JC}. Arguments to functions in Stan have *types*. The first argument, `epsilon_p`, of type `vector`, represents a sequence of plastic strain values ϵ_p, where `epsilon_p[1]` is the first value in the sequence, `epsilon_p[2]` is the second value, and so on. The integer value in the brackets is called an *index*. The second argument, `log_epsilon_p_dot`, of type `real`, represents a single scalar value for $\ln(\dot{\epsilon}_p/\dot{\epsilon}_{p0})$. In general, variables of type `real` represent double precision floating-point numbers. The argument `T_star` represents a sequence of values of T^*, and the arguments `A`, `B`, `n`, `C`, and `m` represent the fitting parameters of the Johnson-Cook model.

Lines 6–9, in the body of the function `jc`, are *variable declarations* in Stan. These indicate the names of variables that are used in the rest of the body of the function (i.e., `length_epsilon_p`, `sigma`) and `edot_factor`, the types of these variables, and they may also initialize the values of these variables. Here, `length_epsilon_p` is of type `int`, which indicates that it is a scalar integer value, and it is initialized to `num_elements(epsilon_p)`, which is the number of values in the vector `epsilon_p`. Like `epsilon_p`, `sigma` is of type `vector`. However, when declaring a variable of this type, the number of values stored in this variable must be given in brackets following the key word "`vector`". Accordingly, the declaration for the variable `sigma` starts with `vector[length_epsilon_p]`, indicating that the number of values stored in `sigma` is `length_epsilon_p`. The variable `edot_factor` is set to an arithmetic expression that corresponds to the factor $[1 + C\ln(\dot{\epsilon}_p/\dot{\epsilon}_{p0})]$ in the Johnson-Cook model. (In this expression, "`*`" is the multiplication operator, as it is in many programming languages.) Like all statements in Stan, the variable declarations are terminated by semicolons. Variable declarations are usually allowed in any block of statements enclosed by braces (e.g., a function body), but they must be at the top of the block, above any other statements.

The token "`//`" indicates the start of comment text. Everything from this token to the end of the line is ignored by a Stan implementation. (The comment text here indicates that the exponentiation operator "`^`" does not work with vectors.)

The expression `for(...)` indicates iteration, which means that the statements within the braces following `for(...)` (i.e., lines 14–15) are executed repeatedly. For each repetition (or iteration), the variable `i` takes on a different value, starting from 1 and continuing through 2, 3, and so on, all the way up to `length_epsilon_p`. This `for` loop specifies how element `i` of the sequence `sigma` is related to element `i` of the sequences `epsilon_p` and `T_star`, given certain values of the other arguments of the function `jc`. The expressions `(A + B*(epsilon_p[i])^n)`, `edot_factor`, and `(1.0 - (T_star[i])^m)` correspond to the factors $(A+B\epsilon_p^n)$, $[1+C\ln(\dot{\epsilon}_p/\dot{\epsilon}_{p0})]$, and $[1-(T^*)^m]$ in the Johnson-Cook model. One may note that the `for` loop could have been written without `edot_factor` as follows.

```
for (i in 1:length_epsilon_p) {
  sigma[i] = (A + B*(epsilon_p[i])^n)*(1.0 + C*log_epsilon_p_dot)*
    (1.0 - (T_star[i])^m);
}
```

However, since the value of the expression `1.0 + C*log_epsilon_p_dot` does not change in any iteration of the `for` loop, it is assigned to `edot_factor` in the last variable declaration above the loop, and then `edot_factor` is used in place of the expression. This avoids repeatedly calculating the expression unnecessarily.

Finally, the `return` statement in line 18 indicates the value or values that function `jc` returns when it is evaluated. In this case, what is returned is `sigma`, the vector of flow stress values as determined from the Johnson-Cook model.

The next block is the `data` block:

```
24 data {
25   int<lower=1> num_curves;
26   int<lower=0> curve_sizes[num_curves];
27   vector[num_curves] epsilon_p_dot;
28
29   vector[sum(curve_sizes)] epsilon_p;
30   vector[sum(curve_sizes)] sigma;
31   vector[sum(curve_sizes)] T;
32
33   real<lower=0.0> T_melt;
34   real<lower=0.0> T_room;
35
36   real<lower=0.0> epsilon_p_dot_0;
37
38   real<lower=0.0> A_guess_mean; real<lower=0.0> A_guess_sd;
39   real<lower=0.0> B_guess_mean; real<lower=0.0> B_guess_sd;
40   real<lower=0.0> C_guess_mean; real<lower=0.0> C_guess_sd;
41   real<lower=0.0> m_guess_mean; real<lower=0.0> m_guess_sd;
42
43   real<lower=0.0> n_alpha; real<lower=0.0> n_beta;
44
45   vector<lower=0.0>[2] sd_sigma_guess_mean;
46   vector<lower=0.0>[2] sd_sigma_guess_sd;
47 }
```

The `data` block is composed entirely of variable declarations, similar to the ones already seen in the body of the function `jc` shown previously. Within this block, one can use the notation `<lower=lo, upper=hi>` to indicate that a variable is constrained to be no lower than `lo` and no higher than `hi`. If the variable has no lower or upper bound, then either `lower=lo` or `upper=hi` is omitted. If at the end of a variable name in a declaration, there is an integer expression in brackets, such as "`[num_curves]`", that means the variable is an *array* variable. This means that `curve_sizes` (line 26) is such a variable. An array variable is similar to a variable of type `vector` in that it is a sequence of values, so that, for example, `curve_sizes[1]` is the first value, `curve_sizes[2]` is the second value, and so on. The number of values in this sequence is the value of the aforementioned integer expression in the brackets of the variable declaration, which for `curve_sizes` is just `num_curves`. There are several differences between array variables and vectors, and these are described in the Stan reference manual.[39] For the purposes of this report, two particular differences are noted:

- Arithmetic operators such as "+" and "$\star$" (but not "^"!) are defined for vectors but not arrays.
- A vector always contains a sequence of real numbers, whereas the values in arrays may be real numbers, integers, or even other Stan types such as vectors or matrices.

For an array or vector variable, like `curve_sizes` or `epsilon_p`, the constraints specified in the `<lower=lo, upper=hi>` notation apply to all values stored in the variable.

Here, `num_curves` is the number of stress-strain curves. The array `curve_sizes` indicates the number of data points in each stress-strain curve. Due to limitations in the types available in the Stan language, the data for strains, stresses, and temperatures are stored in vectors according to a scheme recommended in the Stan language manual for so-called ragged data structures,[39] which is illustrated in Fig. 5. The first `curve_sizes[1]` elements of `sigma`, `epsilon_p`, and `T` are the stress σ, plastic strain ϵ_p, and temperatures T for the first stress-strain curve, measured for strain rate `epsilon_p_dot[1]`, while the next `curve_sizes[2]` elements of `sigma`, `epsilon_p`, and `T` are the stress, plastic strain, and temperatures for the second stress-strain curve, measured for strain rate `epsilon_p_dot[2]`, and so on. Much of the notation for the variables in the `data` block is similar to the mathematical notation used for the Johnson-Cook model and its priors. For example, `T_melt` is T_{melt}, `epsilon_p_dot_0` is $\dot{\epsilon}_{p0}$ (where "`dot`" refers to the dot over the character ϵ), `A_guess_mean` is $A_{guess\ mean}$, and so on.

The next block is the `transformed data` block:

```
transformed data {
  vector[num_curves] log_epsilon_p_dot = log(epsilon_p_dot/epsilon_p_dot_0);
```

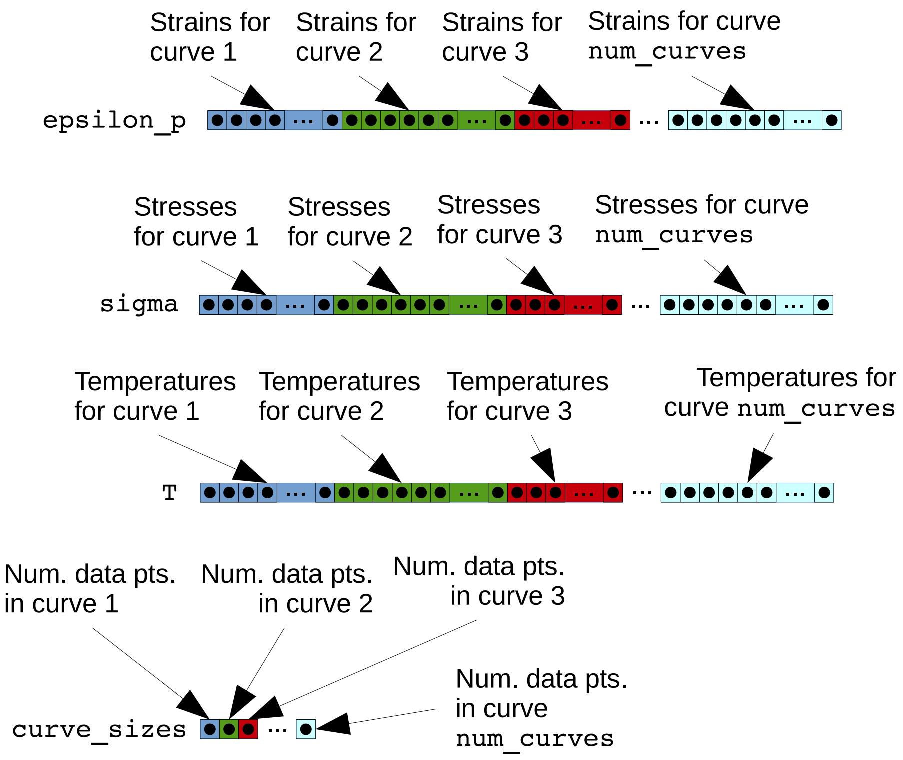


Fig. 5 **Storage of data for stress-strain curves in the Stan vectors `epsilon_p`, `sigma`, `T`, and `curve_sizes`**

```
  vector[sum(curve_sizes)] T_star = (T - T_room)/(T_melt - T_room);
}
```

The purpose of this block is to avoid redundant computation. Rather than repeatedly compute `log(epsilon_p_dot/epsilon_p_dot_0)` or `(T - T_room)/(T_melt - T_room)` again and again as a model is sampled, it is computed once and stored in `log_epsilon_p_dot` and `T_star`. The variable `T_star`, of course, represents T^*. This block also shows an example of arithmetic operators (such as the division operator "/") being applied to vectors. For example, dividing the vector `epsilon_p_dot` by `epsilon_p_dot_0` divides each element of `epsilon_p_dot` by `epsilon_p_dot_0`. The `log` function operates elementwise, so that if `x` is a vector, array, or matrix, and `y = log(x)`, then `y[i]` is the natural logarithm of `x[i]`.

The next block is the `parameters` block:

```
parameters {
  real<lower=0.0> A;
  real<lower=0.0> B;
  real<lower=0.0, upper=1.0> n;
  real<lower=0.0> C;
  real<lower=0.0> m;

  real<lower=0.0> sd_sigma[2];
}
```

This block, of course, contains the parameters of the Johnson-Cook model, along with constraints on their values. This block also contains an array parameter `sd_sigma` with two elements that correspond to parameters $SD_{\sigma,1}$ and $SD_{\sigma,2}$. The same notation used for constraining variables in the `data` block is used in this block as well.

The final block is the `model` block:

```
model {
  A ~ normal(A_guess_mean, A_guess_sd)T[0.0,];
  B ~ normal(B_guess_mean, B_guess_sd)T[0.0,];
  n ~ beta(n_alpha, n_beta);
  C ~ normal(C_guess_mean, C_guess_sd)T[0.0,];
  m ~ normal(m_guess_mean, m_guess_sd)T[0.0,];

  for (i in 1:2) {
    sd_sigma[i] ~
      normal(sd_sigma_guess_mean[i],
             sd_sigma_guess_sd[i])T[0.0,];
```

```
 81    }
 82
 83    {
 84      int start_ind = 1;
 85      for (curve_ind in 1:num_curves) {
 86        int end_ind = start_ind + curve_sizes[curve_ind] - 1;
 87
 88        real curr_sd_sigma = (epsilon_p_dot[curve_ind] <= 1.0
 89                              ? sd_sigma[1]
 90                              : sd_sigma[2]);
 91
 92        sigma[start_ind:end_ind] ~ normal(jc(epsilon_p[start_ind:end_ind],
 93                                             log_epsilon_p_dot[curve_ind],
 94                                             T_star[start_ind:end_ind],
 95                                             A, B, n, C, m),
 96                                          curr_sd_sigma);
 97
 98        start_ind = end_ind + 1;
 99      }
100    }
101  }
```

This contains the representation of the priors (lines 71–81) and the likelihood model (lines 83–100). By design, the "~" operator resembles the sampling statement notation used to specify parts of a Bayesian model, as seen in Sections 2 and 5. The `T[0,]` notation indicates that the normal distribution for the priors has been truncated so that the probability density of the prior is zero for negative parameter values. This corresponds to the $T[0,\infty)$ notation in Section 5. Indeed, aside from that slight difference in notation, variable declarations, and `for` loops, this section of the Stan specification file is nearly a transcription of the sampling statements in Eqs. 24–28 and Eq. 33.

The `for` loop within lines 83–100 is surrounded by braces so that the declaration for `start_ind` can be just above the loop. Otherwise, that declaration would need to be at the top of the `model` block, away from the context where the variable is most relevant.[*] In the body of this `for` loop, the variable `curr_sd_sigma` stands in for $SD_\sigma(\dot{\epsilon}_p)$. The notation with "?" and ":" in lines 88–90 is used here to express the contents of Eq. 21; if `epsilon_p_dot[curve_ind]` is less than 1, then `curr_sd_sigma` equals `sd_sigma[1]`; otherwise, it equals `sd_sigma[2]`. Also, in this same loop, one also can see the range notation for expressing segments of vectors and

[*]As mentioned before, when variable declarations are in a block of statements enclosed by braces, they must be at the top of the block, above any other statements.

arrays, where, for example, `sigma[start_ind:end_ind]` represents the sequence of values `sigma[start_ind]`, `sigma[start_ind + 1]`, ..., `sigma[end_ind]`. When `curve_ind` is 1, `start_ind` and `end_ind` are 1 and `curve_sizes[1]`, making `sigma[start_ind:end_ind]` the sequence of stress values for the first stress-strain curve. When `curve_ind` is 2, `sigma[start_ind:end_ind]` becomes the sequence of stress values for the second stress-strain curve, and so on for `curve_ind` values of 3, 4, and onward. The "~" operator applies to all elements of `sigma[start_ind:end_ind]`; essentially, in lines 92–96, Eq. 33 is applied for i_c = `curve_ind` and j ranging from 1 to `curve_sizes[curve_ind]`.

Given that in the `for` loop in lines 83–100, the "~" operator is used with a segment of a vector, one might expect that the `for` loop in lines 77–81 is unnecessary and that

```
// WARNING: Will not work!
sd_sigma ~ normal(sd_sigma_guess_mean, sd_sigma_guess_sd)T[0.0,];
```

could be used instead. If it were not for the truncation indicated by `T[0.0,]`, this would indeed be the case. Due to limitations of current implementations of Stan, though, this leads to a parsing error with the message "`Outcomes in truncated distributions must be univariate.`"

The Stan model specification file for the Zerilli-Armstrong (BCC) model, `za_bcc.stan`, is similar in form to the specification file for the Johnson-Cook model and is shown in Appendix C.

6.2 Specifying Models with PyMC3

With PyMC3, one creates a Bayesian model by starting from an empty model object and then adding objects to it that represent random variables. The random variables are one of two types:

- a parameter to be fit, which is initially set to its prior, or
- a variable associated with observed data, which defines part of the likelihood.

One can create a PyMC3 model directly in a Python script or Jupyter notebook. However, it is more flexible (and not much more difficult) to define the model via a Python function. This way, if the inputs to the model change, such as the stress-strain data or parameters for priors such as $C_{0,guess\ mean}$, a new model can be recreated simply by executing the function with the new inputs. Provided is a walk-through of a Python module file that contains such a function, `za_bcc_pymc3.py`, which is shown in Appendix D. This module file begins by importing from other modules, as shown:

```python
import numpy as np
import pymc3 as pm
from za_bcc import za_bcc
```

The first two statements, of course, import the NumPy and PyMC3 modules, and the third statement imports a function from a module that implements the Zerilli-Armstrong (BCC) model shown in Eq. 17. The contents of this last module are shown in Appendix D.

After the import statements begins the definition of the `make_za_bcc_model` function, which builds up a PyMC3 model object and then returns it. It has the following arguments:

- `epsilon_p` is a list of 1-D NumPy arrays. `epsilon_p[i]` is an array of strain values for the stress-strain curve `i`, where `epsilon_p[i][0]` is the first strain value for curve `i`, `epsilon_p[i][1]` is the second strain value for curve `i`, and so on.
- `sigma` and `T` are lists of arrays like `epsilon_p`, except that they hold stresses and temperatures, respectively, instead of strains.
- `epsilon_p_dot` is a list or 1-D NumPy array such that `epsilon_p_dot[i]` is the strain rate for curve `i`.
- `prior_params` is a Python dictionary used to set the parameters of the marginal prior PDFs for the Zerilli-Armstrong parameters. For example, the dictionary value `prior_params["C0_guess_mean"]` is $C_{0,guess\ mean}$.

The first statement in the body of the `make_za_bcc_model` function defines the truncated normal distribution $\text{normal}(\ldots)_{T[0,\infty)}$ described in Section 5:

```python
    PosNormal = pm.Bound(pm.Normal, lower = 0.0)
```

Here, `PosNormal` is an object representing a truncated normal distribution. The PyMC3 function `pm.Bound` takes as its first argument an object representing a distribution of a random variable, which in this case is `pm.Normal`, an object representing the normal distribution. It also takes one or more arguments specifying the bounds of truncation, in this case, a lower bound of 0.0.

The next couple statements define some variables that are used later:

```
    num_curves = len(epsilon_p)
    log_epsilon_p_dot = np.log(epsilon_p_dot)
```

Given how `epsilon_p` is defined, the number of its elements is the number of stress-strain curves, so for convenience, this number is assigned to `num_curves`. The array `log_epsilon_p_dot` contains the natural logarithms of the strain rates and is needed by the function `za_bcc`.

At this point, the Bayesian model begins to be built up.

```
    model = pm.Model()

    with model:
```

The first of these statements creates an empty model object and assigns it to the variable named `model`. The next line begins a Python `with` block. The statements that are part of this block (that is, the indented statements beneath the "`with model:`" clause) add to the empty model object. The following statements add information about the priors of the model:

```
        C0 = PosNormal("C0",
                       mu = prior_params["C0_guess_mean"],
                       sd = prior_params["C0_guess_sd"])

        C1 = PosNormal("C1",
                       mu = prior_params["C1_guess_mean"],
                       sd = prior_params["C1_guess_sd"])

        C3 = PosNormal("C3",
                       mu = prior_params["C3_guess_mean"],
                       sd = prior_params["C3_guess_sd"])

        C4 = PosNormal("C4",
                       mu = prior_params["C4_guess_mean"],
                       sd = prior_params["C4_guess_sd"])

        C5 = PosNormal("C5",
                       mu = prior_params["C5_guess_mean"],
85                         sd = prior_params["C5_guess_sd"])
86
87         n = pm.Beta("n",
88                     alpha = prior_params["n_alpha"],
89                     beta = prior_params["n_beta"])
90
91         sd_sigma = PosNormal("sd_sigma",
92                              mu = np.asarray(prior_params["sd_sigma_guess_mean"]),
93                              sd = np.asarray(prior_params["sd_sigma_guess_sd"]),
94                              shape = 2)
```

The statements in lines 67–85 are largely equivalent to Eq. 29. As mentioned previously, the `PosNormal` function in these lines represents normal$(\ldots)_{T[0,\infty)}$. The statement in lines 67–69 creates an object representing a random variable labeled with the string `"C0"` and adds that object to `model`. This object is also assigned to a Python variable that, for the sake of convenience, is also named `C0`. The only reason assignment to a variable is needed is because the variable is needed as an argument to the `za_bcc` function. Even without the assignment, the function call `PosNormal("C0", ...)` is sufficient to add a random variable labeled `"C0"` to the model. Lines 87–89 express Eq. 24, with `pm.Beta` representing the beta distribution. Lines 91–94 in the previous excerpt of Python code, which express Eqs. 30 and 31, create a random variable `sd_sigma` that is a 1-D array with two elements, as indicated by the argument `shape = 2`. Array elements `sd_sigma[0]` and `sd_sigma[1]` correspond to parameters $SD_{\sigma,1}$ and $SD_{\sigma,2}$, respectively.

The following Python `for` loop is used to add information about the likelihood to the model:

```
98         for i in range(num_curves):
99             pm.Normal("sigma_curve{}".format(i),
100                      mu = za_bcc(epsilon_p[i],
101                                  log_epsilon_p_dot[i], T[i],
102                                  C0, C1, C3, C4, C5, n,
103                                  exp_func = pm.math.exp),
104                      sd = (sd_sigma[0]
105                            if (epsilon_p_dot[i] <= 1.0)
106                            else sd_sigma[1]),
107                      observed = sigma[i])
```

The statement in the `for` loop executes a call to the function `pm.Normal`, which adds a normally distributed random variable to the model. The label of this random variable (which starts with `"sigma_curve"`) indicates that it is supposed to refer to flow stress σ associated with stress-strain curve `i`. The association with the flow stress data for curve `i` is established with the argument `observed = sigma[i]` (line 107), which also establishes that the random variable defines part of the likelihood. Since `sigma[i]` is an array, the random variable associated with it is an array as well, one that is the same size as `sigma[i]`. The function call that creates each of the random variables labeled `"sigma_curve0"`, `"sigma_curve1"`, and so on, represents the sampling statement in Eq. 34 being applied for all indices j in the interval $[1, N_{i_c}]$, with i_c = `i`. The argument for `mu` in lines 100–103 is the mean of the likelihood, $\sigma_{ZA,BCC}(\epsilon_{p,j}^{i_c}, \dot{\epsilon}_p^{i_c}, T_j^{i_c}, \boldsymbol{\theta}_{ZA,BCC})$ (again for all indices j in $[1, N_{i_c}]$ and i_c = `i`), while the argument for `sd` in lines 104–106 expresses Eq. 21.

After the `for` loop has finished, the function `make_za_bcc_model` can then return the PyMC3 model object that it has built up.

The function `za_bcc` in lines 100–103 from the previous excerpt of Python code has an additional argument, `exp_func`, that may seem confusing: an argument for an object representing the exponential function. The reasoning for the presence of this argument is as follows. When `za_bcc` is used to define a Bayesian model for PyMC3, it needs a version of the exponential function—namely, `pm.math.exp`—that can take as input PyMC3 objects representing random variables, since the Zerilli-Armstrong parameters are such objects in that context. However, in other contexts (such as those where `za_bcc` is used to generate simulated data), one needs an exponential function that operates on ordinary numbers, in which case the exponential function should be something like `np.exp` instead.

The Python module file for the Johnson-Cook model, `jc_pymc3.py`, is similar in form to the previous module file and is shown in Appendix D.

7. Implementing the Approximate Interval Predictor Model Approach in Python

Implementing an approximate IPM according to the scheme in Section 3 involves five steps:

1. Finding an estimate of $\boldsymbol{\theta}_0$

2. Finding the gradient of σ_{mdl} with respect to its parameters

3. Expressing Eqs. 9 and 12 in a form suitable for a particular linear programming implementation

4. Executing the linear programming implementation

5. Checking if the resulting estimates for $\Delta\boldsymbol{\theta}_{min}$ and $\Delta\boldsymbol{\theta}_{max}$ lead to a reasonable approximation for the set $\boldsymbol{\Theta}$

For the sake of convenience, the first step is implemented by taking $\boldsymbol{\theta}_0$ to be the mean of the joint PDF of $\boldsymbol{\theta}_{mdl}$ as estimated from a previously done Bayesian analysis. (Least squares regression could have been used to obtain $\boldsymbol{\theta}_0$, if the results of a Bayesian analysis were unavailable.)

The second step can be implemented with the aid of a symbolic computation package such as SymPy,[46] and the definition of a Python function resulting from this is as follows:

```python
import numpy as np

def jc_grad(epsilon_p, log_epsilon_p_dot, T_star,
            A, B, n, C, m):

    dJCdA = (-T_star**m + 1)*(C*log_epsilon_p_dot + 1.0)
    dJCdB = (epsilon_p**n)*(-T_star**m + 1)*(C*log_epsilon_p_dot + 1.0)

    dJCdn = np.where(epsilon_p == 0,
                     np.full(len(epsilon_p), 0.0),
                     B*(epsilon_p**n)*(-T_star**m + 1)*
                     (C*log_epsilon_p_dot + 1.0)*
                     np.log(epsilon_p))

    dJCdC = log_epsilon_p_dot*(A + B*epsilon_p**n)*(-T_star**m + 1)

    dJCdm = np.where(T_star == 0,
                     np.full(len(T_star), 0.0),
                     -T_star**m*(A + B*epsilon_p**n)*
                     (C*log_epsilon_p_dot + 1.0)*
                     np.log(T_star))

    return np.vstack((dJCdA, dJCdB, dJCdn, dJCdC, dJCdm))
```

The function `jc_grad` represents the gradient of the Johnson-Cook flow stress Eq. 15 with respect to parameters A, B, n, C, and m. However, it is not the result of a blind copy-and-paste from the output of SymPy. This would be problematic, since the expressions for the derivatives with respect to parameters n and

m are undefined where ϵ_p or T^* are zero, because of the presence of the factors $\epsilon_p^n \ln \epsilon_p$ and $(T^*)^m \ln T^*$, respectively, in those expressions. Mathematically, though, as $\epsilon_p \to 0$ and $T^* \to 0$, these factors approach zero, and the numerical calculation of the derivatives reflects that. Furthermore, to allow the Python function arguments `epsilon_p` and `T_star` to be arrays, `np.where` is used rather than a raw `if` statement.

The third step, of course, depends on one's choice of linear programming implementation. In the case of the `linprog` function from SciPy[9] (specifically its `optimize` submodule), Eq. 9 is expressed through an array $\mathbf{A}$ and vector $\mathbf{b}$ that satisfies the inequality $\mathbf{Au} \le \mathbf{b}$, where the operator "$\le$" is here taken to operate elementwise. The vector $\mathbf{u}$ describes the variables to be optimized. For the purposes of this section, it consists of the desired values of the elements of $\Delta\boldsymbol{\theta}_{min}$ followed by the desired values of the elements of $\Delta\boldsymbol{\theta}_{max}$. The rows of $\mathbf{A}$ are coefficients of the elements of $\mathbf{u}$, Each row of $\mathbf{A}$ and its corresponding element of $\mathbf{b}$ represents the left- and right-hand sides of an inequality. To fit this format, Eq. 12 can be combined with Eqs. 10 and 11 and rearranged to obtain

$$\begin{aligned} &-\frac{1}{2}\left(\mathbf{g}_{\sigma_{mdl}}(\mathbf{e}_i) + |\mathbf{g}_{\sigma_{mdl}}(\mathbf{e}_i)|\right)^T \Delta\boldsymbol{\theta}_{min} \\ &\quad+\frac{1}{2}\left(\mathbf{g}_{\sigma_{mdl}}(\mathbf{e}_i) - |\mathbf{g}_{\sigma_{mdl}}(\mathbf{e}_i)|\right)^T \Delta\boldsymbol{\theta}_{max} \le \sigma_i - \sigma_{mdl}(\mathbf{e}_i, \boldsymbol{\theta}_0) \end{aligned} \tag{35}$$

$$\begin{aligned} &\frac{1}{2}\left(\mathbf{g}_{\sigma_{mdl}}(\mathbf{e}_i) - |\mathbf{g}_{\sigma_{mdl}}(\mathbf{e}_i)|\right)^T \Delta\boldsymbol{\theta}_{min} \\ &\quad-\frac{1}{2}\left(\mathbf{g}_{\sigma_{mdl}}(\mathbf{e}_i) + |\mathbf{g}_{\sigma_{mdl}}(\mathbf{e}_i)|\right)^T \Delta\boldsymbol{\theta}_{max} \le -(\sigma_i - \sigma_{mdl}(\mathbf{e}_i, \boldsymbol{\theta}_0)) \end{aligned} \tag{36}$$

Equation 35 is used to determine row i of $\mathbf{A}$ and element i of $\mathbf{b}$, while Eq. 36 is used to determine row $2i$ of $\mathbf{A}$ and element $2i$ of $\mathbf{b}$. Let `ep_vec`, `log_ep_dot_vec`, `T_star_vec`, and `sigma_vec` represent vectors whose elements are values of ϵ_p, $\dot{\epsilon}_p$, T^*, σ, and let `theta_0` represent $\boldsymbol{\theta}_0$. Then, $\mathbf{A}$ and $\mathbf{b}$ (represented by the Python variables `A_mat` and `b_vec`) can be constructed as follows:

```python
num_data_pts = len(ep_vec)
half_len_u = len(theta_0)
len_u = 2*half_len_u

A_mat = np.empty((2*num_data_pts, len_u))
b_vec = np.empty(2*num_data_pts)

g_sigma_mdl = jc_grad(ep_vec, log_ep_dot_vec, T_star_vec, *theta_0)
```

```python
g_sigma_mdl_abs = np.fabs(g_sigma_mdl)

g_gabs_half_sum = 0.5*(g_sigma_mdl + g_sigma_mdl_abs)
g_gabs_half_diff = 0.5*(g_sigma_mdl - g_sigma_mdl_abs)

sigma_minus_sigma_mdl = sigma_vec - jc(ep_vec,
                                      log_ep_dot_vec,
                                      T_star_vec,
                                      *theta_0)

A_mat[:num_data_pts, :half_len_u] = -g_gabs_half_sum.T
A_mat[:num_data_pts, half_len_u:] = g_gabs_half_diff.T
b_vec[:num_data_pts] = sigma_minus_sigma_mdl

A_mat[num_data_pts:, :half_len_u] = g_gabs_half_diff.T
A_mat[num_data_pts:, half_len_u:] = -g_gabs_half_sum.T
b_vec[num_data_pts:] = -sigma_minus_sigma_mdl
```

Here, column i of `g_sigma_mdl` represents $\mathbf{g}_{\sigma_{mdl}}(\mathbf{e}_i)$, and `jc` is a Python function outputting the Johnson-Cook flow stress, shown in Appendix D.

In SciPy's `linprog` function, Eq. 12 is expressed as a vector whose elements are the coefficients of $\Delta\boldsymbol{\theta}'_{min}$ and $\Delta\boldsymbol{\theta}'_{max}$. Given the previous definitions of `g_sigma_mdl` and `g_sigma_mdl_abs`, this vector of coefficients can be represented in Python as follows:

```python
g_sigma_mdl_abs_sum = g_sigma_mdl_abs.sum(axis = 1)
coefficients = np.concatenate([g_sigma_mdl_abs_sum, g_sigma_mdl_abs_sum])
coefficients /= num_data_pts
```

Strictly speaking, it is not mathematically necessary to divide `coefficients` by `num_data_pts` (i.e., N in Eq. 12), but it makes the minimization more tractable.

The fourth step is largely straightforward:

```python
import scipy.optimize as so

result = so.linprog(coefficients, A_ub = A_mat, b_ub = b_vec,
                    method = "interior-point")

print("result.success = {}".format(result.success))

Delta_theta_min = result.x[:half_len_u]
Delta_theta_max = result.x[half_len_u:]

JC_param_lb = theta_0 - Delta_theta_min
JC_param_ub = theta_0 + Delta_theta_max
```

The Python vectors `JC_param_lb` and `JC_param_ub` represent the estimated lower and upper bounds on the Johnson-Cook parameters. One catch is that the default method used by `linprog` for minimization does not work for this problem, so a more robust alternative method, interior point, is used instead, hence the argument “`method = "interior-point"`” passed to `linprog`.

The fifth step is necessary because the lower and upper bounds in `JC_param_lb` and `JC_param_ub` are estimated *approximately* via a Taylor expansion. To see if these bounds are reasonable, the set $\boldsymbol{\Theta}$ is taken to be the hyperrectangle with the corners `JC_param_lb` and `JC_param_ub`, and $\sigma_{min}(\mathbf{e},\boldsymbol{\Theta})$ and $\sigma_{max}(\mathbf{e},\boldsymbol{\Theta})$ are estimated using Eqs. 6 and 7 (rather than the approximations in Eqs. 10 and 11). One can then determine how much of the flow stress data is actually bounded by $\sigma_{min}(\mathbf{e},\boldsymbol{\Theta})$ and $\sigma_{max}(\mathbf{e},\boldsymbol{\Theta})$. To do this, one first needs to create wrappers around the `jc` function that will work as objective functions for the `minimize` function from the `optimize` submodule of SciPy[9]:

```python
def jc_for_min(ABnCm, ep, l_epdot, T_s):
    return jc(ep, l_epdot, T_s,
              ABnCm[0], ABnCm[1], ABnCm[2], ABnCm[3], ABnCm[4])

def jc_for_max(ABnCm, ep, l_epdot, T_s):
    return -jc_for_min(ABnCm, ep, l_epdot, T_s)
```

Since a *minimization* routine is used to find σ_{max}, `jc_for_max` is the negative of the Johnson-Cook flow stress. (Maximizing an objective function is the same as minimizing the negative of that function.) One can then use the following `for` loop to generate estimates of σ_{min} and σ_{max} for each set of strain, strain rate, and temperature inputs and check how much of the data is within bounds:

```python
num_data_pts_in_bounds = 0

for i in range(num_data_pts):
    result_min = so.minimize(jc_for_min,
                             theta_0,
                             args = (ep_vec[i],
                                     log_ep_dot_vec[i],
                                     T_star_vec[i]),
                             bounds = so.Bounds(JC_param_lb,
                                                JC_param_ub))
    assert result_min.success

    result_max = so.minimize(jc_for_max,
                             theta_0,
                             args = (ep_vec[i],
                                     log_ep_dot_vec[i],
                                     T_star_vec[i]),
                                bounds = so.Bounds(JC_param_lb,
                                                   JC_param_ub))
    assert result_max.success

    sigma_min = result_min.fun
    sigma_max = -result_max.fun

    num_data_pts_in_bounds += int(sigma_min <= sigma[i] <= sigma_max)

print("Fraction of data points in bounds = {}".format(
      num_data_pts_in_bounds/num_data_pts))
```

A more complete example of implementing an approximate IPM in Python can be found in Ramsey.[11] An implementation in R can be found in Ramsey.[10]

8. Fitting Strength Models

8.1 Bayesian Analysis

Fitting the Bayesian models is done with multiple software implementations of MCMC: the `sampling` function of RStan 2.17.2,[13] the `sampling` method of PyStan 2.17.1.0,[16] and the `sample` function of the PyMC3 3.5 module.[6] Each MCMC run uses four chains. Each chain consists of at least 1000 warmup or tuning samples, which are discarded, followed by 1000 samples that are taken to be from the posterior distribution. Before fitting the models to real stress-strain data, they are tested by fitting them to simulated data—that is, data consisting of samples from the likelihood of the model given known model parameters and other model inputs—and ensuring that they yield point estimates for model parameters close to the parameter values used to generate the simulated data. Initial values of model parameters need to be supplied when using RStan or PyStan to run MCMC on the Zerilli-Armstrong (BCC) model, and when using PyMC3 to run MCMC on either the Johnson-Cook or Zerilli-Armstrong models. The initial value used for n is $n_\alpha/(n_\alpha + n_\beta)$, or 0.5. For the other parameters, the initial values used are the values of $A_{guess\ mean}$, $B_{guess\ mean}$, and so on.

After an MCMC run, diagnostics are run on the resulting chains to check for various potential problems. With RStan and PyMC3, some of these diagnostics, such as those pertaining to divergences and tree depth,[47–49] are run automatically. With PyStan, the corresponding diagnostics have to be executed manually after an MCMC run, and furthermore, they are provided by a third-party Python module[50] rather than PyStan itself. Other diagnostics, such as the potential scale reduction factor,

$\hat{R}$,[51] are computed when RStan, PyStan, or PyMC3 is prompted to print a table of statistics summarizing the MCMC run (via a function or method named `summary`). When $\hat{R} \approx 1$, the MCMC run is likely to have converged, provided that the other diagnostics do not indicate any problems.*

Histograms approximating the marginal posterior PDFs of the model parameters have been created for the values of β_{TQ} and f_{area} in Table 1. Figure 6 shows the posteriors for the model parameters of the Johnson-Cook model, assuming weakly informative priors, while Fig. 7 shows posteriors for the same strength model, but with the strongly informative prior for A based on the yield stress data from Benck.[30] The histograms shown happen to have been generated from MCMC samples from RStan and PyStan, respectively. However, histograms have also been generated from PyMC3, and these look largely the same as the ones shown in Figs. 6 and 7. Figure 8 shows the posteriors of model parameters for the Zerilli-Armstrong (BCC) model, when the model is fit to all of the available RHA data from MIDAS. Again, it makes little difference whether the histograms are generated from samples from RStan, PyStan, or PyMC3, so here histograms generated using samples from the last of these are shown. Whereas $SD_{\sigma,2}/SD_{\sigma,1} \approx 3$ for the Johnson-Cook model, for these fits to the Zerilli-Armstrong (BCC) model, $SD_{\sigma,1}$ and $SD_{\sigma,2}$ are much closer in value. To see if this is due to the Zerilli-Armstrong (BCC) model being fit to low-temperature data that are not used with the Johnson-Cook model, another set of fits to the Zerilli-Armstrong model has been done, using only the stress-strain data for temperatures 298 K and above. Posteriors from these fits (here generated from samples from RStan) are shown in Fig. 9.

While histogram plots are useful for visualizing the marginal PDFs of model parameters, they are not nearly as useful for expressing the PDFs in a form that may be input to tools, such as Dakota,[53] that take marginal PDFs of model parameters as input for uncertainty propagation analyses. A simple approximate approach is to report moments of the MCMC samples, such as the mean and standard deviation, for each model parameter. These may later be used to estimate the parameters of a closed-form marginal PDF via the method of moments,[54] and those parameters may be input to uncertainty propagation tools. For $\beta_{TQ} = 0.9$ and $f_{area} = 0.75$, the means and standard deviations of model parameters are shown in Tables 3 and 4.

*There is a bug in PyStan that can cause the calculation of $\hat{R}$ to yield spurious NaN values.[52] A workaround for this is to set `pystan.constants.EPSILON` to `float("-inf")` before starting this calculation.

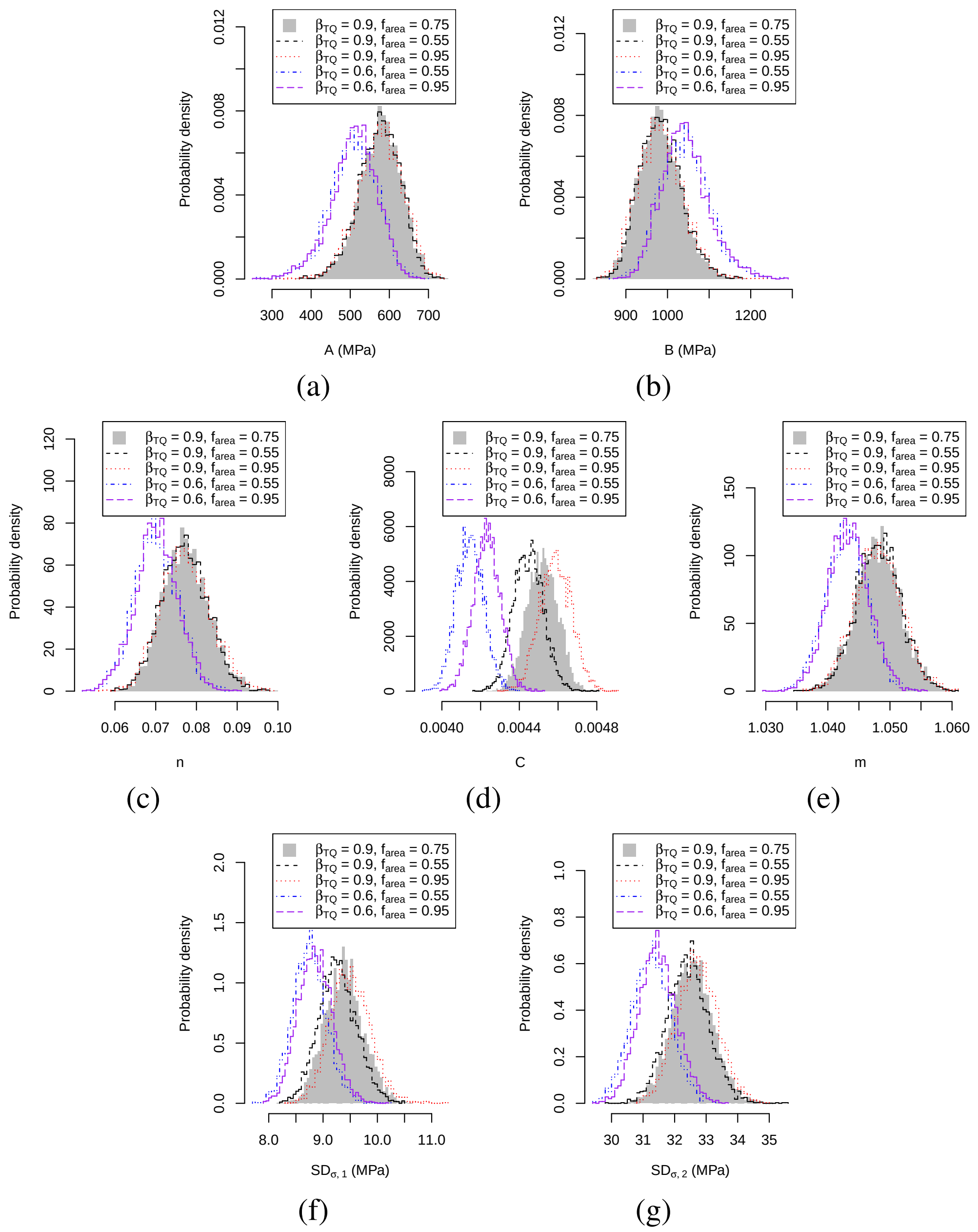


Fig. 6 Histograms approximating the posterior marginal PDFs of Johnson-Cook model parameters and nuisance parameters $SD_{\sigma,1}$ and $SD_{\sigma,2}$. These are generated from samples of RStan MCMC runs with the values of β_{TQ} and f_{area} in Table 1, and weakly informative priors.

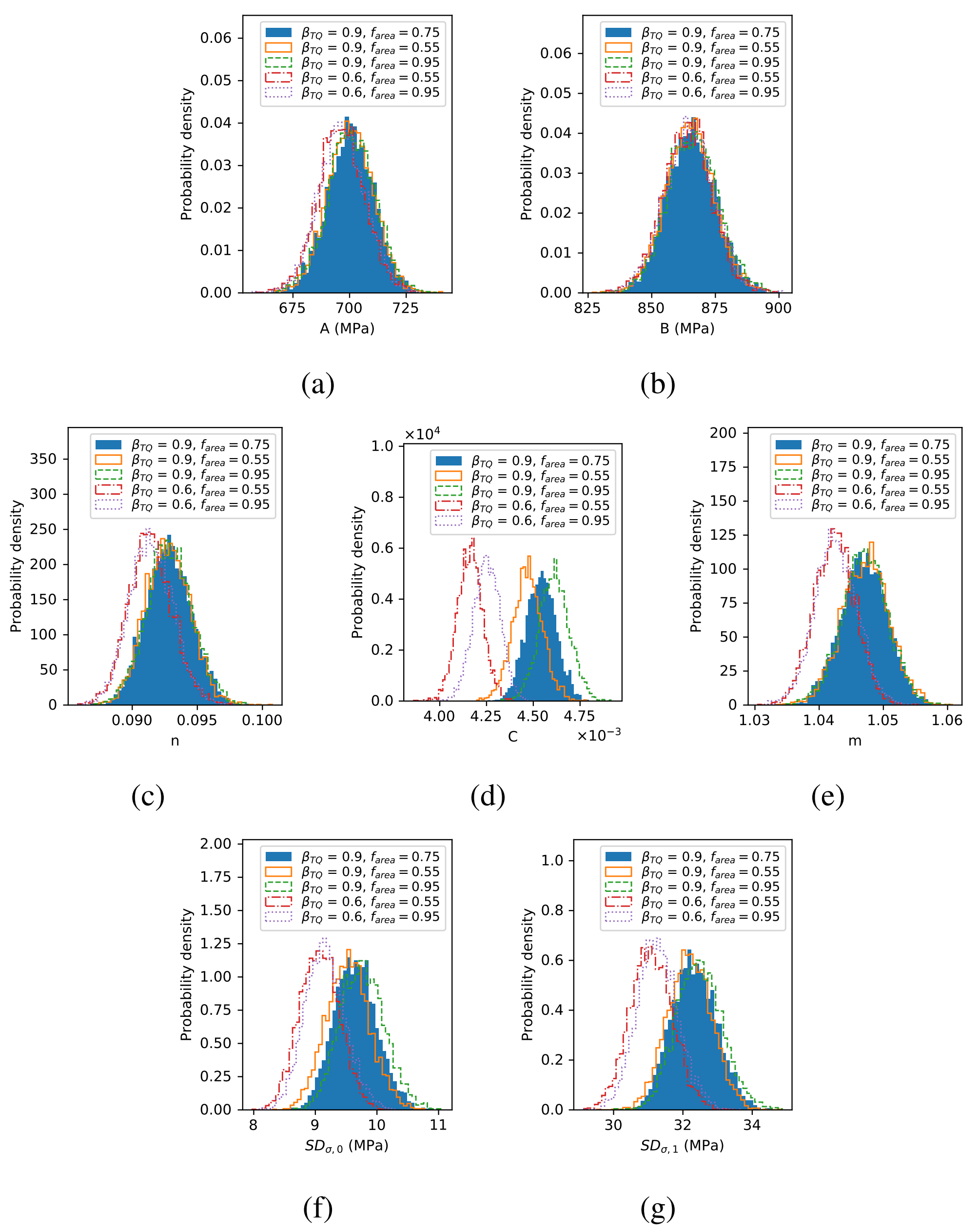


Fig. 7 Histograms approximating the posterior marginal PDFs of Johnson-Cook model parameters and nuisance parameters $SD_{\sigma,1}$ and $SD_{\sigma,2}$. These are generated from samples of PyStan MCMC runs with the values of β_{TQ} and f_{area} in Table 1, and a strongly informative prior for A.

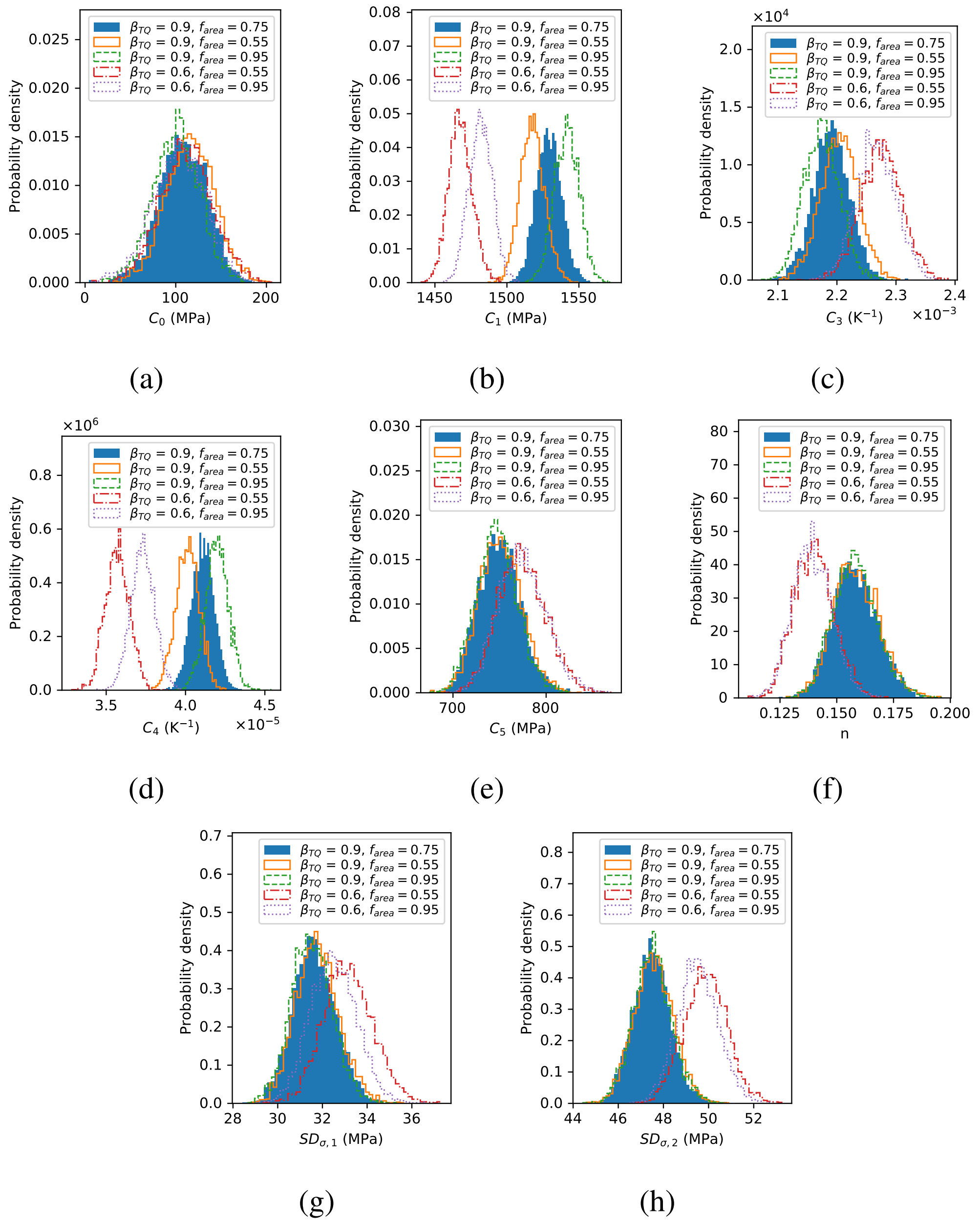


Fig. 8 Histograms approximating the posterior marginal PDFs of Zerilli-Armstrong (BCC) model parameters and nuisance parameters $SD_{\sigma,1}$ and $SD_{\sigma,2}$. These are generated from samples of PyMC3 MCMC runs with the values of β_{TQ} and f_{area} in Table 1, using data for all temperatures.

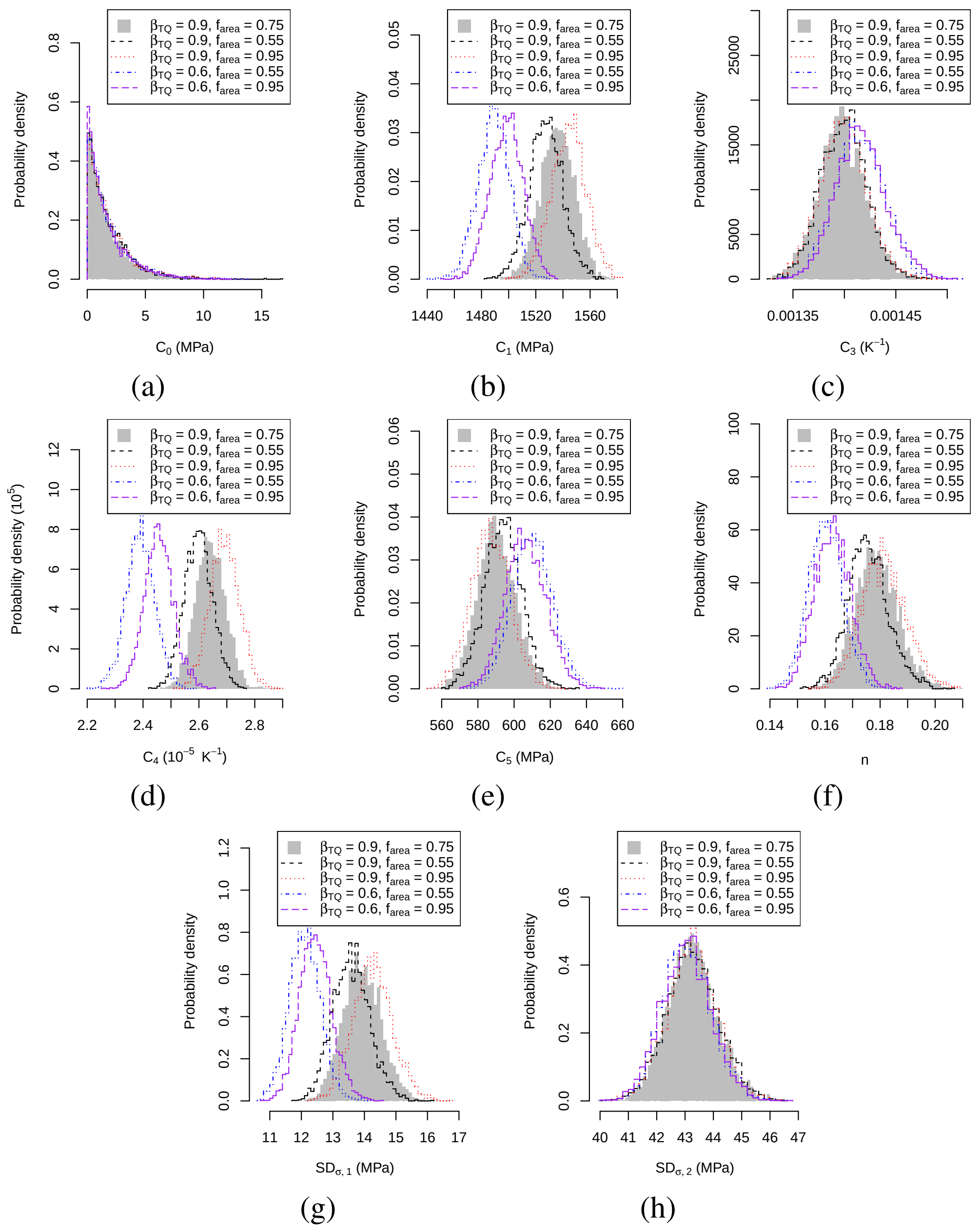


Fig. 9 Histograms approximating the posterior marginal PDFs of Zerilli-Armstrong (BCC) model parameters and nuisance parameters $SD_{\sigma,1}$ and $SD_{\sigma,2}$. These are generated from samples of RStan MCMC runs with the values of β_{TQ} and f_{area} in Table 1, using the same data used to fit the Johnson-Cook model.

For other values of β_{TQ} and f_{area}, they are shown in Tables 5–8.

Table 3 Mean and standard deviation of MCMC samples (from RStan) of parameters of Johnson-Cook model and nuisance parameters $SD_{\sigma,1}$ and $SD_{\sigma,2}$, for an MCMC run with weakly informative priors and a run with a strongly informative prior on A, with β_{TQ} = 0.9 and f_{area} = 0.75

	Weak prior	**Strong prior for *A***
Mean of A (MPa)	576.572779	699.842690
SD of A (MPa)	52.744813	10.100359
Mean of B (MPa)	982.583681	866.224370
SD of B (MPa)	50.288671	9.544459
Mean of n	0.077363	0.092704
SD of n	0.005634	0.001700
Mean of C	0.004519	0.004542
SD of C	0.000080	0.000081
Mean of m	1.048066	1.047197
SD of m	0.003670	0.003576
Mean of $SD_{\sigma,1}$ (MPa)	9.388639	9.645859
SD of $SD_{\sigma,1}$ (MPa)	0.355841	0.361472
Mean of $SD_{\sigma,2}$ (MPa)	32.563166	32.346724
SD of $SD_{\sigma,2}$ (MPa)	0.657904	0.665879

In addition to statistics for the marginal PDFs of the model parameters, one may also need information on how the PDFs of these parameters are correlated, especially if one intends to use these PDFs as input to uncertainty propagation analyses. For example, when the software Dakota is used for such analyses, it takes as input either a correlation or rank correlation matrix, depending on the method of uncertainty propagation used.[53] In R, these can be calculated via the `cor` function, and in Python, these can be calculated via the `corr` method of so-called data frame objects from the module Pandas.[55] Correlation matrices are shown in Tables 9–16.

8.2 Approximate Interval Predictor Approach

In the estimation of intervals for the Johnson-Cook parameters, $\boldsymbol{\theta}_0$ is taken to be the mean of the PDFs of the Johnson-Cook parameters for the case of a strong prior on A (using MCMC samples from PyMC3). This choice of point estimate $\boldsymbol{\theta}_0$ is motivated by the finding in Section 9.5 that this point estimate leads to a more accurate estimate of the yield stress. The bounds are shown in Tables 17 and 18.

In the estimation of intervals for the Zerilli-Armstrong (BCC) parameters, $\boldsymbol{\theta}_0$ is

Table 4 Mean and standard deviation of MCMC samples (from PyStan) of parameters of Zerilli-Armstrong (BCC) model and nuisance parameters $SD_{\sigma,1}$ and $SD_{\sigma,2}$, for an MCMC run using MIDAS RHA data for all available temperatures and a run using data for temperatures of 298 K and above, with $\beta_{TQ} = 0.9$ and $f_{area} = 0.75$

	All temps.	**Temps. above 298 K**
Mean of C_0 (MPa)	108.553321	1.839621
SD of C_0 (MPa)	25.904923	1.815009
Mean of C_1 (MPa)	1529.402241	1535.481564
SD of C_1 (MPa)	8.600941	12.284287
Mean of C_3 (K^{-1})	0.002189	0.001400
SD of C_3 (K^{-1})	0.000031	0.000023
Mean of C_4 (K^{-1})	0.000041	0.000026
SD of C_4 (K^{-1})	0.000001	0.000001
Mean of C_5 (MPa)	748.575960	590.804923
SD of C_5 (MPa)	22.102742	10.464798
Mean of n	0.158338	0.178280
SD of n	0.009668	0.007571
Mean of $SD_{\sigma,1}$ (MPa)	31.621727	13.919752
SD of $SD_{\sigma,1}$ (MPa)	0.939569	0.583100
Mean of $SD_{\sigma,2}$ (MPa)	47.465057	43.251601
SD of $SD_{\sigma,2}$ (MPa)	0.844922	0.904651

Table 5 Mean and standard deviations of MCMC samples (from PyMC3) of parameters of the Johnson-Cook model, given weakly informative priors

	$\beta_{TQ} = 0.9$ $f_{area} = 0.55$	$\beta_{TQ} = 0.9$ $f_{area} = 0.95$	$\beta_{TQ} = 0.6$ $f_{area} = 0.55$	$\beta_{TQ} = 0.6$ $f_{area} = 0.95$
Mean of A (MPa)	571.729244	577.465150	505.617796	508.842938
SD of A (MPa)	55.380371	54.856011	61.367112	60.277468
Mean of B (MPa)	986.494201	982.442422	1045.630598	1043.442048
SD of B (MPa)	52.838208	52.263373	59.117625	58.059811
Mean of n	0.076851	0.077521	0.069949	0.070252
SD of n	0.005833	0.005874	0.005396	0.005263
Mean of C	0.004448	0.004583	0.004141	0.004235
SD of C	0.000078	0.000083	0.000069	0.000071
Mean of m	1.048050	1.048270	1.043259	1.043241
SD of m	0.003611	0.003723	0.003182	0.003305
Mean of $SD_{\sigma,1}$ (MPa)	9.276518	9.491527	8.738919	8.853090
SD of $SD_{\sigma,1}$ (MPa)	0.354264	0.355389	0.310506	0.319895
Mean of $SD_{\sigma,2}$ (MPa)	32.418929	32.749525	31.299643	31.442369
SD of $SD_{\sigma,2}$ (MPa)	0.654751	0.654508	0.611589	0.627223

Table 6 Mean and standard deviations of MCMC samples (from PyMC3) of parameters of the Johnson-Cook model, given strongly informative prior for parameter A

	$\beta_{TQ} = 0.9$ $f_{area} = 0.55$	$\beta_{TQ} = 0.9$ $f_{area} = 0.95$	$\beta_{TQ} = 0.6$ $f_{area} = 0.55$	$\beta_{TQ} = 0.6$ $f_{area} = 0.95$
Mean of A (MPa)	700.329969	700.634539	695.752574	696.396703
SD of A (MPa)	9.820886	10.253312	10.181293	10.167980
Mean of B (MPa)	865.049053	866.181603	864.675977	864.968226
SD of B (MPa)	9.290963	9.780525	9.659818	9.613680
Mean of n	0.092745	0.092832	0.091313	0.091417
SD of n	0.001659	0.001703	0.001648	0.001666
Mean of C	0.004469	0.004612	0.004160	0.004253
SD of C	0.000076	0.000080	0.000069	0.000072
Mean of m	1.047219	1.047243	1.042508	1.042553
SD of m	0.003429	0.003622	0.003234	0.003336
Mean of $SD_{\sigma,1}$ (MPa)	9.531653	9.752201	9.040389	9.160295
SD of $SD_{\sigma,1}$ (MPa)	0.358033	0.357239	0.312129	0.325318
Mean of $SD_{\sigma,2}$ (MPa)	32.184856	32.512920	31.098566	31.274576
SD of $SD_{\sigma,2}$ (MPa)	0.621758	0.648511	0.600998	0.610700

Table 7 Mean and standard deviations of MCMC samples (from RStan) of parameters of the Zerilli-Armstrong (BCC) model, using data for all available temperatures

	$\beta_{TQ} = 0.9$ $f_{area} = 0.55$	$\beta_{TQ} = 0.9$ $f_{area} = 0.95$	$\beta_{TQ} = 0.6$ $f_{area} = 0.55$	$\beta_{TQ} = 0.6$ $f_{area} = 0.95$
Mean of C_0 (MPa)	113.827793	101.181205	109.965407	103.115558
SD of C_0 (MPa)	25.701954	25.116845	28.755151	27.341270
Mean of C_1 (MPa)	1517.897271	1541.482625	1467.823733	1481.833286
SD of C_1 (MPa)	8.633638	8.812933	8.309163	8.317124
Mean of C_3 (K^{-1})	0.002203	0.002175	0.002275	0.002267
SD of C_3 (K^{-1})	0.000031	0.000031	0.000035	0.000034
Mean of C_4 (K^{-1})	0.000040	0.000042	0.000036	0.000037
SD of C_4 (K^{-1})	0.000001	0.000001	0.000001	0.000001
Mean of C_5 (MPa)	752.244448	746.702766	775.958027	774.334031
SD of C_5 (MPa)	22.217745	21.753644	26.143405	24.276243
Mean of n	0.158015	0.158304	0.139478	0.139233
SD of n	0.009719	0.009423	0.008828	0.008342
Mean of $SD_{\sigma,1}$ (MPa)	31.796849	31.540939	33.165289	32.537641
SD of $SD_{\sigma,1}$ (MPa)	0.973442	0.941127	1.109610	1.076349
Mean of $SD_{\sigma,2}$ (MPa)	47.551671	47.521018	49.886479	49.514722
SD of $SD_{\sigma,2}$ (MPa)	0.851183	0.835937	0.905822	0.901693

Table 8 Mean and standard deviations of MCMC samples (from RStan) of parameters of the Zerilli-Armstrong (BCC) model, using the same data used to fit the Johnson-Cook model (i.e., data for temperatures above 298 K)

	$\beta_{TQ} = 0.9$ $f_{area} = 0.55$	$\beta_{TQ} = 0.9$ $f_{area} = 0.95$	$\beta_{TQ} = 0.6$ $f_{area} = 0.55$	$\beta_{TQ} = 0.6$ $f_{area} = 0.95$
Mean of C_0 (MPa)	1.869449	1.904104	1.841419	1.768092
SD of C_0 (MPa)	1.834897	1.877007	1.837713	1.773979
Mean of C_1 (MPa)	1527.354004	1544.250604	1489.155368	1498.549931
SD of C_1 (MPa)	12.444307	12.803857	11.826671	12.110162
Mean of C_3 (K^{-1})	0.001399	0.001399	0.001415	0.001416
SD of C_3 (K^{-1})	0.000022	0.000023	0.000024	0.000025
Mean of C_4 (K^{-1})	0.000026	0.000027	0.000024	0.000025
SD of C_4 (K^{-1})	0.000001	0.000001	0.000001	0.000001
Mean of C_5 (MPa)	593.815529	587.184189	610.048428	606.561237
SD of C_5 (MPa)	10.373255	10.749925	11.255388	11.251672
Mean of n	0.175715	0.181337	0.160380	0.162693
SD of n	0.007359	0.008088	0.006260	0.006489
Mean of $SD_{\sigma,1}$ (MPa)	13.584468	14.273011	12.116796	12.451585
SD of $SD_{\sigma,1}$ (MPa)	0.580964	0.624221	0.495483	0.510742
Mean of $SD_{\sigma,2}$ (MPa)	43.274173	43.216117	43.063903	43.053094
SD of $SD_{\sigma,2}$ (MPa)	0.911625	0.895961	0.867597	0.877680

taken to be the mean of the PDFs of the parameters fit only to data for temperatures of 298 K and above (again using MCMC samples from PyMC3). This choice of point estimate $\boldsymbol{\theta}_0$ is motivated by indications in Section 9 that the Zerilli-Armstrong (BCC) model appears ill-suited to fitting the low-temperature data for RHA. The bounds are shown in Tables 19 and 20. For the case where $\beta_{TQ} = 0.6$ and $f_{area} = 0.55$, the estimated lower bound is originally calculated to be on the order of -10^{-7} but is truncated to zero.

Table 9 Correlation matrices of model parameters of a Johnson-Cook model with weakly informative priors, generated from MCMC samples of RStan runs, for the values of β_{TQ} and f_{area} from Table 1

β_{TQ}	f_{area}	Correlation matrix					
			A	B	n	C	m
0.9	0.75	A	1.0	−1.0	0.99	0.06	−0.05
		B	−1.0	1.0	−0.99	−0.05	0.04
		n	0.99	−0.99	1.0	0.07	−0.07
		C	0.06	−0.05	0.07	1.0	−0.69
		m	−0.05	0.04	−0.07	−0.69	1.0
			A	B	n	C	m
0.9	0.55	A	1.0	−1.0	0.99	0.05	−0.06
		B	−1.0	1.0	−0.99	−0.04	0.04
		n	0.99	−0.99	1.0	0.05	−0.07
		C	0.05	−0.04	0.05	1.0	−0.69
		m	−0.06	0.04	−0.07	−0.69	1.0
			A	B	n	C	m
0.9	0.95	A	1.0	−1.0	0.99	0.13	−0.11
		B	−1.0	1.0	−0.99	−0.11	0.1
		n	0.99	−0.99	1.0	0.12	−0.12
		C	0.13	−0.11	0.12	1.0	−0.69
		m	−0.11	0.1	−0.12	−0.69	1.0
			A	B	n	C	m
0.6	0.55	A	1.0	−1.0	0.99	0.08	−0.05
		B	−1.0	1.0	−0.99	−0.07	0.05
		n	0.99	−0.99	1.0	0.07	−0.06
		C	0.08	−0.07	0.07	1.0	−0.65
		m	−0.05	0.05	−0.06	−0.65	1.0
			A	B	n	C	m
0.6	0.95	A	1.0	−1.0	0.99	0.01	−0.03
		B	−1.0	1.0	−0.99	−0.0	0.02
		n	0.99	−0.99	1.0	0.0	−0.03
		C	0.01	−0.0	0.0	1.0	−0.63
		m	−0.03	0.02	−0.03	−0.63	1.0

Table 10 Rank correlation matrices of model parameters of a Johnson-Cook model with weakly informative priors, generated from MCMC samples of RStan runs, for the values of β_{TQ} and f_{area} from Table 1

β_{TQ}	f_{area}	Rank correlation matrix					
			A	B	n	C	m
0.9	0.75	A	1.0	−1.0	0.99	0.07	−0.06
		B	−1.0	1.0	−0.99	−0.06	0.05
		n	0.99	−0.99	1.0	0.07	−0.07
		C	0.07	−0.06	0.07	1.0	−0.67
		m	−0.06	0.05	−0.07	−0.67	1.0
			A	B	n	C	m
0.9	0.55	A	1.0	−1.0	0.99	0.04	−0.05
		B	−1.0	1.0	−0.99	−0.03	0.03
		n	0.99	−0.99	1.0	0.04	−0.06
		C	0.04	−0.03	0.04	1.0	−0.67
		m	−0.05	0.03	−0.06	−0.67	1.0
			A	B	n	C	m
0.9	0.95	A	1.0	−1.0	1.0	0.12	−0.12
		B	−1.0	1.0	−0.99	−0.11	0.11
		n	1.0	−0.99	1.0	0.12	−0.12
		C	0.12	−0.11	0.12	1.0	−0.68
		m	−0.12	0.11	−0.12	−0.68	1.0
			A	B	n	C	m
0.6	0.55	A	1.0	−1.0	1.0	0.07	−0.06
		B	−1.0	1.0	−0.99	−0.07	0.05
		n	1.0	−0.99	1.0	0.07	−0.06
		C	0.07	−0.07	0.07	1.0	−0.63
		m	−0.06	0.05	−0.06	−0.63	1.0
			A	B	n	C	m
0.6	0.95	A	1.0	−1.0	1.0	0.01	−0.03
		B	−1.0	1.0	−0.99	0.0	0.02
		n	1.0	−0.99	1.0	0.0	−0.03
		C	0.01	0.0	0.0	1.0	−0.61
		m	−0.03	0.02	−0.03	−0.61	1.0

Table 11 Correlation matrices of model parameters of a Johnson-Cook model with a strongly informative prior for A, generated from MCMC samples of PyStan runs, for the values of β_{TQ} and f_{area} from Table 1

β_{TQ}	f_{area}	**Correlation matrix**					
			A	B	n	C	m
		A	1.0	−0.98	0.91	−0.01	0.01
0.9	0.75	B	−0.98	1.0	−0.83	0.07	−0.07
		n	0.91	−0.83	1.0	−0.0	−0.02
		C	−0.01	0.07	−0.0	1.0	−0.69
		m	0.01	−0.07	−0.02	−0.69	1.0
			A	B	n	C	m
		A	1.0	−0.98	0.92	0.02	−0.02
0.9	0.55	B	−0.98	1.0	−0.83	0.03	−0.05
		n	0.92	−0.83	1.0	0.02	−0.04
		C	0.02	0.03	0.02	1.0	−0.7
		m	−0.02	−0.05	−0.04	−0.7	1.0
			A	B	n	C	m
		A	1.0	−0.98	0.91	0.05	−0.06
0.9	0.95	B	−0.98	1.0	−0.82	0.01	−0.01
		n	0.91	−0.82	1.0	0.04	−0.09
		C	0.05	0.01	0.04	1.0	−0.7
		m	−0.06	−0.01	−0.09	−0.7	1.0
			A	B	n	C	m
		A	1.0	−0.99	0.92	0.04	−0.03
0.6	0.55	B	−0.99	1.0	−0.85	−0.01	−0.02
		n	0.92	−0.85	1.0	0.01	−0.04
		C	0.04	−0.01	0.01	1.0	−0.62
		m	−0.03	−0.02	−0.04	−0.62	1.0
			A	B	n	C	m
		A	1.0	−0.99	0.92	0.0	0.0
0.6	0.95	B	−0.99	1.0	−0.85	0.04	−0.05
		n	0.92	−0.85	1.0	−0.02	−0.0
		C	0.0	0.04	−0.02	1.0	−0.64
		m	0.0	−0.05	−0.0	−0.64	1.0

Table 12 Rank correlation matrices of model parameters of a Johnson-Cook model with a strongly informative prior for A, generated from MCMC samples of PyStan runs, for the values of β_{TQ} and f_{area} from Table 1

β_{TQ}	f_{area}	Rank correlation matrix					
			A	B	n	C	m
0.9	0.75	A	1.0	−0.98	0.9	−0.0	0.01
		B	−0.98	1.0	−0.81	0.06	−0.07
		n	0.9	−0.81	1.0	0.01	−0.02
		C	−0.0	0.06	0.01	1.0	−0.68
		m	0.01	−0.07	−0.02	−0.68	1.0
			A	B	n	C	m
0.9	0.55	A	1.0	−0.98	0.91	0.03	−0.02
		B	−0.98	1.0	−0.82	0.03	−0.04
		n	0.91	−0.82	1.0	0.03	−0.05
		C	0.03	0.03	0.03	1.0	−0.67
		m	−0.02	−0.04	−0.05	−0.67	1.0
			A	B	n	C	m
0.9	0.95	A	1.0	−0.98	0.9	0.03	−0.05
		B	−0.98	1.0	−0.81	0.02	−0.02
		n	0.9	−0.81	1.0	0.02	−0.08
		C	0.03	0.02	0.02	1.0	−0.68
		m	−0.05	−0.02	−0.08	−0.68	1.0
			A	B	n	C	m
0.6	0.55	A	1.0	−0.98	0.91	0.04	−0.03
		B	−0.98	1.0	−0.84	−0.01	−0.02
		n	0.91	−0.84	1.0	0.01	−0.04
		C	0.04	−0.01	0.01	1.0	−0.59
		m	−0.03	−0.02	−0.04	−0.59	1.0
			A	B	n	C	m
0.6	0.95	A	1.0	−0.98	0.92	0.02	−0.0
		B	−0.98	1.0	−0.84	0.02	−0.04
		n	0.92	−0.84	1.0	−0.01	−0.01
		C	0.02	0.02	−0.01	1.0	−0.62
		m	−0.0	−0.04	−0.01	−0.62	1.0

Table 13 Correlation matrices of model parameters of a Zerilli-Armstrong (BCC) model fit to data for all available temperatures, generated from MCMC samples of PyMC3 runs, for the values of β_{TQ} and f_{area} from Table 1

β_{TQ}	f_{area}	Correlation matrix						
			C_0	C_1	C_3	C_4	C_5	n
		C_0	1.0	−0.18	0.21	0.16	−0.89	0.89
		C_1	−0.18	1.0	−0.81	−0.29	−0.2	0.19
0.9	0.75	C_3	0.21	−0.81	1.0	0.58	0.2	−0.16
		C_4	0.16	−0.29	0.58	1.0	0.01	−0.04
		C_5	−0.89	−0.2	0.2	0.01	1.0	−0.88
		n	0.89	0.19	−0.16	−0.04	−0.88	1.0
			C_0	C_1	C_3	C_4	C_5	n
		C_0	1.0	−0.14	0.19	0.14	−0.89	0.89
		C_1	−0.14	1.0	−0.81	−0.26	−0.23	0.21
0.9	0.55	C_3	0.19	−0.81	1.0	0.56	0.21	−0.17
		C_4	0.14	−0.26	0.56	1.0	0.01	−0.06
		C_5	−0.89	−0.23	0.21	0.01	1.0	−0.89
		n	0.89	0.21	−0.17	−0.06	−0.89	1.0
			C_0	C_1	C_3	C_4	C_5	n
		C_0	1.0	−0.15	0.18	0.16	−0.89	0.89
		C_1	−0.15	1.0	−0.81	−0.24	−0.24	0.22
0.9	0.95	C_3	0.18	−0.81	1.0	0.56	0.23	−0.19
		C_4	0.16	−0.24	0.56	1.0	−0.01	−0.03
		C_5	−0.89	−0.24	0.23	−0.01	1.0	−0.88
		n	0.89	0.22	−0.19	−0.03	−0.88	1.0
			C_0	C_1	C_3	C_4	C_5	n
		C_0	1.0	−0.12	0.19	0.12	−0.91	0.91
		C_1	−0.12	1.0	−0.74	−0.08	−0.21	0.18
0.6	0.55	C_3	0.19	−0.74	1.0	0.44	0.19	−0.1
		C_4	0.12	−0.08	0.44	1.0	−0.03	0.0
		C_5	−0.91	−0.21	0.19	−0.03	1.0	−0.89
		n	0.91	0.18	−0.1	0.0	−0.89	1.0
			C_0	C_1	C_3	C_4	C_5	n
		C_0	1.0	−0.11	0.17	0.12	−0.91	0.91
		C_1	−0.11	1.0	−0.76	−0.16	−0.22	0.2
0.6	0.95	C_3	0.17	−0.76	1.0	0.47	0.2	−0.13
		C_4	0.12	−0.16	0.47	1.0	−0.02	−0.02
		C_5	−0.91	−0.22	0.2	−0.02	1.0	−0.9
		n	0.91	0.2	−0.13	−0.02	−0.9	1.0

Table 14 Rank correlation matrices of model parameters of a Zerilli-Armstrong (BCC) model fit to data for all available temperatures, generated from MCMC samples of PyMC3 runs, for the values of β_{TQ} and f_{area} from Table 1

β_{TQ}	f_{area}	**Rank correlation matrix**						
			C_0	C_1	C_3	C_4	C_5	n
		C_0	1.0	−0.16	0.19	0.15	−0.89	0.89
		C_1	−0.16	1.0	−0.79	−0.29	−0.19	0.19
0.9	0.75	C_3	0.19	−0.79	1.0	0.58	0.18	−0.15
		C_4	0.15	−0.29	0.58	1.0	0.0	−0.04
		C_5	−0.89	−0.19	0.18	0.0	1.0	−0.88
		n	0.89	0.19	−0.15	−0.04	−0.88	1.0
			C_0	C_1	C_3	C_4	C_5	n
		C_0	1.0	−0.12	0.17	0.12	−0.89	0.89
		C_1	−0.12	1.0	−0.79	−0.26	−0.22	0.21
0.9	0.55	C_3	0.17	−0.79	1.0	0.55	0.2	−0.16
		C_4	0.12	−0.26	0.55	1.0	0.02	−0.06
		C_5	−0.89	−0.22	0.2	0.02	1.0	−0.88
		n	0.89	0.21	−0.16	−0.06	−0.88	1.0
			C_0	C_1	C_3	C_4	C_5	n
		C_0	1.0	−0.14	0.17	0.15	−0.87	0.88
		C_1	−0.14	1.0	−0.8	−0.23	−0.24	0.22
0.9	0.95	C_3	0.17	−0.8	1.0	0.55	0.23	−0.19
		C_4	0.15	−0.23	0.55	1.0	0.0	−0.04
		C_5	−0.87	−0.24	0.23	0.0	1.0	−0.87
		n	0.88	0.22	−0.19	−0.04	−0.87	1.0
			C_0	C_1	C_3	C_4	C_5	n
		C_0	1.0	−0.11	0.18	0.12	−0.9	0.9
		C_1	−0.11	1.0	−0.73	−0.08	−0.21	0.18
0.6	0.55	C_3	0.18	−0.73	1.0	0.42	0.18	−0.11
		C_4	0.12	−0.08	0.42	1.0	−0.03	−0.0
		C_5	−0.9	−0.21	0.18	−0.03	1.0	−0.89
		n	0.9	0.18	−0.11	−0.0	−0.89	1.0
			C_0	C_1	C_3	C_4	C_5	n
		C_0	1.0	−0.1	0.16	0.12	−0.91	0.91
		C_1	−0.1	1.0	−0.75	−0.15	−0.21	0.19
0.6	0.95	C_3	0.16	−0.75	1.0	0.46	0.19	−0.12
		C_4	0.12	−0.15	0.46	1.0	−0.02	−0.01
		C_5	−0.91	−0.21	0.19	−0.02	1.0	−0.89
		n	0.91	0.19	−0.12	−0.01	−0.89	1.0

Table 15 Correlation matrices of model parameters of a Zerilli-Armstrong (BCC) model fit to the same data used to fit the Johnson-Cook model, generated from MCMC samples of RStan runs, for the values of β_{TQ} and f_{area} from Table 1

β_{TQ}	f_{area}	Correlation matrix						
			C_0	C_1	C_3	C_4	C_5	n
		C_0	1.0	−0.08	0.1	0.06	−0.04	0.04
		C_1	−0.08	1.0	−0.64	−0.14	−0.86	0.87
0.9	0.75	C_3	0.1	−0.64	1.0	0.74	0.86	−0.86
		C_4	0.06	−0.14	0.74	1.0	0.45	−0.47
		C_5	−0.04	−0.86	0.86	0.45	1.0	−0.86
		n	0.04	0.87	−0.86	−0.47	−0.86	1.0
			C_0	C_1	C_3	C_4	C_5	n
		C_0	1.0	−0.11	0.13	0.07	−0.01	0.0
		C_1	−0.11	1.0	−0.62	−0.09	−0.85	0.87
0.9	0.55	C_3	0.13	−0.62	1.0	0.71	0.85	−0.84
		C_4	0.07	−0.09	0.71	1.0	0.41	−0.42
		C_5	−0.01	−0.85	0.85	0.41	1.0	−0.85
		n	0.0	0.87	−0.84	−0.42	−0.85	1.0
			C_0	C_1	C_3	C_4	C_5	n
		C_0	1.0	−0.05	0.1	0.11	−0.05	0.05
		C_1	−0.05	1.0	−0.63	−0.12	−0.85	0.87
0.9	0.95	C_3	0.1	−0.63	1.0	0.74	0.85	−0.85
		C_4	0.11	−0.12	0.74	1.0	0.42	−0.45
		C_5	−0.05	−0.85	0.85	0.42	1.0	−0.85
		n	0.05	0.87	−0.85	−0.45	−0.85	1.0
			C_0	C_1	C_3	C_4	C_5	n
		C_0	1.0	−0.05	0.09	0.08	−0.06	0.06
		C_1	−0.05	1.0	−0.66	−0.16	−0.88	0.87
0.6	0.55	C_3	0.09	−0.66	1.0	0.72	0.88	−0.87
		C_4	0.08	−0.16	0.72	1.0	0.46	−0.49
		C_5	−0.06	−0.88	0.88	0.46	1.0	−0.89
		n	0.06	0.87	−0.87	−0.49	−0.89	1.0
			C_0	C_1	C_3	C_4	C_5	n
		C_0	1.0	−0.1	0.11	0.09	−0.01	0.01
		C_1	−0.1	1.0	−0.64	−0.15	−0.87	0.86
0.6	0.95	C_3	0.11	−0.64	1.0	0.73	0.88	−0.87
		C_4	0.09	−0.15	0.73	1.0	0.46	−0.49
		C_5	−0.01	−0.87	0.88	0.46	1.0	−0.88
		n	0.01	0.86	−0.87	−0.49	−0.88	1.0

Table 16 Rank correlation matrices of model parameters of a Zerilli-Armstrong (BCC) model fit to the same data used to fit the Johnson-Cook model, generated from MCMC samples of RStan runs, for the values of β_{TQ} and f_{area} from Table 1

β_{TQ}	f_{area}	Rank correlation matrix						
			C_0	C_1	C_3	C_4	C_5	n
		C_0	1.0	−0.06	0.07	0.04	−0.04	0.04
		C_1	−0.06	1.0	−0.62	−0.13	−0.85	0.87
0.9	0.75	C_3	0.07	−0.62	1.0	0.72	0.85	−0.84
		C_4	0.04	−0.13	0.72	1.0	0.42	−0.44
		C_5	−0.04	−0.85	0.85	0.42	1.0	−0.85
		n	0.04	0.87	−0.84	−0.44	−0.85	1.0
			C_0	C_1	C_3	C_4	C_5	n
		C_0	1.0	−0.09	0.12	0.07	−0.01	0.0
		C_1	−0.09	1.0	−0.58	−0.08	−0.83	0.86
0.9	0.55	C_3	0.12	−0.58	1.0	0.7	0.84	−0.82
		C_4	0.07	−0.08	0.7	1.0	0.4	−0.41
		C_5	−0.01	−0.83	0.84	0.4	1.0	−0.83
		n	0.0	0.86	−0.82	−0.41	−0.83	1.0
			C_0	C_1	C_3	C_4	C_5	n
		C_0	1.0	−0.06	0.1	0.1	−0.03	0.04
		C_1	−0.06	1.0	−0.61	−0.11	−0.84	0.87
0.9	0.95	C_3	0.1	−0.61	1.0	0.72	0.84	−0.84
		C_4	0.1	−0.11	0.72	1.0	0.41	−0.43
		C_5	−0.03	−0.84	0.84	0.41	1.0	−0.84
		n	0.04	0.87	−0.84	−0.43	−0.84	1.0
			C_0	C_1	C_3	C_4	C_5	n
		C_0	1.0	−0.04	0.08	0.07	−0.05	0.05
		C_1	−0.04	1.0	−0.64	−0.15	−0.86	0.86
0.6	0.55	C_3	0.08	−0.64	1.0	0.71	0.87	−0.87
		C_4	0.07	−0.15	0.71	1.0	0.44	−0.48
		C_5	−0.05	−0.86	0.87	0.44	1.0	−0.88
		n	0.05	0.86	−0.87	−0.48	−0.88	1.0
			C_0	C_1	C_3	C_4	C_5	n
		C_0	1.0	−0.1	0.11	0.08	0.0	0.0
		C_1	−0.1	1.0	−0.63	−0.15	−0.86	0.86
0.6	0.95	C_3	0.11	−0.63	1.0	0.71	0.87	−0.86
		C_4	0.08	−0.15	0.71	1.0	0.44	−0.46
		C_5	0.0	−0.86	0.87	0.44	1.0	−0.87
		n	0.0	0.86	−0.86	−0.46	−0.87	1.0

Table 17 Lower and upper bounds centered on point estimate for Johnson-Cook parameters given strong prior on A, for $\beta_{TQ} = 0.9$

	f_{area} = 0.55	f_{area} = 0.75	f_{area} = 0.95
Lower bound of A (MPa)	700.329969	700.618008	700.634539
Upper bound of A (MPa)	700.329969	700.618008	700.634540
Lower bound of B (MPa)	865.049053	865.515315	866.181602
Upper bound of B (MPa)	865.049053	865.515315	866.181612
Lower bound of n	0.072046	0.072011	0.071939
Upper bound of n	0.123858	0.123377	0.122826
Lower bound of C	0.004469	0.004541	0.004612
Upper bound of C	0.006972	0.007103	0.007242
Lower bound of m	0.876410	0.874260	0.872110
Upper bound of m	1.053115	1.051230	1.049291

Table 18 Lower and upper bounds centered on point estimate for Johnson-Cook parameters given strong prior on A, for $\beta_{TQ} = 0.6$

	f_{area} = 0.55	f_{area} = 0.95
Lower bound of A (MPa)	695.752574	696.396703
Upper bound of A (MPa)	695.752574	696.396703
Lower bound of B (MPa)	864.675977	864.968226
Upper bound of B (MPa)	864.675977	864.968226
Lower bound of n	0.071672	0.071666
Upper bound of n	0.121127	0.120480
Lower bound of C	0.004160	0.004253
Upper bound of C	0.006701	0.006872
Lower bound of m	0.878135	0.875202
Upper bound of m	1.053570	1.051180

Table 19 Lower and upper bounds centered on point estimate for Zerilli-Armstrong (BCC) parameters fit only to data for temperatures of 298 K and above, for β_{TQ} = 0.9

	f_{area} = 0.55	f_{area} = 0.75	f_{area} = 0.95
Lower bound of C_0 (MPa)	1.8539766	1.8678995	1.8794430
Upper bound of C_0 (MPa)	1.8539766	1.8678995	1.8794430
Lower bound of C_1 (MPa)	1527.7659993	1535.4145842	1544.2866591
Upper bound of C_1 (MPa)	1527.7659993	1535.4145842	1544.2866591
Lower bound of C_3 (K^{-1})	0.0013978	0.0013996	0.0013985
Upper bound of C_3 (K^{-1})	0.0013978	0.0013996	0.0013985
Lower bound of C_4 (K^{-1})	0.0000046	0.0000057	0.0000070
Upper bound of C_4 (K^{-1})	0.0000486	0.0000489	0.0000491
Lower bound of C_5 (MPa)	593.2154268	590.7882786	586.9919218
Upper bound of C_5 (MPa)	593.2154268	590.7882786	586.9919218
Lower bound of n (MPa)	0.1761679	0.1783322	0.1814527
Upper bound of n (MPa)	0.2122496	0.2162962	0.2222087

Table 20 Lower and upper bounds centered on point estimate for Zerilli-Armstrong (BCC) parameters fit only to data for temperatures of 298 K and above, for β_{TQ} = 0.6

	f_{area} = 0.55	f_{area} = 0.95
Lower bound of C_0 (MPa)	1.8041549	1.8190732
Upper bound of C_0 (MPa)	1.8041549	1.8190732
Lower bound of C_1 (MPa)	1489.5834168	1498.8622115
Upper bound of C_1 (MPa)	1489.5834168	1498.8622115
Lower bound of C_3 (K^{-1})	0.0014146	0.0014162
Upper bound of C_3 (K^{-1})	0.0014146	0.0014162
Lower bound of C_4 (K^{-1})	0.0000000	0.0000010
Upper bound of C_4 (K^{-1})	0.0000483	0.0000487
Lower bound of C_5 (MPa)	609.8943328	606.3797476
Upper bound of C_5 (MPa)	609.8943328	606.3797476
Lower bound of n (MPa)	0.1605899	0.1627577
Upper bound of n (MPa)	0.1881080	0.1918422

9. Evaluations of Model Fits

9.1 Comparison of Priors to Posteriors

As a sanity check, one may compare the priors for the model parameters to their corresponding posteriors. If a posterior largely resembles its corresponding prior, this suggests that the posterior has been largely determined by the prior rather than the likelihood, which is a problem if a prior is only weakly informative and little more than an educated guess. Accordingly, plots showing the marginal prior PDFs for Bayesian model parameters along with histograms estimating the marginal posterior PDFs are shown in Figs. 10–13. These plots are based on results from RStan, but again, the results from PyStan and PyMC3 are largely the same.

As should be the case, where weakly informative priors are used, the marginal posterior PDFs are, for the most part, substantially narrower than their corresponding prior PDFs. The one exception to this is for the prior and marginal posterior of C_0, in Fig. 12a, where the Zerilli-Armstrong model is being fit to all temperature data. This suggests that this model may not be sufficient to capture the trends in the stress and strain at all temperatures. It may also indicate that a more informative prior is needed for C_0, which depends on physical considerations such as the average grain diameter and the influence on the yield stress of such things as the presence of solutes and the initial dislocation density,[34] not all of which are known at this time.

Usually, the peaks of these posterior PDFs are different from the priors as well. There are three cases where this is not true. Two of these pertain to the parameter m in the Johnson-Cook fits. Here, both the prior and marginal posterior PDFs peak near $m = 1$, as shown in Figs. 10e and 11e. Since $m \approx 1$ in most of the fits done by Johnson and Cook in their original paper on their strength model,[33] this is not surprising and likely reflects the physical trend in thermal softening. Of perhaps more concern is the case for parameter B in the Johnson-Cook fit where all priors are weakly informative. It may very well be that the prior estimate of the mean value of parameter for B, 1000 MPa, is simply a very good guess, but given that 1) the fit with weakly informative priors appears to underestimate the yield strength of RHA and 2) the posterior mean for B is not nearly as close to 1000 MPa when a strongly informative prior for A is used, this appears unlikely.

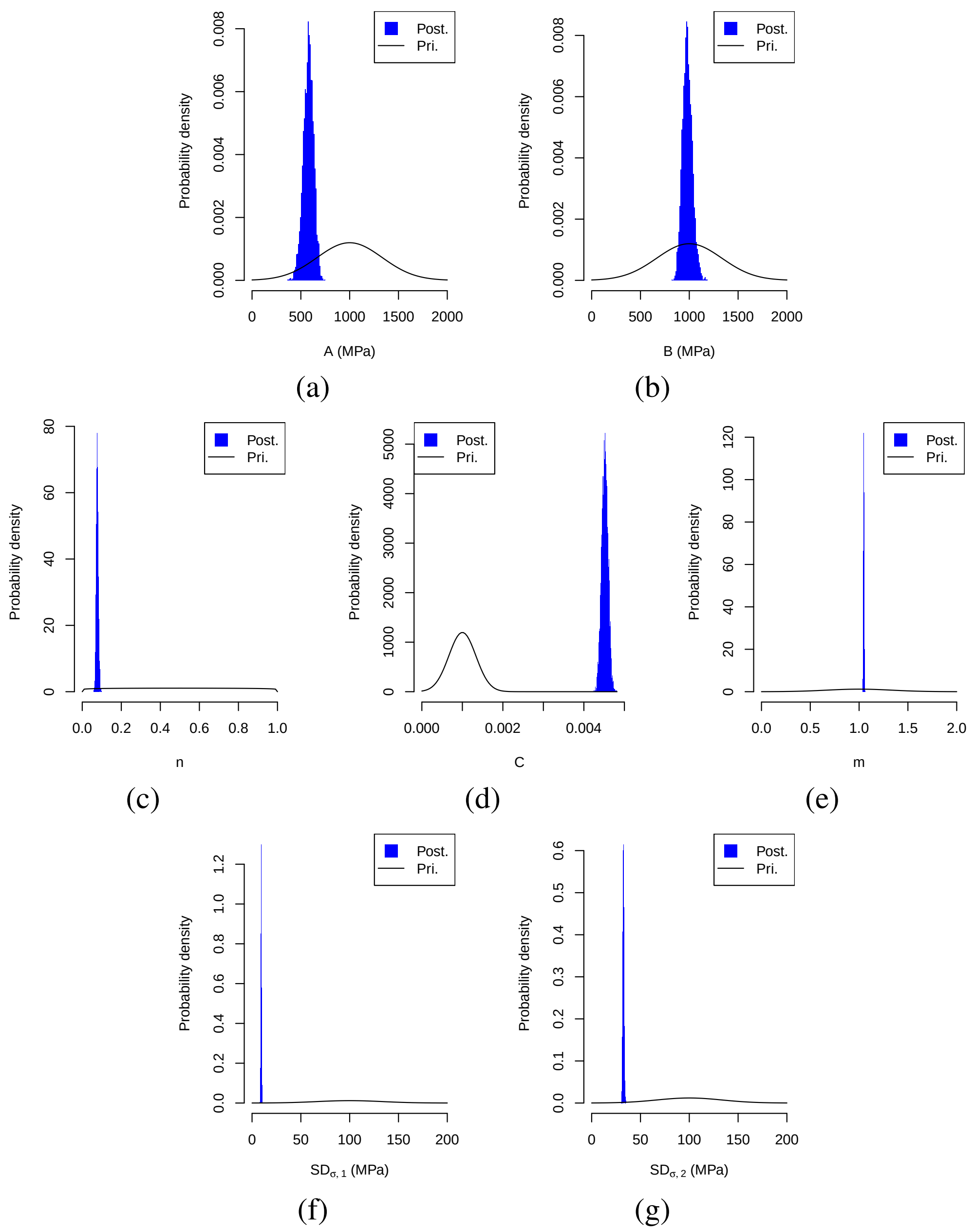


Fig. 10 Histograms approximating the posterior marginal PDFs of Johnson-Cook model parameters and nuisance parameters $SD_{\sigma,1}$ and $SD_{\sigma,2}$. These are generated from samples of an RStan MCMC run with $\beta_{TQ} = 0.9$, $f_{area} = 0.75$, and weakly informative priors. Priors are superimposed over the histograms.

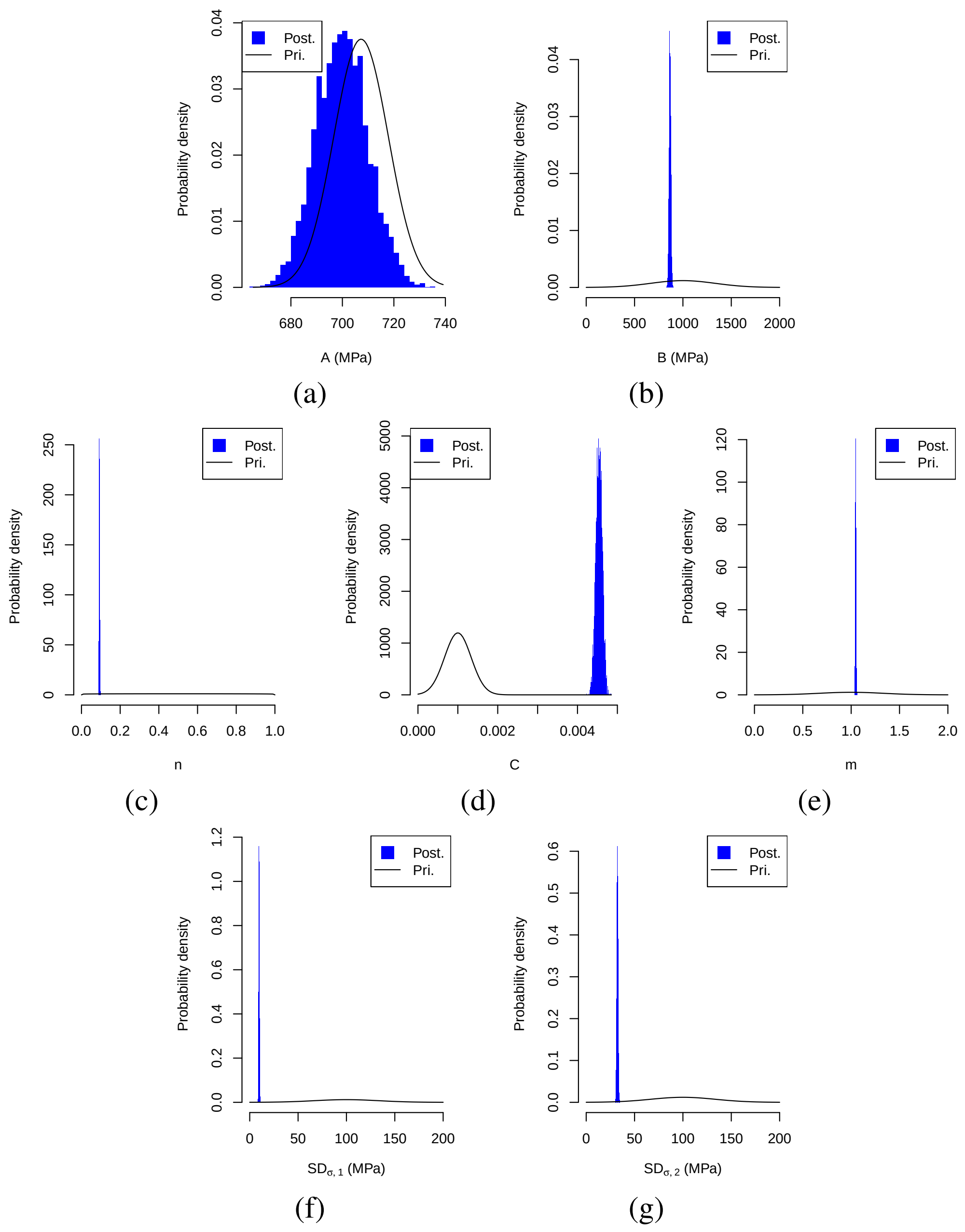


Fig. 11 Histograms approximating the posterior marginal PDFs of Johnson-Cook model parameters and nuisance parameters $SD_{\sigma,1}$ and $SD_{\sigma,2}$. These are generated from samples of an RStan MCMC run with $\beta_{TQ} = 0.9$, $f_{area} = 0.75$, and a strongly informative prior for A. Priors are superimposed over the histograms.

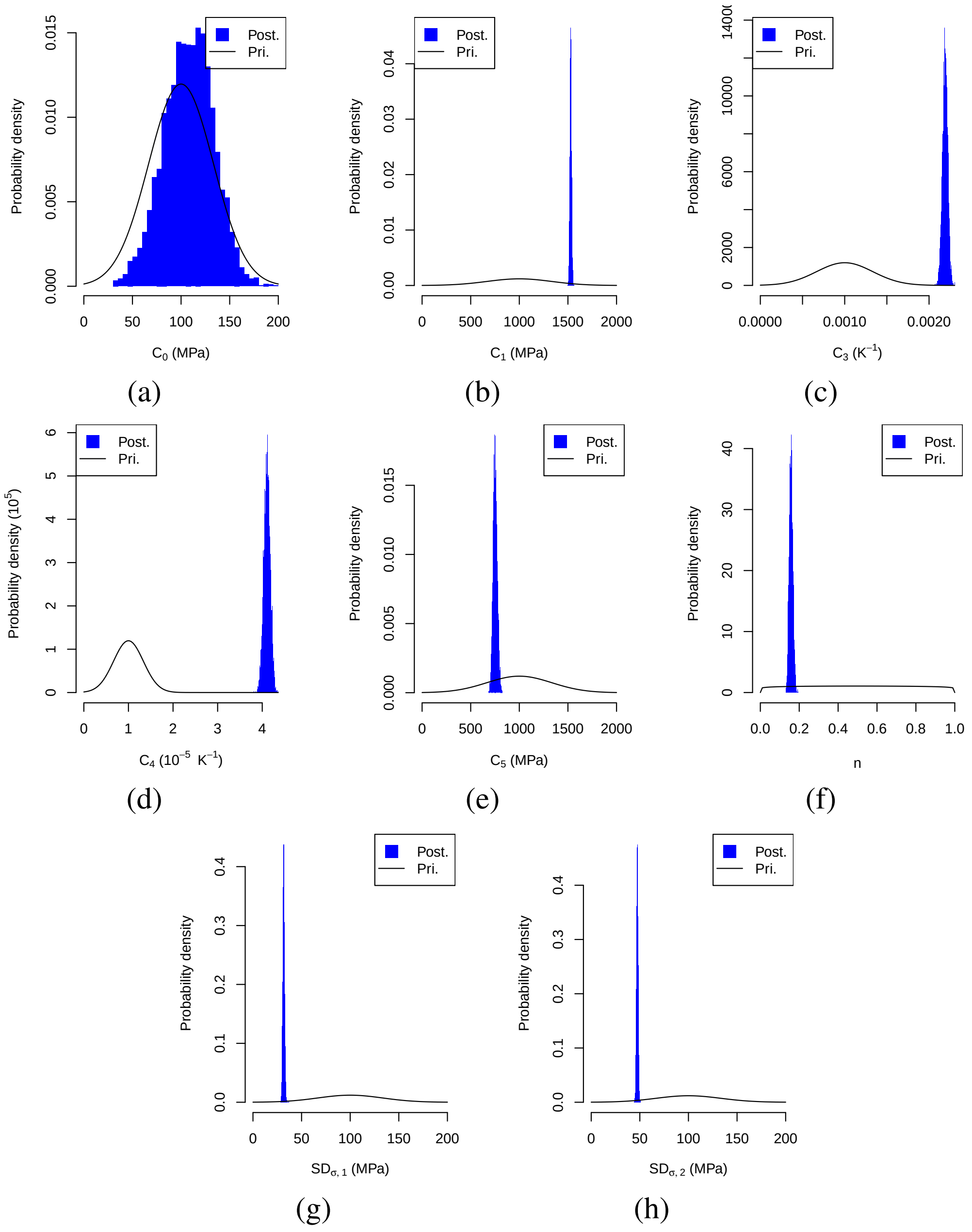


Fig. 12 Histograms approximating the posterior marginal PDFs of Zerilli-Armstrong (BCC) model parameters and nuisance parameters $SD_{\sigma,1}$ and $SD_{\sigma,2}$. These are generated from samples of an RStan MCMC run with $\beta_{TQ} = 0.9$, $f_{area} = 0.75$, using data for all temperatures. Priors are superimposed over the histograms.

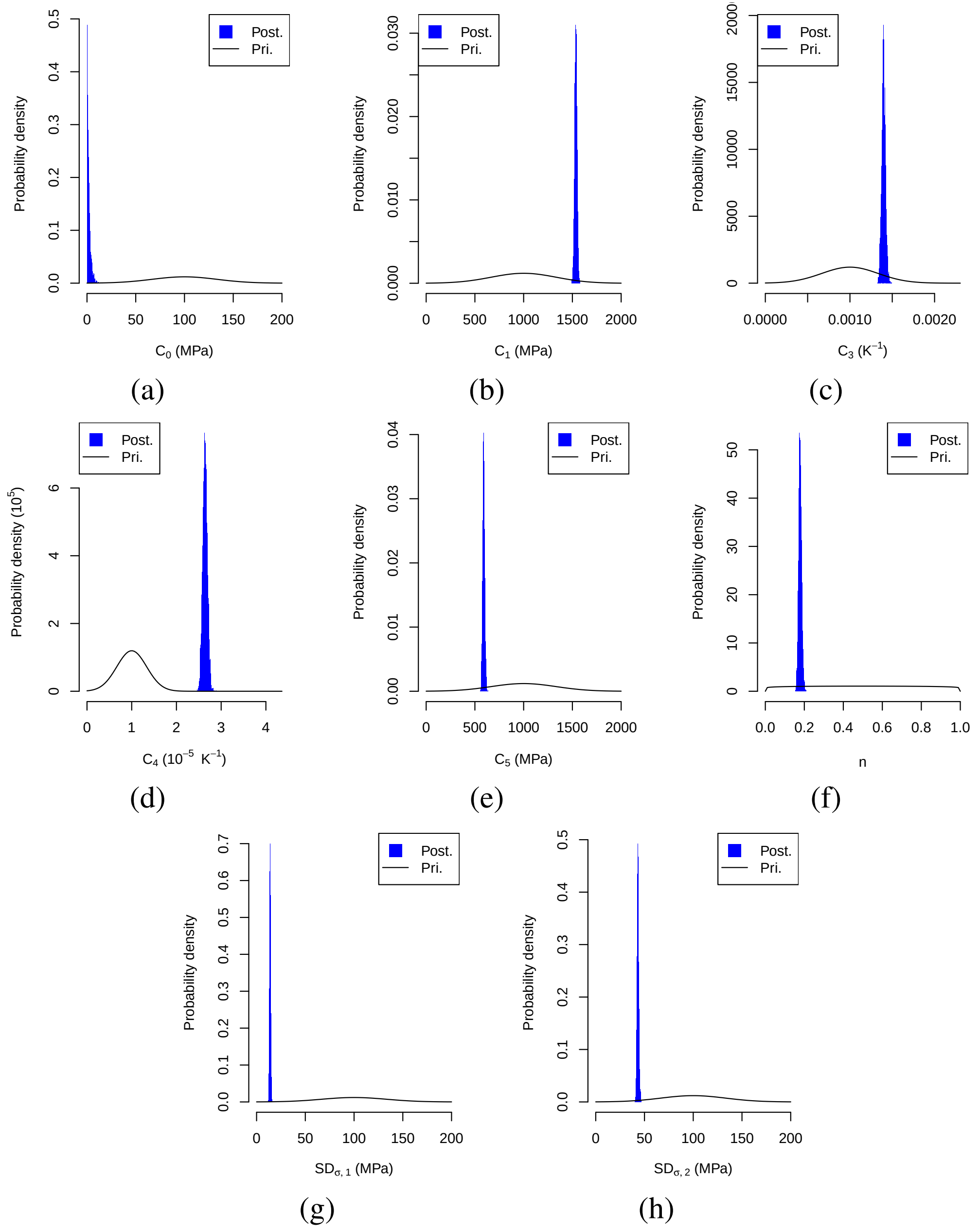


Fig. 13 Histograms approximating the posterior marginal PDFs of Zerilli-Armstrong (BCC) model parameters and nuisance parameters $SD_{\sigma,1}$ and $SD_{\sigma,2}$. These are generated from samples of an RStan MCMC run with $\beta_{TQ} = 0.9$, $f_{area} = 0.75$, using the same data used to fit the Johnson-Cook model. Priors are superimposed over the histograms.

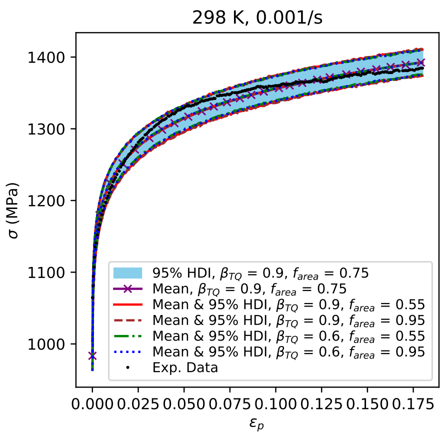
9.2 Comparison of PPD to Experimental Data Trends

As mentioned in Section 2, the PPD can be used to check how well a model's predictions accord with experimental data. The sampling statement for the PPD that corresponds to the likelihood in Eq. 32 is

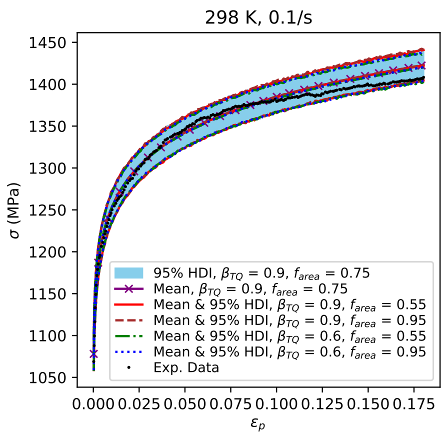
$$\sigma_j^{i_c,pred}(\mathbf{e}_j^{i_c}) \sim \text{normal}(\sigma_{mdl}(\mathbf{e}_j^{i_c}, \boldsymbol{\theta}_{mdl}), SD_{\sigma,k}), \text{ if } \{\boldsymbol{\theta}_{mdl}, SD_{\sigma,k}\} \sim \mathcal{D}_{post} \tag{37}$$

where $\mathcal{D}_{post}$ is the posterior distribution, $k = 1$ for strain rates of 1/s or less, and $k = 2$ otherwise. Unlike the PPD in the more general Eq. 4, here, there is a distinct PPD associated with each value of $\mathbf{e}_j^{i_c}$. To generate samples for the PPD, each MCMC sample of $\{\boldsymbol{\theta}_{mdl}, SD_{\sigma,k}\}$ is substituted into the likelihood $\text{normal}(\sigma_{mdl}(\mathbf{e}_i, \boldsymbol{\theta}_{mdl}), SD_{\sigma,k})$, and then a sample from that likelihood is taken to be a value $\sigma_j^{i_c,pred}(\mathbf{e}_j^{i_c})$, following Gelman et al.[18] Accordingly, the number of samples of the PPD for $\mathbf{e}_j^{i_c}$ (i.e., the number of values of $\sigma_j^{i_c,pred}(\mathbf{e}_j^{i_c})$ drawn) is the same as the number of MCMC samples. Since there are $n_c N_{i_c}$ PPDs (one for each value of $i_c \in [1, n_c]$ and $j \in [1, N_{i_c}]$), and this number of PPDs may be large, visualizing each PPD with a histogram would be unwieldy. Instead, two statistics are to be taken from each PPD, the mean and the 95% highest density interval (HDI). The 95% HDI of $\sigma_j^{i_c,pred}(\mathbf{e}_j^{i_c})$ is the interval such that 1) the probability that a value of $\sigma_j^{i_c,pred}(\mathbf{e}_j^{i_c})$ is in this interval is 95% and 2) the values within this interval all have higher probability densities than values outside of it.[17] Figures 14–22 show the mean and 95% HDI for the PPDs of the Johnson-Cook and Zerilli-Armstrong (BCC) models under various fitting conditions, along with the experimental data.

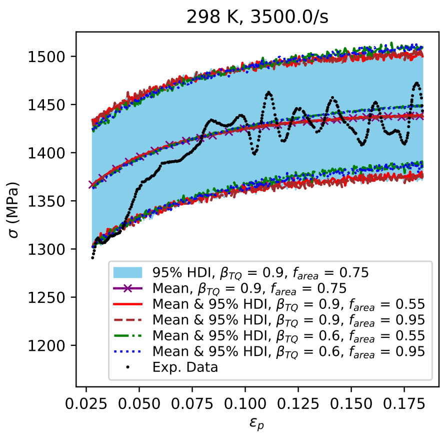
The experimental data are largely within the 95% HDI of the model fits shown in the aforementioned figures. However, as a measure of the fit of the model to experimental data, this is a low bar, since the width of the HDI is determined largely by $SD_{\sigma,k}$, which tends to increase with data spread or misfit. If one compares the means of the model fits to experimental data, one can see that they do not quite track the trends in the data, even for the quasi-static data that do not have the same problems with oscillation as the high-strain-rate data. For example, in Fig. 14a, where the initial sample temperature is 298 K and the strain rate is 0.001/s, the curvature of the mean prediction curves (for various values of β_{TQ} and f_{area}) of the Johnson-Cook model are such that they overpredict the flow stress for plastic strains between about 2.5% to 10%, and then underpredict the strains thereafter. In Fig. 14b, where the strain rate is 0.1/s, the mean model predictions largely track the



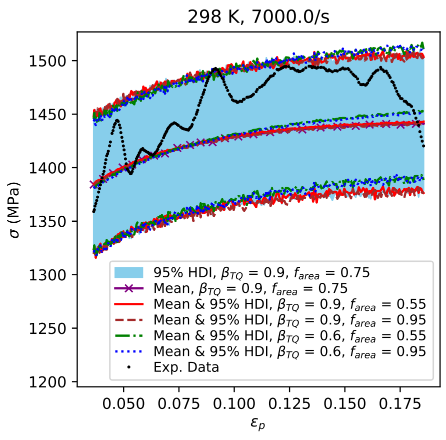
(a)



(b)

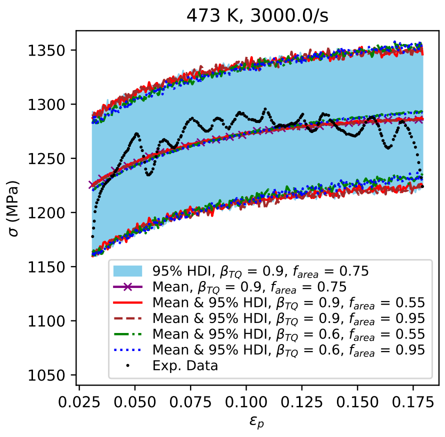


(c)

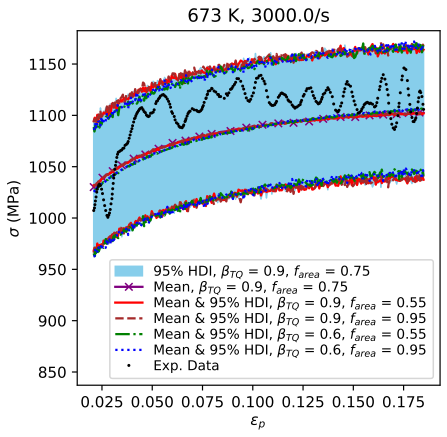


(d)

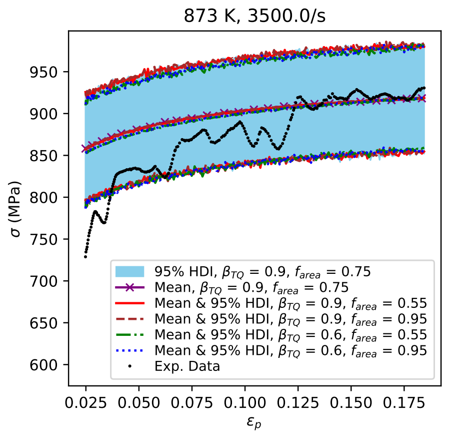
Fig. 14 Stress-strain data for initial sample temperatures of 298 K, along with estimates of the mean and the 95% HDI for PPDs generated from samples of PyStan MCMC runs for the Johnson-Cook model with weakly informative priors. The 95% HDI for β_{TQ} = 0.9 and f_{area} = 0.75 is plotted as a shaded region between the minimum and maximum of the HDI.



(a)

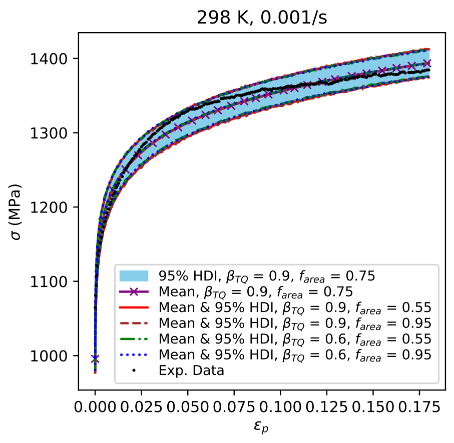


(b)

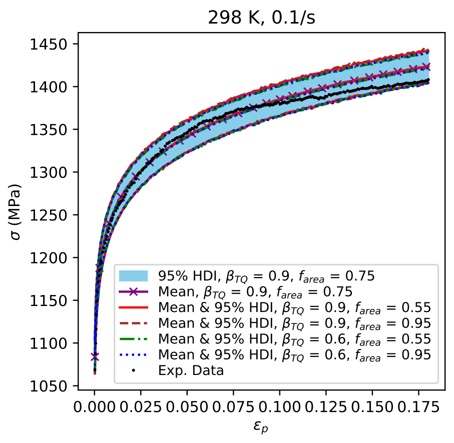


(c)

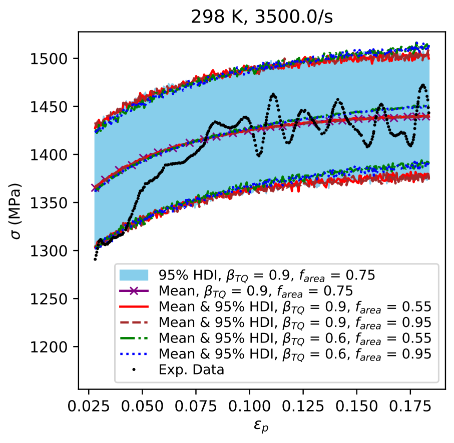
Fig. 15 Stress-strain data for high initial sample temperatures along with estimates of the mean and the 95% HDI for PPDs generated from samples of PyStan MCMC runs for the Johnson-Cook model with weakly informative priors. The 95% HDI for $\beta_{TQ} = 0.9$ and $f_{area} = 0.75$ is plotted as a shaded region between the minimum and maximum of the HDI.



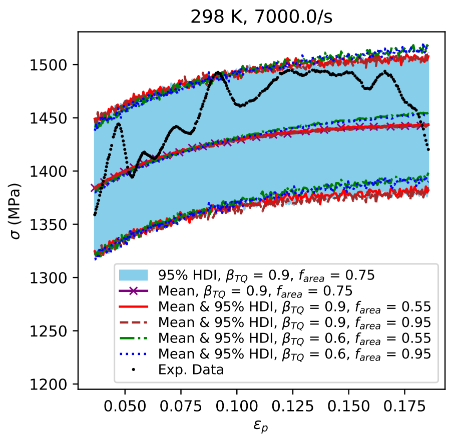
(a)



(b)

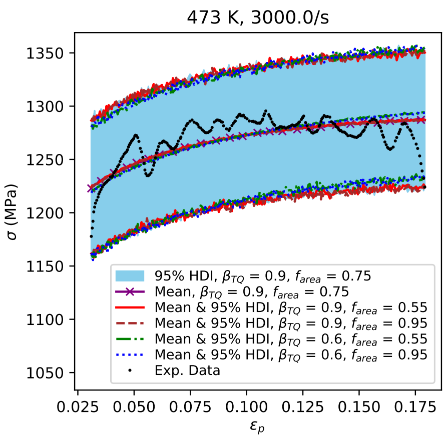


(c)

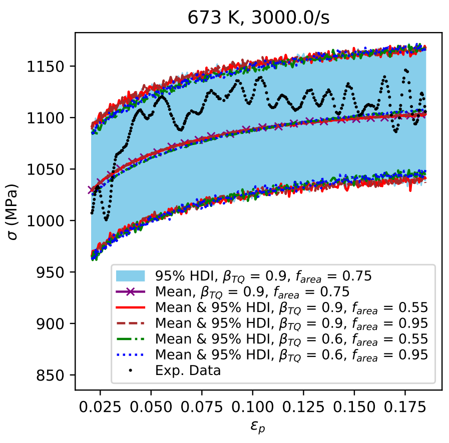


(d)

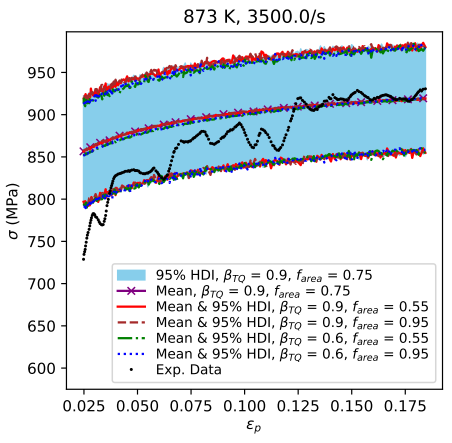
Fig. 16 Stress-strain data for initial sample temperatures of 298 K, along with estimates of the mean and the 95% HDI for PPDs generated from samples of PyStan MCMC runs for the Johnson-Cook model with a strongly informative prior for A. The 95% HDI for β_{TQ} = 0.9 and f_{area} = 0.75 is plotted as a shaded region between the minimum and maximum of the HDI.



(a)

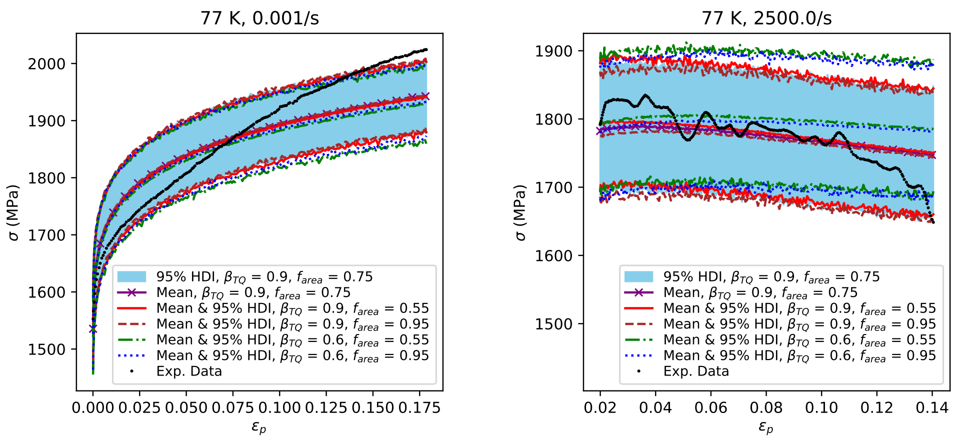


(b)



(c)

Fig. 17 Stress-strain data for high initial sample temperatures along with estimates of the mean and the 95% HDI for PPDs generated from samples of PyStan MCMC runs for the Johnson-Cook model with a strongly informative prior for A. The 95% HDI for $\beta_{TQ} = 0.9$ and $f_{area} = 0.75$ is plotted as a shaded region between the minimum and maximum of the HDI.



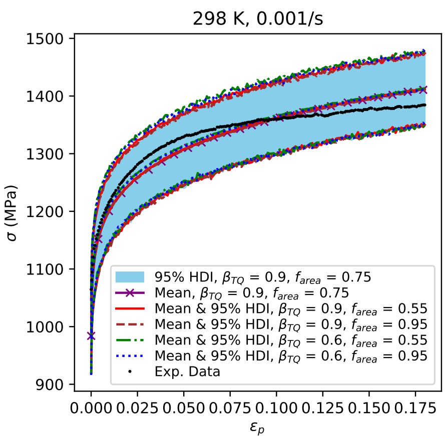
(a) (b)

Fig. 18 Stress-strain data for initial sample temperatures of 77 K, along with estimates of the mean and the 95% HDI for PPDs generated from samples of PyStan MCMC runs for the Zerilli-Armstrong (BCC) model fit data for all available temperatures. The 95% HDI for $\beta_{TQ} = 0.9$ and $f_{area} = 0.75$ is plotted as a shaded region between the minimum and maximum of the HDI.

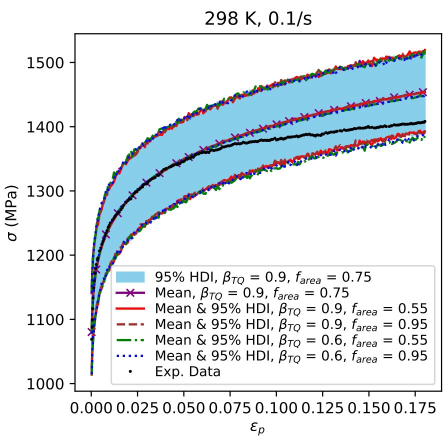
experimental data up to about 10% plastic strain, and then increasingly underpredict the flow stress. A similar problem can be seen in Fig. 19b, where the initial sample temperature is 298 K and the strain rate is 0.1/s, the mean model predictions of the Zerilli-Armstrong (BCC) model are such that they largely track the experimental data until about 5% strain, where they also begin to increasingly underpredict the flow stress.

Figure 18a shows why the mean of $SD_{\sigma,1}$ for the Zerilli-Armstrong (BCC) models fit to data for all available temperatures tends to be larger than the mean of $SD_{\sigma,1}$ for models fit only to data for temperatures 298 K and above. Here, the experimental data show a response that is much stiffer than the overall flatter response predicted from the Zerilli-Armstrong model, which also has to fit the comparatively flatter stress-strain data for other initial sample temperatures and strain rates. The parameter $SD_{\sigma,1}$, then, has to expand to account for this discrepancy.

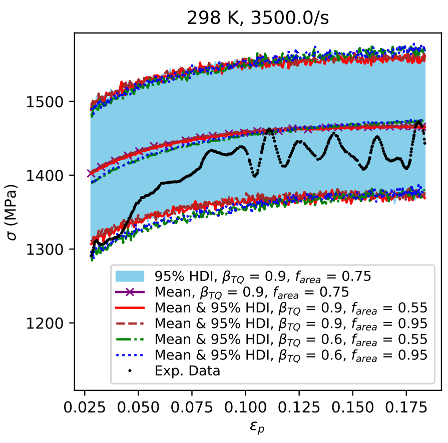
While the model fits are rather approximate, they do appear to be improved by accounting for the temperature rise during sample deformation. In one particular case, the one for a sample with initial temperature of 77 K and strain rate $\dot{\epsilon}_p = 2500/\text{s}$, it appears necessary for an even remotely reasonable fit to be obtained at all. As seen in Fig. 1, for this case, the stress-strain curve slopes downward, at least for



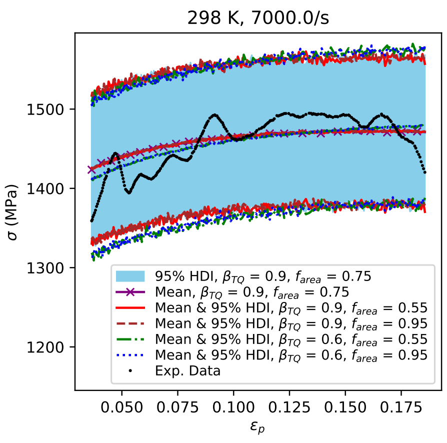
(a)



(b)

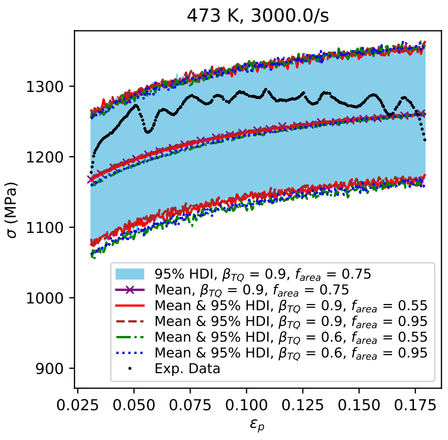


(c)

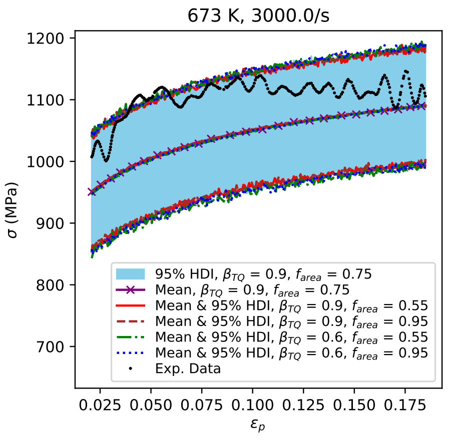


(d)

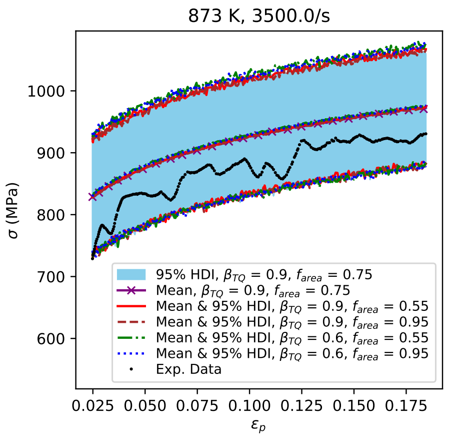
Fig. 19 Stress-strain data for initial sample temperatures of 298 K, along with estimates of the mean and the 95% HDI for PPDs generated from samples of PyStan MCMC runs for the Zerilli-Armstrong (BCC) model fit data for all available temperatures. The 95% HDI for $\beta_{TQ} = 0.9$ and $f_{area} = 0.75$ is plotted as a shaded region between the minimum and maximum of the HDI.



(a)



(b)



(c)

Fig. 20 Stress-strain data for high initial sample temperatures along with estimates of the mean and the 95% HDI for PPDs generated from samples of PyStan MCMC runs for the Zerilli-Armstrong (BCC) model fit to data for all available temperatures. The 95% HDI for $\beta_{TQ} = 0.9$ and $f_{area} = 0.75$ is plotted as a shaded region between the minimum and maximum of the HDI.

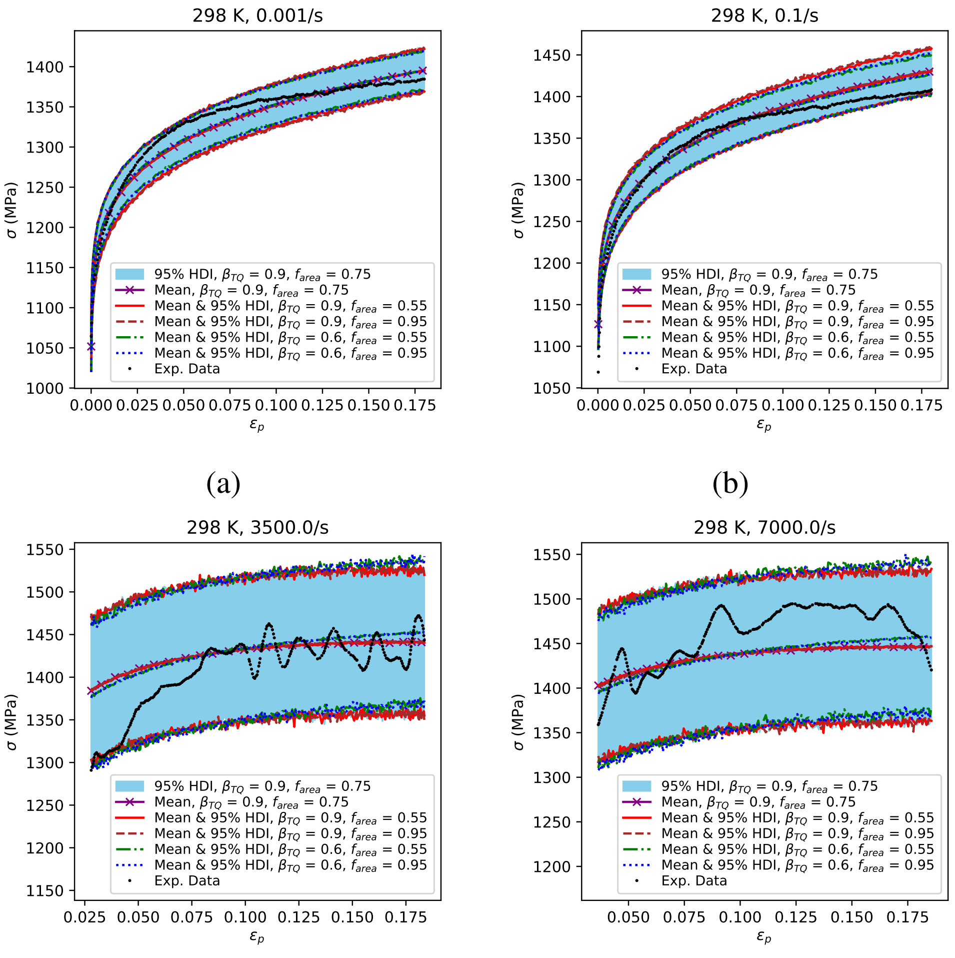
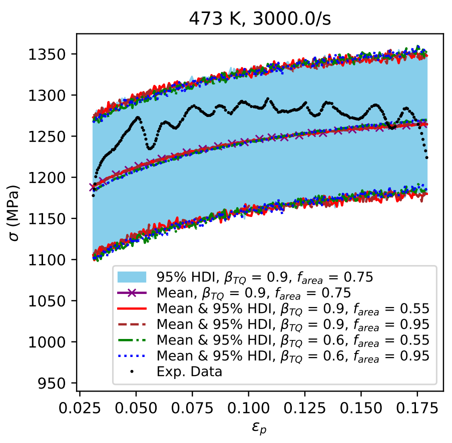
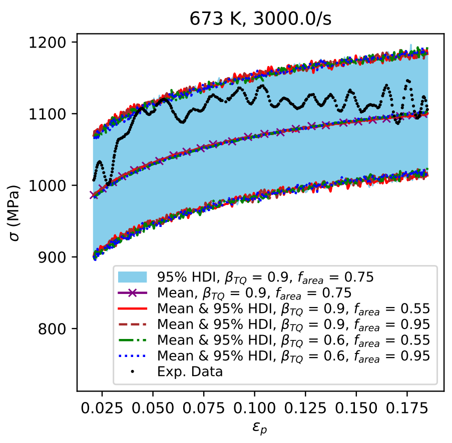


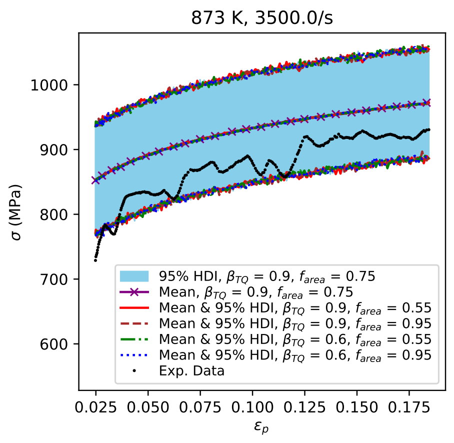
Fig. 21 Stress-strain data for initial sample temperatures of 298 K, along with estimates of the mean and the 95% HDI for PPDs generated from samples of PyStan MCMC runs for the Zerilli-Armstrong (BCC) model fit to the same data used for the Johnson-Cook model. The 95% HDI for β_{TQ} = 0.9 and f_{area} = 0.75 is plotted as a shaded region between the minimum and maximum of the HDI.



(a)



(b)



(c)

Fig. 22 Stress-strain data for high initial sample temperatures along with estimates of the mean and the 95% HDI for PPDs generated from samples of PyStan MCMC runs for the Zerilli-Armstrong (BCC) model fit to the same data used for the Johnson-Cook model. The 95% HDI for β_{TQ} = 0.9 and f_{area} = 0.75 is plotted as a shaded region between the minimum and maximum of the HDI.

plastic strains beyond 3%. This trend is due to thermal softening, and it cannot be captured by any model of the form $\sigma_{model} = a_{\dot{\epsilon}_p}(T) + b_{\dot{\epsilon}_p}(T)\epsilon_p^n$—which describes both the Johnson-Cook and Zerilli-Armstrong (BCC) models—if temperature T is treated as constant. This is because if $a_{\dot{\epsilon}_p}(T)$ and $b_{\dot{\epsilon}_p}(T)$ are constant, the model predicts a monotonically increasing σ_{model} with increasing ϵ_p, which obviously does not account for the experimentally determined trend. Figure 18b shows the Zerilli-Armstrong (BCC) model predicting a downward slope of the flow stress with increasing plastic strain, especially for $\beta_{TQ} = 0.9$, because the temperature rise has been at least approximately taken into account.

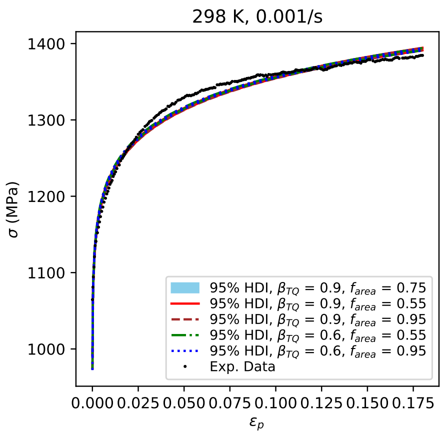
9.3 Comparison of PFP to Experimental Data Trends

As with the PPDs, for a fixed $\mathbf{e}_j^{i_c}$, each of the MCMC samples of $\boldsymbol{\theta}_{mdl}$ is substituted into $\sigma_{mdl}(\mathbf{e}_j^{i_c}, \boldsymbol{\theta}_{mdl})$. Thus, one obtains a set of samples from the PFP for a given $\mathbf{e}_j^{i_c}$, which is the same size as the set of MCMC samples. The bounds of the 95% HDI of the PFPs for all of the values of $\mathbf{e}_j^{i_c}$, for the Johnson-Cook and Zerilli-Armstrong (BCC) models under various fitting conditions, are shown in Figs. 23–31.

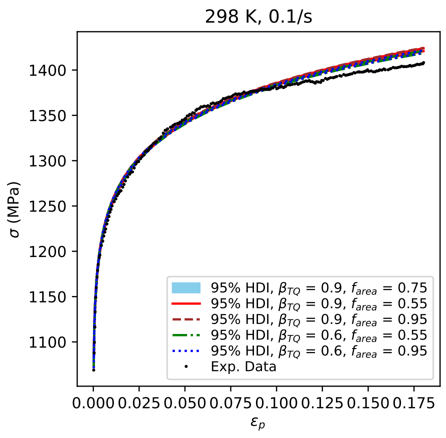
Since the PFPs do not include the effects of the nuisance parameters, they are much narrower than their corresponding PPDs, and unlike the PPDs, they do not envelop most of the experimental data. For the noisier data from experiments at high strain rates, this is to be expected, since noise is primarily taken into account through the nuisance parameters, whose influence is excluded here. Ideally, the plots of this noisy data should oscillate around the 95% HDI of the PFPs, as it approximately does in, for example, Fig. 28d. By contrast, the less noisy data taken from quasi-static experiments should be largely within the 95% HDI of the PFPs, provided that the Bayesian model is correct in attributing all the discrepancy between the model and experimental results to measurement noise. However, the results shown in Figs. 23a–b, 25a–b, 27a, 28a–b, and 30a–b indicate that the Bayesian model is *not* correct in this regard. This is consistent with the findings in Section 9.2 that the means of the PPDs do not fully track the experimental trends.

9.4 Evaluation of Output from Approximate Interval Predictor Model

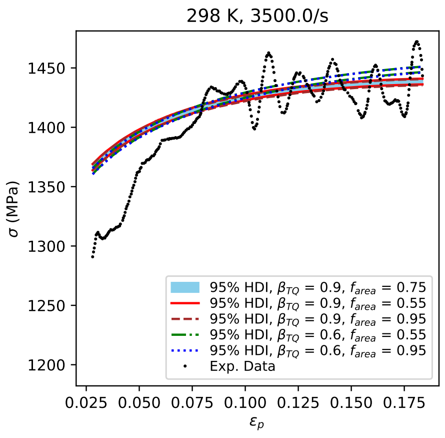
The vast majority of the experimental flow stress values—99.9% for the Johnson-Cook model and 99.7% for the Zerilli-Armstrong (BCC) model—are between the values of $\sigma_{min}(\mathbf{e}, \boldsymbol{\Theta})$ and $\sigma_{max}(\mathbf{e}, \boldsymbol{\Theta})$ calculated from the parameter bounds in Tables 17–20, as can be seen in Figs. 32–35. However, only a subset of model pa-



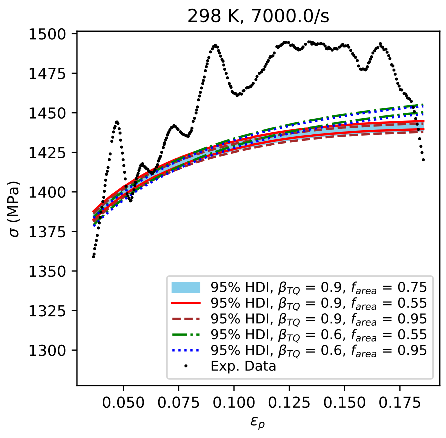
(a)



(b)

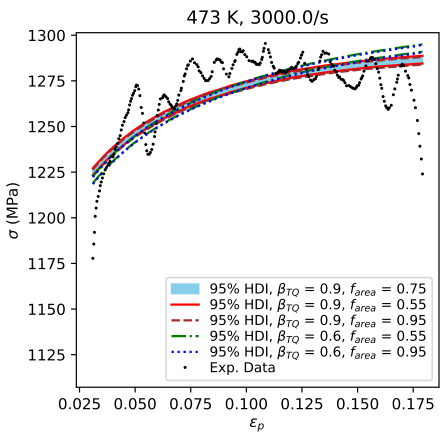


(c)

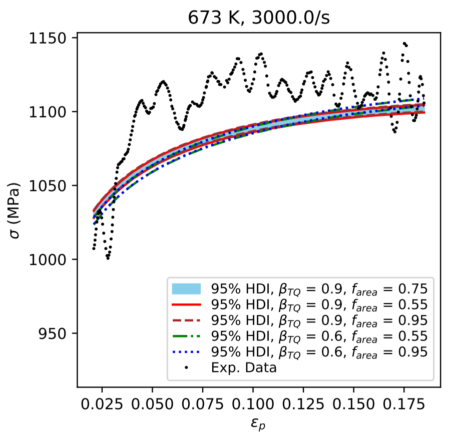


(d)

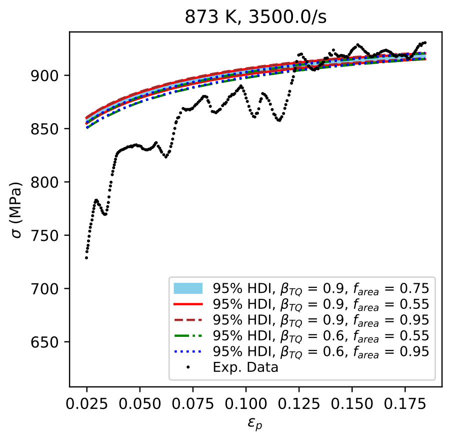
Fig. 23 Stress-strain data for initial sample temperatures of 298 K, along with estimates of the 95% HDI for PFPs generated from samples of PyStan MCMC runs for the Johnson-Cook model with weakly informative priors. The 95% HDI for $\beta_{TQ} = 0.9$ and $f_{area} = 0.75$ is plotted as a shaded region between the minimum and maximum of the HDI.



(a)

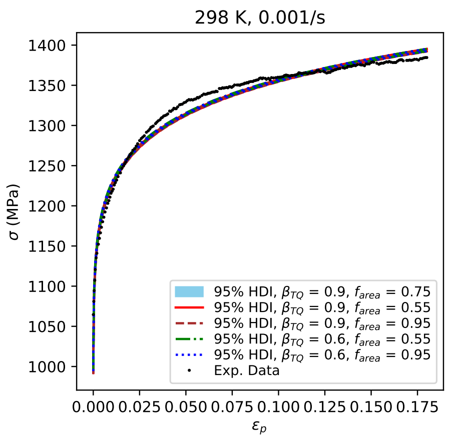


(b)

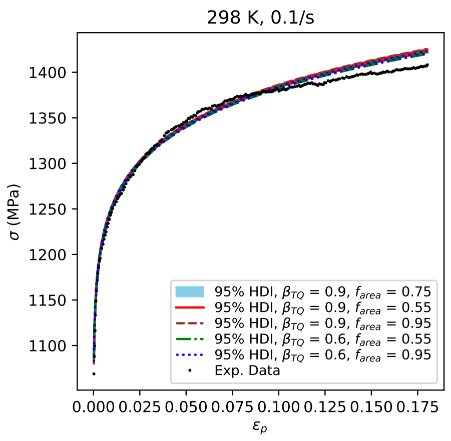


(c)

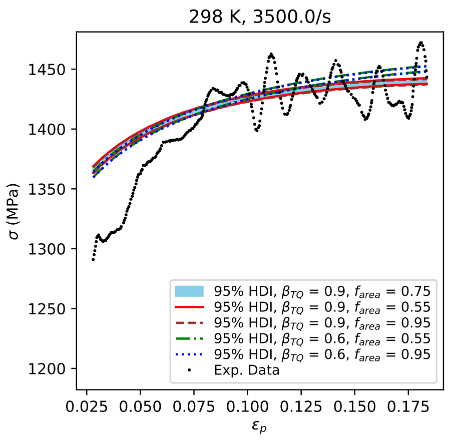
Fig. 24 Stress-strain data for high initial sample temperatures along with estimates of the 95% HDI for PFPs generated from samples of PyStan MCMC runs for the Johnson-Cook model with weakly informative priors. The 95% HDI for $\beta_{TQ} = 0.9$ and $f_{area} = 0.75$ is plotted as a shaded region between the minimum and maximum of the HDI.



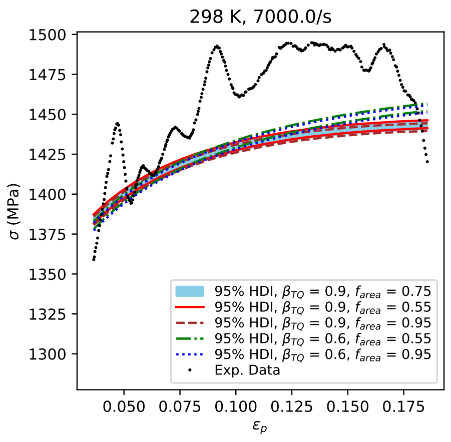
(a)



(b)

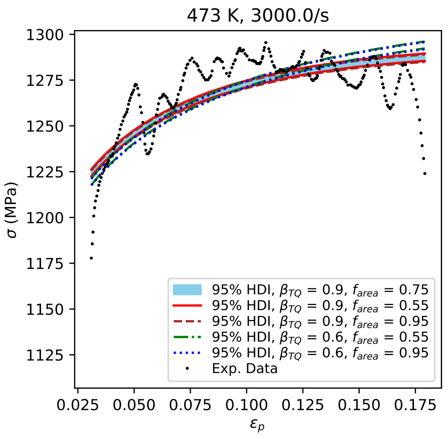


(c)

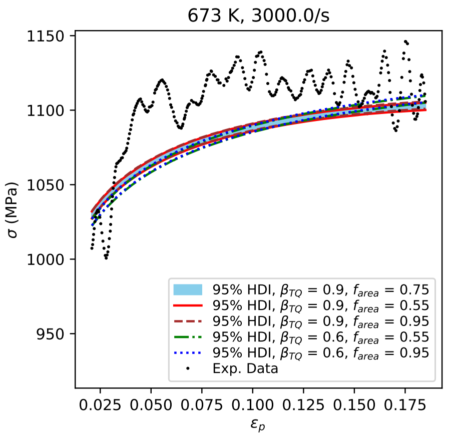


(d)

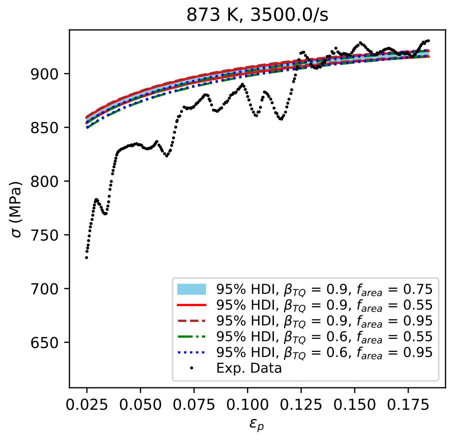
Fig. 25 Stress-strain data for initial sample temperatures of 298 K, along with estimates of the 95% HDI for PFPs generated from samples of PyStan MCMC runs for the Johnson-Cook model with a strongly informative prior for A. The 95% HDI for $\beta_{TQ} = 0.9$ and $f_{area} = 0.75$ is plotted as a shaded region between the minimum and maximum of the HDI.



(a)

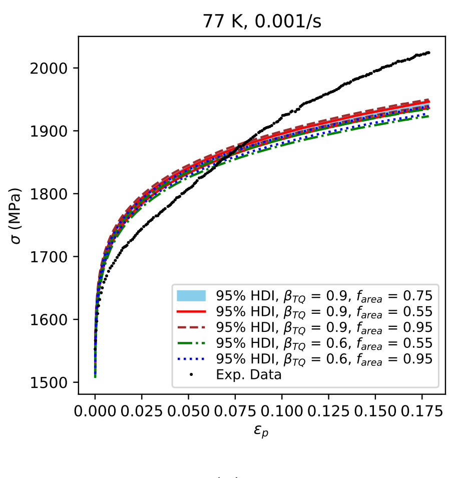


(b)

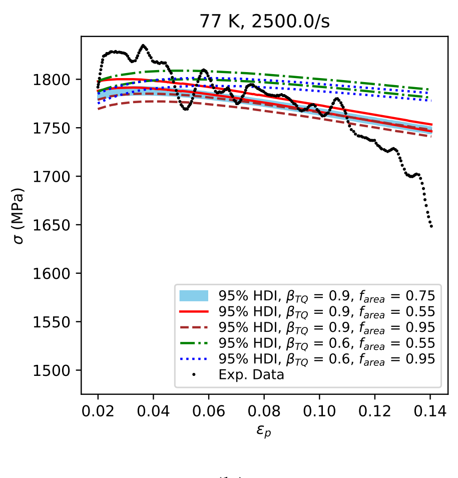


(c)

Fig. 26 Stress-strain data for high initial sample temperatures along with estimates of the 95% HDI for PFPs generated from samples of PyStan MCMC runs for the Johnson-Cook model with a strongly informative prior for A. The 95% HDI for $\beta_{TQ} = 0.9$ and $f_{area} = 0.75$ is plotted as a shaded region between the minimum and maximum of the HDI.



(a)

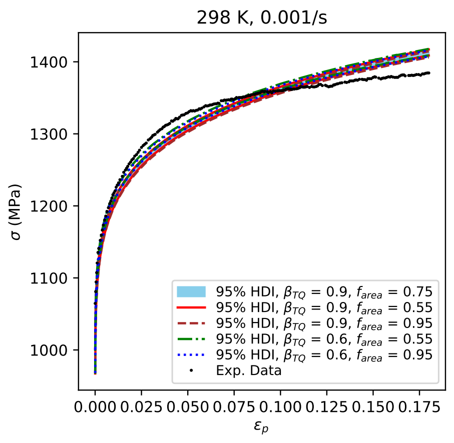


(b)

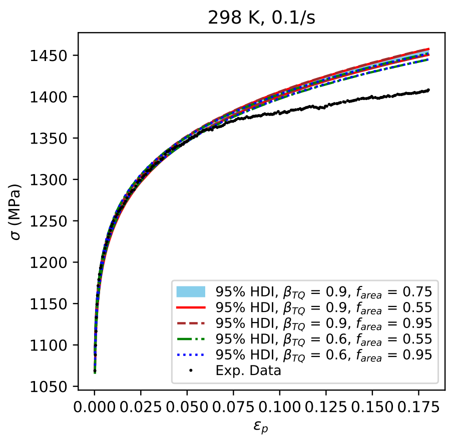
Fig. 27 Stress-strain data for initial sample temperatures of 77 K, along with estimates of the 95% HDI for PFPs generated from samples of PyStan MCMC runs for the Zerilli-Armstrong (BCC) model fit data for all available temperatures. The 95% HDI for $\beta_{TQ} = 0.9$ and $f_{area} = 0.75$ is plotted as a shaded region between the minimum and maximum of the HDI.

rameters have significant differences between their upper and lower bounds. For the Johnson-Cook model, these are n, C, and m.* For the Zerilli-Armstrong (BCC) model, only the intervals for parameters C_4 and n have significant non-zero width. This does not mean that it makes sense to say, for example, that there is no uncertainty in parameters A and B. Rather, it simply means that varying those parameters is not necessary to obtain a set $\mathbf{\Theta}$) that accounts for the vast bulk of the available experimental data. For example, two very different values for A in the Bayesian fits for the Johnson-Cook model (shown in 3) appeared to account for the experimental data about equally well, indicating that varying A does not help much to account for the data in the first place. Similarly, the curvatures of the predicted stress-strain curves according to the Johnson-Cook model are determined largely from B and n, and varying n appears to be enough to account for the curvatures. It is likely most appropriate to say that given certain baseline values of the model parameters, only a subset of the parameters need to be varied to account for discrepancies between model predictions and experimental results.

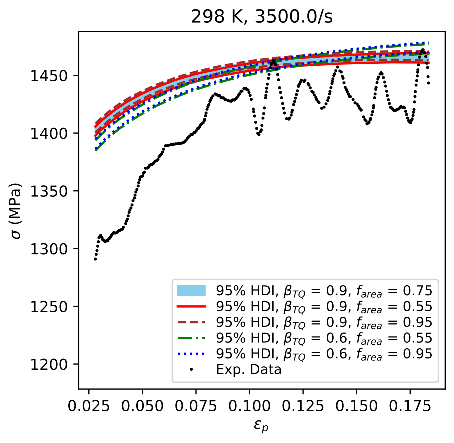
*For the case where $\beta_{TQ} = 0.9$ and $f_{area} = 0.95$, the differences between the upper and lower bounds for A and B are practically negligible and almost certainly due to the inexactness of floating-point arithmetic.



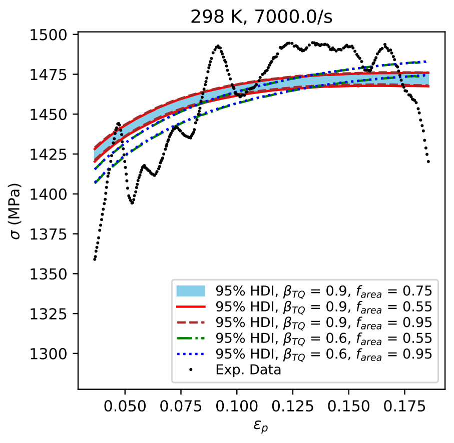
(a)



(b)

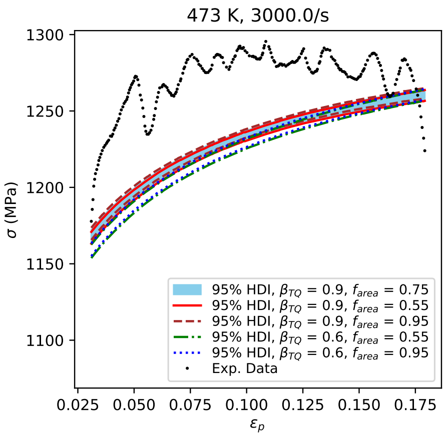


(c)

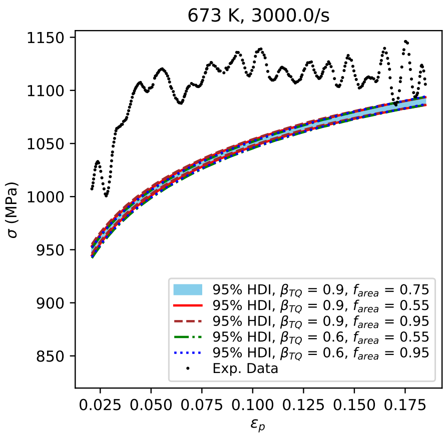


(d)

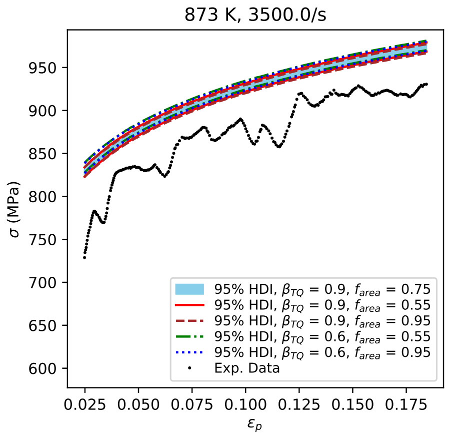
Fig. 28 Stress-strain data for initial sample temperatures of 298 K, along with estimates of the 95% HDI for PFPs generated from samples of PyStan MCMC runs for the Zerilli-Armstrong (BCC) model fit data for all available temperatures. The 95% HDI for $\beta_{TQ} = 0.9$ and $f_{area} = 0.75$ is plotted as a shaded region between the minimum and maximum of the HDI.



(a)

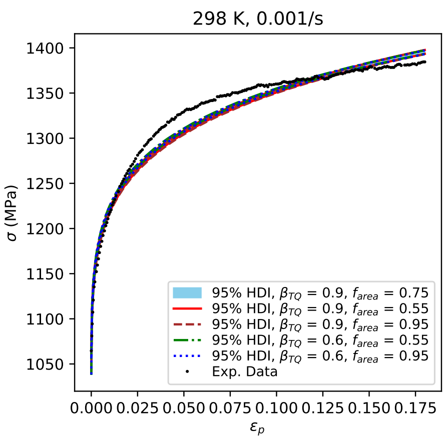


(b)

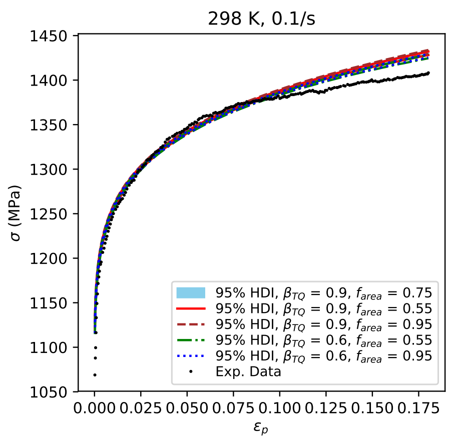


(c)

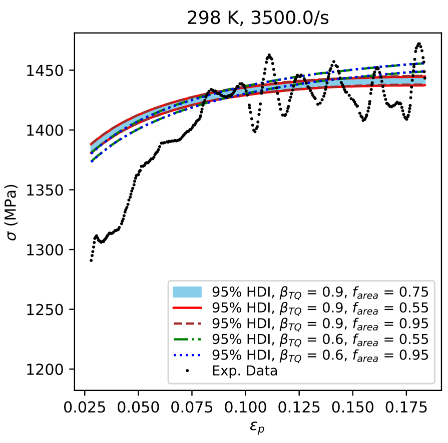
Fig. 29 Stress-strain data for high initial sample temperatures along with estimates of the 95% HDI for PFPs generated from samples of PyStan MCMC runs for the Zerilli-Armstrong (BCC) model fit to data for all available temperatures. The 95% HDI for $\beta_{TQ} = 0.9$ and $f_{area} = 0.75$ is plotted as a shaded region between the minimum and maximum of the HDI.



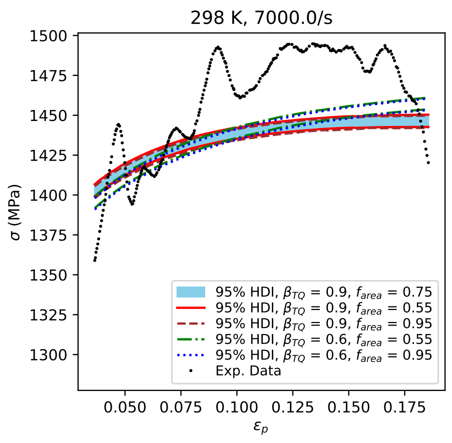
(a)



(b)

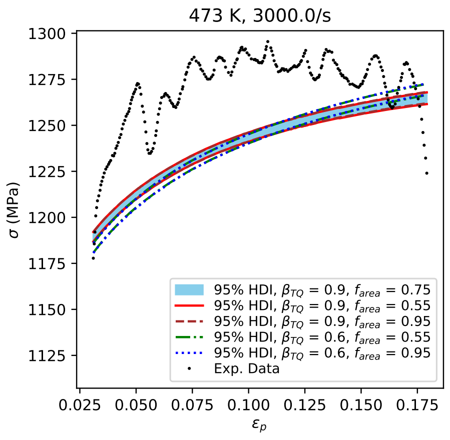


(c)

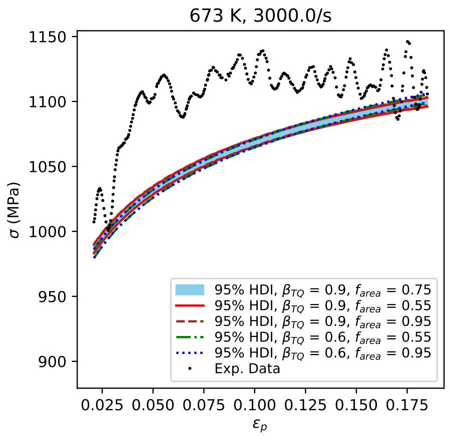


(d)

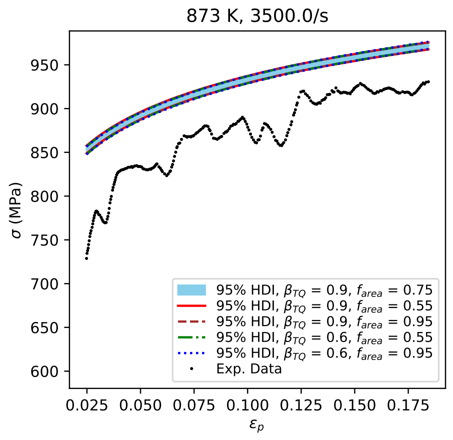
Fig. 30 Stress-strain data for initial sample temperatures of 298 K, along with estimates of the 95% HDI for PFPs generated from samples of PyStan MCMC runs for the Zerilli-Armstrong (BCC) model fit to the same data used for the Johnson-Cook model. The 95% HDI for β_{TQ} = 0.9 and f_{area} = 0.75 is plotted as a shaded region between the minimum and maximum of the HDI.



(a)

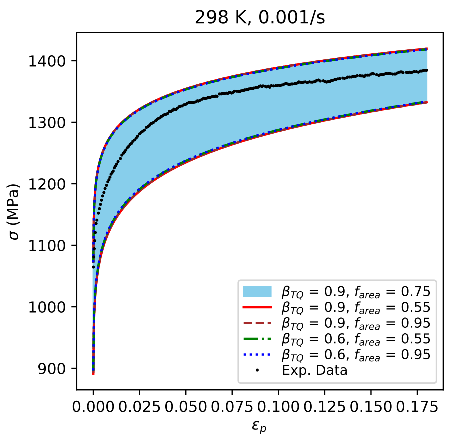


(b)

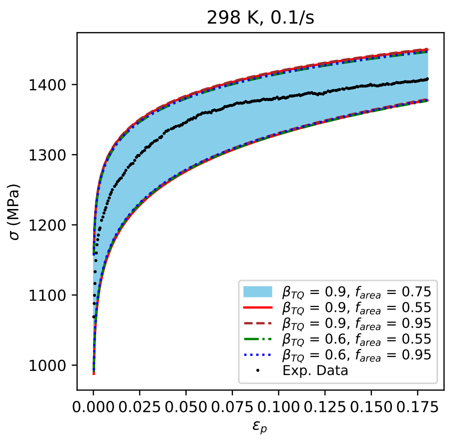


(c)

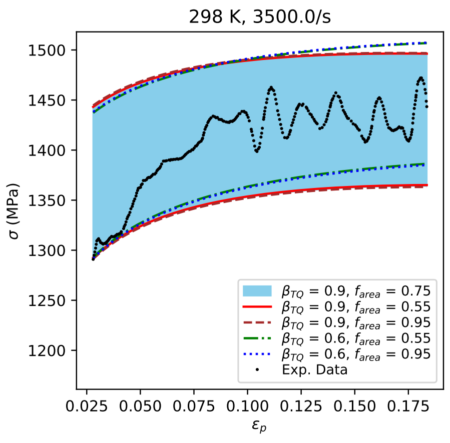
Fig. 31 Stress-strain data for high initial sample temperatures along with estimates of the 95% HDI for PFPs generated from samples of PyStan MCMC runs for the Zerilli-Armstrong (BCC) model fit to the same data used for the Johnson-Cook model. The 95% HDI for β_{TQ} = 0.9 and f_{area} = 0.75 is plotted as a shaded region between the minimum and maximum of the HDI.



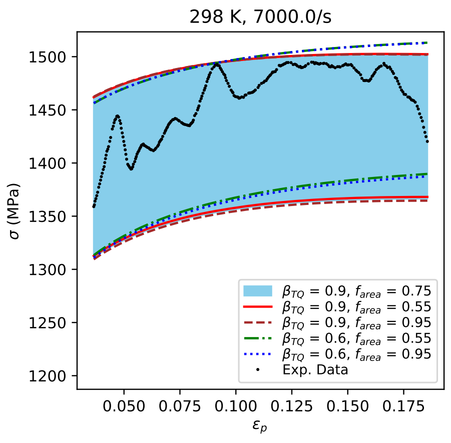
(a)



(b)

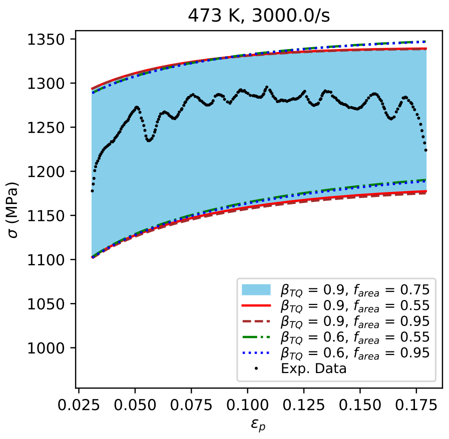


(c)

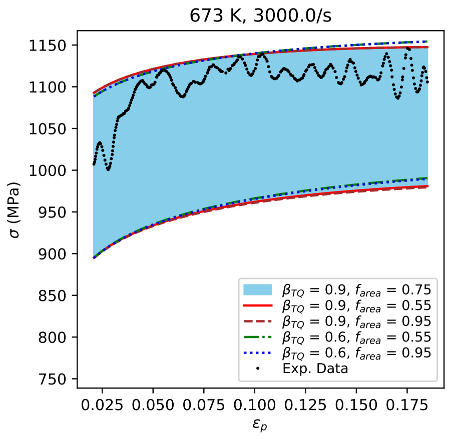


(d)

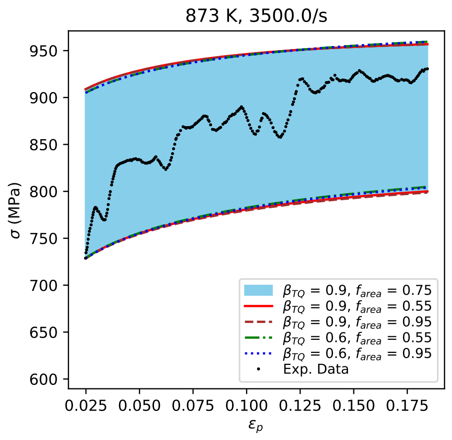
Fig. 32 Stress-strain data for initial sample temperatures of 298 K, along with estimates of the minimum and maximum flow stress values generated from parameter bounds for the Johnson-Cook model. For $\beta_{TQ} = 0.9$ and $f_{area} = 0.75$, the region between the flow stress values is plotted as a shaded region.



(a)

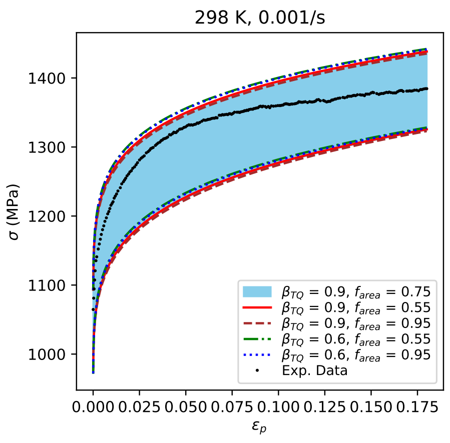


(b)

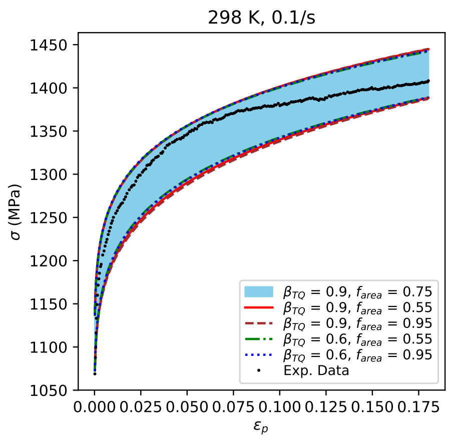


(c)

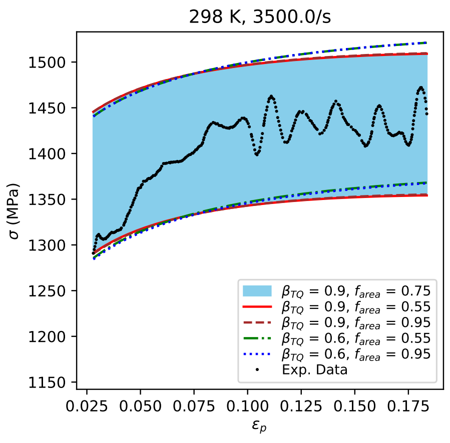
Fig. 33 Stress-strain data for high initial sample temperatures along with estimates of the minimum and maximum flow stress values generated from parameter bounds for the Johnson-Cook model. For $\beta_{TQ} = 0.9$ and $f_{area} = 0.75$, the region between the flow stress values is plotted as a shaded region.



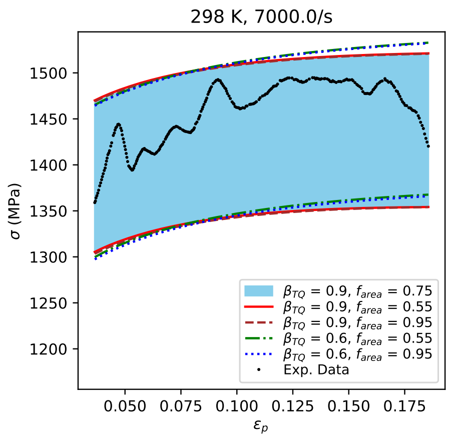
(a)



(b)

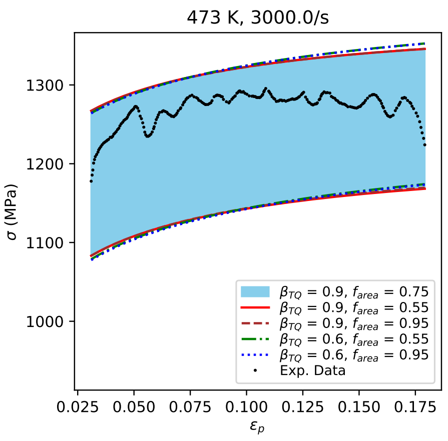


(c)

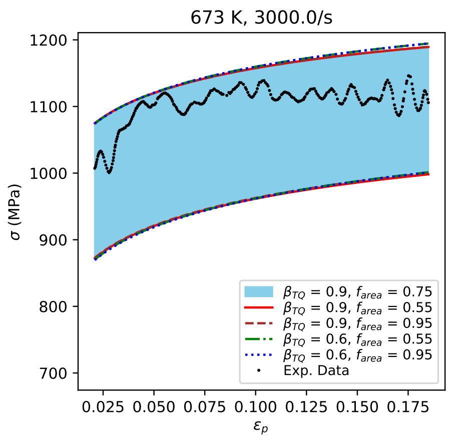


(d)

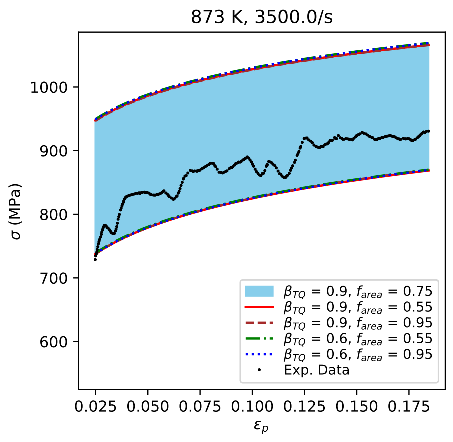
Fig. 34 Stress-strain data for initial sample temperatures of 298 K, along with estimates of the minimum and maximum flow stress values generated from parameter bounds for the Zerilli-Armstrong (BCC) model. For $\beta_{TQ} = 0.9$ and $f_{area} = 0.75$, the region between the flow stress values is plotted as a shaded region.



(a)



(b)



(c)

Fig. 35 Stress-strain data for high initial sample temperatures along with estimates of the minimum and maximum flow stress values generated from parameter bounds for the Zerilli-Armstrong (BCC) model. For $\beta_{TQ} = 0.9$ and $f_{area} = 0.75$, the region between the flow stress values is plotted as a shaded region.

9.5 Predictions of Yield Strength

For the Johnson-Cook model, parameter A is an estimate of the yield strength. For the Zerilli-Armstrong (BCC) model, no single parameter of the model provides an estimate. Instead, for this model, the quasi-static yield strength is estimated as the mean of the PPD for $\epsilon_p = 0$ and $\dot{\epsilon}_p = 0.001/\text{s}$. Estimated yield strengths for various values of β_{TQ} and f_{area} are shown in Table 21 for the Zerilli-Armstrong (BCC) model fit to the MIDAS data for all available temperatures, and in Table 22 for the Zerilli-Armstrong (BCC) model fit to the same data used with the Johnson-Cook model.

Table 21 Yield strengths, estimated as the mean of a PPD for $\epsilon_p = 0$ and $\dot{\epsilon}_p = 0.001/s$, for PPDs generated from samples of RStan MCMC runs for the Zerilli-Armstrong (BCC) model fit to data for all available temperatures and the β_{TQ} and f_{area} values from Table 1

β_{TQ}	f_{area}	Yield stress (MPa)
0.9	0.75	983.291
0.9	0.55	983.784
0.9	0.95	984.853
0.6	0.55	982.384
0.6	0.95	981.5

Table 22 Yield strengths, estimated as the mean of a PPD for $\epsilon_p = 0$ and $\dot{\epsilon}_p = 0.001/s$, for PPDs generated from samples of RStan MCMC runs for the Zerilli-Armstrong (BCC) model fit to the same data used for the Johnson-Cook model and the β_{TQ} and f_{area} values from Table 1

β_{TQ}	f_{area}	Yield stress (MPa)
0.9	0.75	1051.94
0.9	0.55	1050.09
0.9	0.95	1053.12
0.6	0.55	1045.7
0.6	0.95	1046.62

A strong prior on at least one parameter appears to be necessary for a strength model to have a reasonable estimate of the yield strength. Without such a prior, the Johnson-Cook model tends to underpredict the yield strength (i.e., parameter A in the model) by about 18%–27%, as can be seen by comparing parameter A in Tables 3 and 5 with the estimated yield strength of RHA discussed in Section 5, about 700 MPa. The Zerilli-Armstrong (BCC) model overpredicts the yield strength by about 40%–50%, as can be seen by comparing the yield strengths shown in Tables 21 and 22 with the aforementioned estimated yield strength of RHA. Unfortunately, there does not appear to be a parameter in the Zerilli-Armstrong model for which a physically based strong prior can be set, except perhaps for C_0, on which little if any information appears to be available.

9.6 Effects of Uncertainties in Temperature Rise Estimation

The approximation of sample temperature rise discussed in Section 4 involves two parameters with substantial uncertainties, β_{TQ} and f_{area}. Of the two parameters, β_{TQ} appears to have the larger effect on the fits.

In Figs. 6–9, there are usually two clusters of marginal PDFs for each model parameter, one for $\beta_{TQ} = 0.6$ and one for $\beta_{TQ} = 0.9$. There are some exceptions to this. For example, in the Johnson-Cook model with a strongly informative prior for A, the marginal PDFs of A, B, or n for various values of β_{TQ} and f_{area} (in Fig. 7) nearly overlap, showing little sensitivity to either β_{TQ} or f_{area}. This makes sense, since 1) the marginal posterior PDF of A is largely fixed by its prior and 2) there is a high correlation between parameters A, B, and n, as shown in Tables 11 and 12. Parameter C_0 of the Zerilli-Armstrong (BCC) model also appears insensitive to β_{TQ} and f_{area}, for both the fits to all MIDAS data and the fits to data for initial sample temperatures of 298 K and above (the same data used for the Johnson-Cook model). For the former set of fits, this insensitivity appears to be due to the marginal posterior PDF of C_0 being largely dictated by the prior on C_0, rather than the data. For the latter set of fits, there appears to be some trend driving C_0 toward zero, though the reason for this trend is unclear. There are also some parameters, such as C for the Johnson-Cook model and C_1 and C_4 for the Zerilli-Armstrong (BCC) model, that show more sensitivity to both β_{TQ} and f_{area}.

The mean and 95% HDI of the PPDs shown in Figs. 14–22, as well as the minimum and maximum flow stresses shown in Figs. 32–35 estimated from the approximate IPM approach, are almost completely insensitive to f_{area}, while usually showing some sensitivity to β_{TQ}. Generally, all the curves pertaining to $\beta_{TQ} = 0.6$ overlap each other, and all the curves pertaining to $\beta_{TQ} = 0.9$ overlap each other, but the two sets of curves often do not overlap. One apparent exception to this is for the cases where the initial sample temperature is 77 K and the strain rate is 2500/s, shown in Fig. 18b, which do show some sensitivity to f_{area}. This may be because the stress at the beginning of the stress-strain curve for this initial sample temperature and strain rate, 1791 MPa, is relatively high compared to that of the other curves, about 25% higher than the stress at the beginning of the stress-strain curve for sample temperature 298 K and strain rate 7000/s. Thus, for the low-temperature curve, a change in f_{area} represents a larger change in the area under the missing part of the curve than it does for other curves.

While the uncertainties in temperature rise estimation may affect the values of the model parameters and model predictions, they appear to have little effect on interactions between parameters. This can be seen from the correlation matrices, shown in Tables 9–16, which are insensitive to both β_{TQ} and f_{area}.

10. Conclusions

This report has discussed how to mathematically describe a Bayesian model associated with a material strength model and related experimental data. It has given examples of how one may represent the priors in a Bayesian model, which represent background knowledge, and how priors may be weakly or strongly informative. It has discussed how the likelihood of a Bayesian model may be constructed from assumptions of how the experimental data are expected to deviate from the model. It has also discussed how to translate a mathematical description of a Bayesian model into forms usable by software tools, such as a specification file written in the Stan language or a Python function to be used with PyMC3. In this report are also examples of how uncertainties in model parameters may be reported, such as histograms to visualize the marginal PDFs of model parameters and tables of moments of the distributions with these PDFs. The correlations between the random distributions of these model parameters are also reported, and in forms suitable as input to tools used for uncertainty propagation analysis, such as Dakota.[53]

Evaluations are presented of the quality of Bayesian fits of particular strength models (namely, the Johnson-Cook and Zerilli-Armstrong [BCC] models) to experimental data. The evaluations that have been done include comparing priors to their associated posteriors, comparing PPDs and PFPs to experimental data, and comparing what the strength models predict to be the yield strength to an experimentally obtained yield strength. Other evaluations could have been done, such as cross-validation,[18] where one fits a model to a subset of the data (e.g., all but one of the stress-strain curves) and checks how well the resulting fitted model predicts the experimental data that has not been used to fit the data (e.g. the stress-strain curve that is left out). However, the evaluations that have been done—particularly the comparision of PPDs and PFPs to the data—already show significant discrepancies between the experimental data and what the strength models discussed in this report are able to predict, even with the PDFs of the parameters taken into account. This indicates the importance of publishing the nuisance parameters, even if they cannot be directly be input to uncertainty propagation analyses, since they provide an indication of model discrepancies that future researchers can use to estimate the degree of trust to put in such analyses.

Also discussed in this report is an alternative to the Bayesian approach, based on an approximate IPM approach, that can obtain bounds for model parameters. A caveat with this approach is that if varying a model parameter about its baseline values does little to account for discrepancies between model predictions and experimental results, then the estimated lower and upper bounds for that parameters may be the same. In spite of this, though, the estimated bounds on the model parameters lead to predictions that encompasss the vast majority of the experimental data.

It is hoped that this report will be useful to future researchers who do model fits, both within and outside ARL, so that they can present their results in a fashion that is more useful for uncertainty quantification.

11. References

1. 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.

2. Hubert W. Meyer J, Kleponis DS. An analysis of parameters for the Johnson-Cook strength model for 2-in-thick rolled homogeneous armor. Aberdeen Proving Ground (MD): Army Research Laboratory (US); 2001 June. Report No.: ARL-TR-2528.

3. Whittington WR, Oppedal AL, Turnage S, Hammi Y, Rhee H, Allison PG, Crane CK, Horstemeyer MF. Capturing the effect of temperature, strain rate, and stress state on the plasticity and fracture of rolled homogeneous armor (RHA) steel. Materials Science and Engineering: A. 2014;594:82–88.

4. 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.

5. Stan Development Team. Stan. c2018 [accessed 2018 Mar]. `http://mc-stan.org/`.

6. PyMC Development Team. PyMC3 documentation. c2018 [accessed 2018 Mar]. `http://docs.pymc.io/`.

7. Crespo LG, Kenny SP, Giesy DP. Calibration of predictor models using multiple validation experiments. In: 17th AIAA Non-Deterministic Approaches Conference; (AIAA SciTech Forum; no. AIAA 2015-0659) American Institute of Aeronautics and Astronautics; 2015.

8. 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.

9. SciPy developers. SciPy. c2018 [accessed 2018 May]. `http://www.scipy.org/`.

10. Ramsey JJ. Quantifying uncertainties in parameterizations of strength models of rolled homogeneous armor: part 2, R-based workflow. Aberdeen Proving Ground (MD): Army Research Laboratory (US); 2019 Sep. Report No.: ARL-TR-8827.

11. Ramsey JJ. Quantifying uncertainties in parameterizations of strength models of rolled homogeneous armor: part 3, Python-based workflow. Aberdeen Proving Ground (MD): Army Research Laboratory (US); 2019 Sep. Report No.: ARL-TR-8828.

12. R Core Team. R: A language and environment for statistical computing—reference index. c2018 [accessed 2018 May]. `https://cran.r-project.org/doc/manuals/r-release/fullrefman.pdf`.

13. Guo J, Gabry J, Goodrich B, Lee D, Sakrejda K, Trustees of Columbia University, Sklyar O, The R Core Team, Oehlschlaegel-Akiyoshi J, Wickham H, de Guzman J, Fletcher J, Heller T, Niebler E. Package "rstan". c2018 [accessed 2018 Mar]. `https://cran.r-project.org/web/packages/rstan/rstan.pdf`.

14. Berkelaar M et al. lpSolve: Interface to 'Lp_solve' v. 5.5 to solve linear/integer programs. 2015 [accessed 2019 Mar]. `https://cran.r-project.org/package=lpSolve`.

15. Python Software Foundation. Welcome to python.org. c2018 [accessed 2018 May]. `https://www.python.org`.

16. PyStan developers. PyStan: the Python interface to Stan, API. c2018 [accessed 2018 Mar]. `http://pystan.readthedocs.io/en/latest/api.html`.

17. Kruschke JK. Doing Bayesian data analysis: a tutorial with R, JAGS, and Stan. 2nd ed. Waltham (MA): Academic Press; 2015.

18. Gelman A, Carlin JB, Stern HS, Dunson DB, Vehtari A, Rubin DB. Bayesian data analysis. 3rd ed. Boca Raton (FL): CRC Press; 2013.

19. 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.

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

21. Chowdhary K, Najm HN. Data free inference with processed data products. Statistics and Computing. 2016;26(1):149–169.

22. Betancourt M. A conceptual introduction to Hamiltonian Monte Carlo. c2017 [accessed 2018 Mar]. `https://arxiv.org/abs/1701.02434`.

23. Hoffman MD, Gelman A. The no-U-turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. Journal of Machine Learning Research. 2014;15(1).

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

25. 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`.

26. Walley SM, Proud WG, Rae PJ, Field JE. Comparison of two methods of measuring the rapid temperature rises in split Hopkinson bar specimens. Review of Scientific Instruments. 2000;71(4):1766–1771.

27. Kapoor R, Nemat-Nasser S. Determination of temperature rise during high strain rate deformation. Mechanics of Materials. 1998;27(1):1–12.

28. Børvik T, Dey S, Clausen AH. Perforation resistance of five different high-strength steel plates subjected to small-arms projectiles. International Journal of Impact Engineering. 2009;36(7):948–964.

29. Rittel D, Zhang LH, Osovski S. The dependence of the Taylor-Quinney coefficient on the dynamic loading mode. Journal of the Mechanics and Physics of Solids. 2017;107:96–114.

30. 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.

31. Kerley GI. Multiphase equation of state for iron. Albuquerque (NM): Sandia National Laboratories; 1993 Feb. Report No.: SAND93-0027.

32. Austin JB. Heat capacity of iron: a review. Industrial & Engineering Chemistry. 1932;24(11):1225–1235.

33. 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.

34. Zerilli FJ, Armstrong RW. Dislocation-mechanics-based constitutive relations for material dynamics calculations. Journal of Applied Physics. 1987;61(5):1816–1825.

35. 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.

36. Ling Y, Mullins J, Mahadevan S. Selection of model discrepancy priors in Bayesian calibration. Journal of Computational Physics. 2014;276:665–680.

37. Bishop CM. Pattern recognition and machine learning. New York (NY): Springer; 2006.

38. Gelman A, Lee D, Guo J. Stan: A probabilistic programming language for Bayesian inference and optimization. Journal of Educational and Behavioral Statistics. 2015;40(5):530–543.

39. Stan Development Team. Stan modeling language user's guide and reference manual. 2017 Dec.

40. MIL-A-12560H(MR). Armor plate, steel, wrought, homogeneous (for use in combat vehicles and for ammunition testing). Watertown (MA): US Army Laboratory Command, Materials Technology Laboratory; Nov 1990.

41. Salvatier J, Wiecki TV, Fonnesbeck C. Getting started with PyMC3. c2018 [accessed 2018 Mar]. http://docs.pymc.io/notebooks/getting_started.

42. Stan Development Team. CmdStan interface user's guide: Version 2.18.0. 2018 July.

43. Betancourt M. Robust RStan workflow. c2017 [accessed 2018 July]. http://mc-stan.org/users/documentation/case-studies.html#robust-rstan-workflow.

44. Betancourt M. Robust PyStan workflow. c2017 [accessed 2018 Mar]. http://mc-stan.org/users/documentation/case-studies.html#robust-pystan-workflow.

45. Carpenter B. Generalizing the Stan language for Stan 3. c2017 [accessed 2018 Apr]. http://discourse.mc-stan.org/t/generalizing-the-stan-language-for-stan-3/1599.

46. SymPy development team. SymPy. c2018 [accessed 2019 Mar]. http://www.sympy.org/.

47. Stan Development Team. Brief guide to stan's warnings. c2018 [accessed 2018 May]. http://mc-stan.org/misc/warnings.html.

48. PyMC Development Team. Diagnosing biased inference with divergences. c2017 [accessed 2018 July]. http://docs.pymc.io/notebooks/Diagnosing_biased_Inference_with_Divergences.html.

49. Bastos MM. UserWarning after sampling. c2017 [accessed 2018 July]. https://discourse.pymc.io/t/userwarning-after-sampling/118.

50. Ramsey JJ. Issue #427: Integrate Betancourt's stan_utility functions into PyStan #433. c2018 [accessed 2018Mar]. https://github.com/stan-dev/pystan/pull/433.

51. Gelman A, Rubin DB. Inference from iterative simulation using multiple sequences. Statistical Science. 1992;7(4):457–472.

52. Ramsey JJ. split_potential_scale_reduction can yield spurious nan values #441. c2018 [accessed 2018 Mar]. https://github.com/stan-dev/pystan/issues/441.

53. Adams BM et al. Dakota, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: Version 6.8 reference manual. Albuquerque (NM): Sandia National Laboratories; 2018 May.

54. National Institute of Standards and Technology (NIST). NIST/SEMATECH e-handbook of statistical methods. c2018 [accessed 2018 June]. https://www.itl.nist.gov/div898/handbook/index.htm.

55. Pandas developers. pandas.dataframe.corr. c2017 [accessed 2018 June]. http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.corr.html.

Appendix A. Probability Density Functions of Random Distributions

A random variable has a certain distribution, which defines how likely it is that an instance, or *sample*, of the variable has a certain value or be within a certain range of values. As a simple example, the distribution of an unfair coin toss might be such that the probability of heads is, say, 60%, and samples from this distribution might be $\{H, H, T, H, T, H, T, H, H, T\}$, where H is heads and T is tails. Continuous random variables, of course, do not have a discrete set of possible values, such as the heads and tails of coins, or the values one through six on the faces of dice. Rather, the distribution of random values for continuous variables is usually characterized via a probability density function (PDF).

If a is a continuous scalar quantity, then the PDF $p(a)$ might be plotted as a curve like the one shown in Fig. A-1. Intuitively, one may guess that the most probable ranges of values of a are near the maximum of $p(a)$. More generally, the probability that a is between a^{low} and a^{high} is[1]

$$P(a^{low} \leq a \leq a^{high}) = \int_{a^{low}}^{a^{high}} p(a)da \tag{A-1}$$

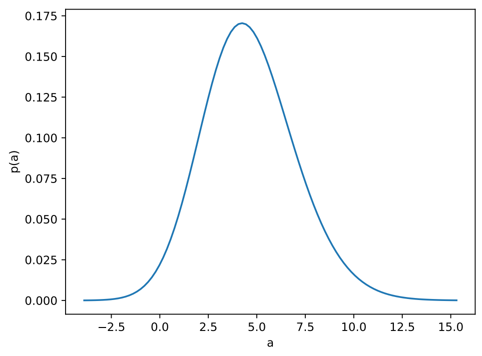


Fig. A-1 Example probability density $p(a)$

One may note that while $p(a)$ is restricted to be nonnegative, it can exceed 1. It is only the probability $P(a^{low} \leq a \leq a^{high})$ that must be less than or equal to 1. If one generates a large number of samples from a probability distribution and creates a histogram from these samples, the result tends to resemble the distribution's PDF. An example of this is shown in Fig. A-2.

[1]Grinstead CM, Snell JL. Introduction to probability. 2nd ed. Providence (RI): American Mathematical Society; 1997.

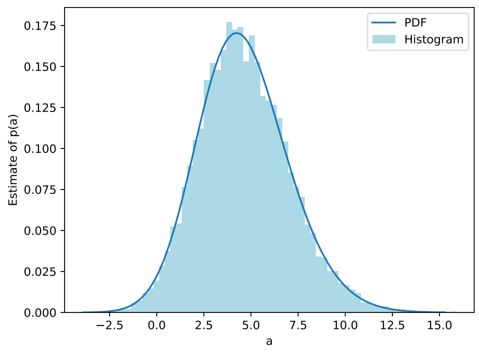


Fig. A-2 Example histogram associated with the probability density $p(a)$. The counts in each bin of the histogram have been normalized such that the area under the histogram is 1.

PDFs are not just for scalar quantities. For a variable **a** that could be a vector, matrix, tensor, and so on, the probability that **a** is within a subset of possible values **A** is

$$P(\mathbf{a} \in \mathbf{A}) = \int_{\mathbf{A}} p(\mathbf{a}) d\mathbf{a} \tag{A-2}$$

where $p(\mathbf{a})$ is the PDF of **a**. Note that by the definition of probability, as the set **A** expands to include all possible values of **a**, $P(\mathbf{a} \in \mathbf{A})$ approaches 1. Associated with the components of **a** is a marginal probability distribution, and the marginal PDF of component a_i of **a** is[2]

$$p(a_i) = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} p(\mathbf{a}) da_1 \cdots da_{i-1} da_{i+1} \cdots da_{n_{\mathbf{a}}} \tag{A-3}$$

where $n_{\mathbf{a}}$ is the number of components of **a**. Individual marginal PDFs can be visualized readily (as in, for example, Fig. A-1), even when the overall (or joint) PDF $p(\mathbf{a})$ is difficult or impossible to visualize directly.

There are several known random distributions, which are described as follows.[3] One of the simplest is the uniform distribution. For a continuous variable a that is

[2]Gelman A, Carlin JB, Stern HS, Dunson DB, Vehtari A, Rubin DB. Bayesian data analysis. 3rd ed. Boca Raton (FL): CRC Press; 2013.

[3]Grinstead CM, ibid.

equally likely to be anywhere between $[a^{min}, a^{max}]$, the PDF of a is simply

$$p_{\text{unif}}(a|a^{min}, a^{max}) = \begin{cases} 1/(a^{max} - a^{min}) & a^{min} \leq a \leq a^{max} \\ 0 & \text{otherwise} \end{cases} \tag{A-4}$$

Because a^{min} and a^{max} are parameters of the uniform distribution itself, they are placed after the "|" in the previous equation. Other parameterized distributions are the exponential, beta, and normal distributions, and these have the following PDFs.

$$p_{\text{exp}}(a|\lambda) = \begin{cases} \lambda \exp(-\lambda a) & x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{A-5}$$

$$p_{\text{beta}}(a|\alpha, \beta) = \begin{cases} a^{\alpha-1}(1-a)^{\beta-1} / \int_0^1 (a^*)^{\alpha-1}(1-a^*)^{\beta-1} da^* & 0 \leq a \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{A-6}$$

$$p_{\text{normal}}(a|\mu, SD) = \frac{1}{SD\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{a-\mu}{SD}\right)^2\right] \tag{A-7}$$

Again, parameters pertaining to the random distribution itself are placed after the "|" in the previous equations. Parameters μ and SD in particular represent the mean and standard deviation of the normal distribution. Example plots of these parameterized PDFs are shown in Fig. A-3.

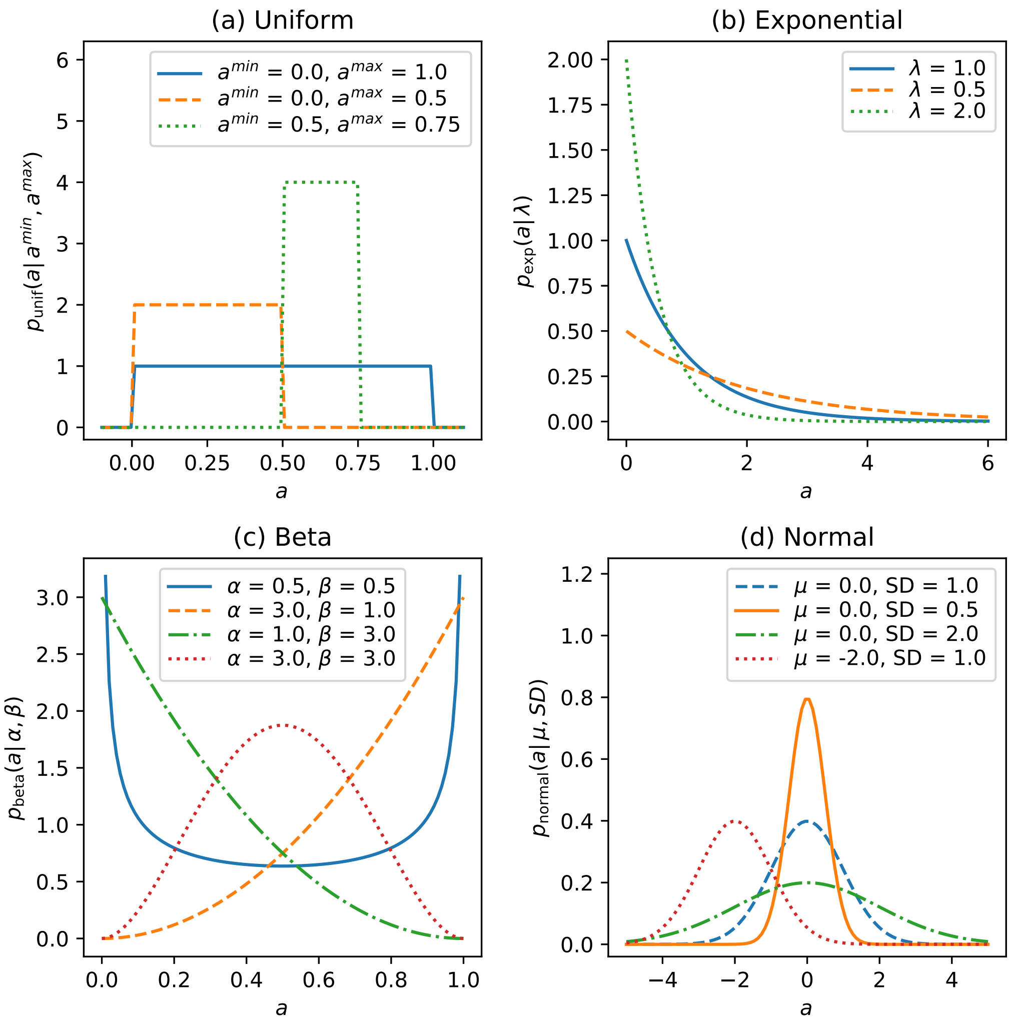


Fig. A-3 Examples of parameterized PDFs

Appendix B. Data Tables

These are tables of the data that have been used in Bayesian analyses of strength models of rolled homogeneous armor (RHA). Table B-1 contains values for the specific heat of body-centered cubic (BCC) iron—which is assumed to approximate the specific heat of RHA—as a function of temperature. In this table, the specific heat values are for constant volume, except for values for temperatures above 773 K, where only values for constant pressure are available. The specific heat values are converted from molar heat capacity values from Austin[1] using the molar mass of iron taken from the CRC Handbook,[2] 55.845 g/mol. Tables B-2 through B-10 contain the stress-strain data for RHA that comes from the Material Implementation, Database, and Analysis Source (MIDAS).[3] The original source for these data is Gray et al.,[4] who have obtained high-strain-rate data with a split Hopkinson pressure bar and low-strain-rate data (where the plastic strain rate is no greater than 1/s) with "either an Instron or an MTS testing system". However, the original published data are engineering stress and strain, while in the MIDAS database, it has been corrected to true stress and true plastic strain.[5]

Table B-1 Specific heat of BCC iron versus temperature

Temp. (K)	Spec. heat $(\mathrm{J/kg \cdot K})$	Temp. (K)	Spec. heat $(\mathrm{J/kg \cdot K})$	Temp. (K)	Spec. heat $(\mathrm{J/kg \cdot K})$	Temp. (K)	Spec. heat $(\mathrm{J/kg \cdot K})$	Temp. (K)	Spec. heat $(\mathrm{J/kg \cdot K})$
20	4.123	200	382.356	323	454.329	573	565.287	1023	1154.566
30	11.246	225	400.349	333	457.328	623	583.281	1033	1341.245
40	27.515	250	419.092	343	459.577	673	602.773	1073	877.170
50	53.230	273.1	430.338	353	461.826	723	623.016	1123	812.694
75	134.949	283	436.336	363	464.825	773	647.756	1173	778.957
100	212.920	293	442.334	373	470.823	823	718.230		
125	272.148	298	444.583	423	494.814	873	790.203		
150	322.379	303	447.582	473	519.555	923	871.172		
175	356.866	313	451.330	523	541.296	973	962.638		

[1]Austin JB. Heat capacity of iron: A review. Industrial & Engineering Chemistry. 1932;24(11):1225–1235.

[2]Rumble J, editor. CRC handbook of chemistry and physics. 98th ed. Boca Raton (FL): CRC Press; 2017.

[3]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`.

[4]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.

[5]Florando J. Lawrence Livermore National Laboratory, Livermore, CA. Personal communication, 2017.

Table B-2 Flow stress versus plastic strain of RHA for initial temperature 77 K and plastic strain rate 0.001/s

Strain	Stress (MPa)	Strain	Stress (MPa)	Strain	Stress (MPa)	Strain	Stress (MPa)	Strain	Stress (MPa)
0.000062	1552.7	0.032648	1764.6	0.067995	1854.4	0.104813	1930	0.143498	1986.7
0.00022	1566.5	0.033594	1765	0.068757	1855.3	0.105444	1931.3	0.144944	1987.5
0.000535	1582.5	0.034303	1768	0.069571	1857.4	0.107179	1930.5	0.145548	1989.3
0.000903	1597.1	0.035066	1770.6	0.070412	1857.9	0.108177	1933.9	0.146337	1989.3
0.001376	1609.2	0.035828	1771.9	0.071385	1859.2	0.109045	1934.4	0.147125	1991
0.001875	1621.7	0.036563	1774.1	0.072199	1862.2	0.109912	1938.7	0.147835	1991.4
0.002453	1631.6	0.037483	1776.2	0.072856	1864.3	0.110779	1940.8	0.149359	1994.5
0.003294	1642.4	0.039034	1779.7	0.07375	1865.2	0.111699	1943	0.150568	1995.8
0.003872	1650.2	0.03977	1781	0.074486	1867.4	0.112881	1944.3	0.151093	1995.8
0.004712	1659.2	0.040427	1782.7	0.075248	1870	0.113565	1945.2	0.151882	1996.6
0.005475	1664	0.041136	1785.3	0.076036	1872.1	0.114669	1946	0.152749	1997.9
0.006184	1669.2	0.041872	1787	0.076956	1875.1	0.115378	1946.9	0.153643	2000.5
0.007209	1676.5	0.042818	1790.1	0.077928	1875.1	0.116456	1948.6	0.15451	2001
0.008076	1681.7	0.043659	1793.5	0.079295	1876.9	0.117402	1949.1	0.155666	2002.3
0.00897	1685.1	0.044605	1794.4	0.080057	1879.5	0.118505	1951.2	0.156428	2003.1
0.009889	1689.4	0.045315	1795.7	0.08074	1881.6	0.119268	1952.1	0.157296	2003.1
0.010835	1693.8	0.046445	1800	0.081529	1882.9	0.120266	1953.4	0.158373	2004
0.011808	1698.5	0.047338	1800.9	0.08237	1884.7	0.121081	1954.2	0.159477	2006.6
0.012675	1702.4	0.048048	1803	0.083316	1887.2	0.122237	1956.8	0.16087	2007.9
0.013437	1705.8	0.048941	1805.6	0.084446	1889.8	0.123341	1958.1	0.162	2010.5
0.014199	1709.7	0.049756	1807.3	0.085287	1891.6	0.124471	1958.6	0.162604	2010.5
0.014961	1713.6	0.050492	1808.6	0.086128	1893.7	0.12526	1960.3	0.163971	2010.5
0.01575	1715.3	0.051569	1810.4	0.0876	1897.2	0.126232	1961.6	0.164838	2011.3
0.01638	1716.2	0.052332	1812.5	0.088335	1898.9	0.127204	1962.5	0.165863	2012.2
0.017064	1717.9	0.053173	1816	0.08936	1900.2	0.128098	1963.8	0.167361	2013.9
0.017852	1722.3	0.05383	1817.7	0.090438	1902.8	0.129202	1965.5	0.168255	2013.5
0.019114	1725.3	0.054618	1820.7	0.090858	1903.7	0.130279	1967.2	0.169306	2015.2
0.020086	1727.4	0.055748	1825	0.091726	1905	0.13091	1968.9	0.170252	2016.5
0.020874	1730.5	0.056799	1825.9	0.092619	1905.8	0.131725	1971.1	0.171067	2018.3
0.021584	1733.5	0.057456	1826.8	0.093513	1906.7	0.132618	1972.4	0.171881	2020.4
0.022504	1736.5	0.05835	1828.5	0.094538	1908.4	0.133538	1974.6	0.172591	2020.4
0.023266	1739.5	0.059086	1831.1	0.095773	1911.9	0.134353	1974.6	0.173564	2020
0.024159	1741.7	0.060137	1833.2	0.097218	1914	0.135167	1975.9	0.174378	2020.9
0.024869	1743.8	0.061057	1835	0.097823	1914.9	0.136035	1975.9	0.175745	2022.2
0.026051	1747.7	0.06195	1836.7	0.098795	1917.9	0.136797	1978	0.176691	2022.2
0.027234	1750.3	0.062712	1838.9	0.09961	1920.1	0.138216	1980.6	0.177479	2023.9
0.027996	1752	0.063842	1841	0.100398	1924.4	0.139083	1981.9	0.178136	2024.3
0.028653	1755.1	0.064815	1844.9	0.101292	1924.8	0.13995	1982.8	0.178583	2024.3
0.029678	1757.2	0.065524	1847.9	0.102054	1925.7	0.140844	1983.2		
0.030466	1759	0.066181	1849.7	0.102816	1926.1	0.141659	1984.9		
0.031229	1762	0.067154	1852.2	0.103657	1926.6	0.142631	1985.8		

Table B-3 Flow stress versus plastic strain of RHA for initial temperature 77 K and plastic strain rate 2500/s

Strain	Stress (MPa)	Strain	Stress (MPa)	Strain	Stress (MPa)	Strain	Stress (MPa)	Strain	Stress (MPa)
0.019951	1791.9	0.042917	1817.4	0.065427	1787.8	0.088992	1780.1	0.116886	1739.6
0.020109	1794.6	0.043495	1816.8	0.065901	1789.4	0.089491	1778.7	0.117385	1738.7
0.02032	1797.1	0.043915	1816.8	0.066295	1789.4	0.090043	1777	0.117884	1737.8
0.020504	1801	0.044336	1816.9	0.066663	1790.8	0.090673	1775	0.118331	1737
0.020742	1805.7	0.044808	1813.9	0.066873	1791.2	0.091224	1774.4	0.118804	1737
0.021005	1809.9	0.045176	1812.8	0.067188	1790.2	0.091776	1772.6	0.119277	1736.9
0.021242	1813.7	0.045517	1810.5	0.067477	1788.1	0.092301	1769.7	0.119881	1737
0.021505	1817.4	0.045937	1808.5	0.068002	1785.1	0.092957	1768.1	0.120432	1734.1
0.021664	1820.1	0.046436	1806.3	0.06829	1783.8	0.093561	1767.4	0.12101	1731.9
0.021979	1823.8	0.046829	1801.9	0.068605	1782.4	0.094271	1768.1	0.121587	1730.9
0.022426	1824.6	0.047249	1797.4	0.069156	1778.5	0.09477	1769.2	0.122086	1729.2
0.022768	1824.8	0.047747	1792.4	0.069603	1777.3	0.095401	1769.2	0.122428	1728.2
0.023162	1826.6	0.048245	1788.5	0.070049	1774.9	0.0959	1771.1	0.123347	1727.4
0.02353	1826.4	0.048665	1784.9	0.070469	1774.9	0.096347	1772.6	0.123872	1726.4
0.024055	1827.1	0.049085	1780.8	0.070864	1775.8	0.097109	1773.6	0.124608	1725.5
0.024529	1828.3	0.049557	1777.1	0.071232	1776.8	0.097793	1773.9	0.125238	1726.3
0.024949	1828.4	0.050003	1773.8	0.071626	1778.9	0.098475	1772.3	0.125817	1727.2
0.025868	1827.9	0.050476	1772.1	0.0721	1782.1	0.099105	1768.8	0.126237	1728
0.026341	1828.7	0.051053	1769.8	0.072441	1783.1	0.099683	1767.3	0.126736	1728.6
0.026893	1827.7	0.051474	1769.1	0.072993	1785.7	0.100155	1764.9	0.127393	1727.5
0.027629	1828	0.051841	1769.6	0.073309	1788.7	0.100759	1763.1	0.127944	1726.1
0.028207	1827.7	0.052262	1769.3	0.073625	1791.4	0.101022	1762.2	0.128496	1722.9
0.028653	1827.3	0.052709	1771.8	0.073862	1792.5	0.101758	1762.4	0.129046	1718.8
0.029231	1827.1	0.053129	1773.5	0.074282	1793.4	0.102152	1763.3	0.129414	1715.9
0.02973	1826.6	0.053445	1774.9	0.074781	1794.1	0.102651	1764.7	0.129886	1711.3
0.030177	1825.7	0.053787	1777.2	0.075202	1794.5	0.103151	1765.4	0.130647	1707.3
0.030702	1823.9	0.054155	1780.2	0.075911	1794.4	0.103703	1768.3	0.131145	1703.4
0.031174	1821.4	0.05455	1784.4	0.076226	1793.4	0.104019	1770.5	0.131644	1701.4
0.031699	1819.5	0.054919	1788.3	0.077014	1792	0.104308	1771.5	0.132117	1701.3
0.032198	1818.2	0.055129	1789.9	0.077618	1790.5	0.104834	1774.8	0.132432	1700.4
0.032645	1818.6	0.055629	1793.9	0.078091	1790	0.105044	1775.6	0.1328	1699.7
0.03346	1819.2	0.056076	1797.6	0.078511	1789.2	0.105465	1778.4	0.133168	1699.9
0.033985	1822	0.056497	1800.9	0.079089	1788.2	0.105964	1779.8	0.133509	1699.6
0.034327	1824.3	0.056813	1803.2	0.079483	1787.9	0.10649	1780	0.133982	1700.3
0.034748	1827.3	0.057207	1806.2	0.079877	1787.1	0.107015	1778.3	0.134376	1701.2
0.035169	1829.8	0.057707	1808.9	0.080507	1785.6	0.107724	1775.8	0.134666	1701.8
0.035537	1831.6	0.058154	1809.7	0.081111	1784.5	0.108117	1773.2	0.13506	1702.2
0.035905	1833.5	0.058627	1808.9	0.081611	1784	0.108564	1771.2	0.1359	1701.5
0.036273	1834.9	0.059047	1807.7	0.082057	1783.9	0.109194	1767.3	0.136504	1697.8
0.036667	1833.7	0.059388	1806.2	0.082556	1783.4	0.109482	1764.9	0.137081	1692.7
0.037087	1832.5	0.059755	1803.5	0.08295	1783.5	0.109928	1760.2	0.137553	1687.3
0.03756	1830.8	0.06028	1800.5	0.083633	1783.1	0.110295	1756.4	0.137841	1681.6
0.038137	1827.2	0.060727	1798.5	0.084264	1782.7	0.110846	1751.9	0.138102	1675.8
0.03861	1824.6	0.061304	1794.1	0.084711	1783.5	0.111476	1748.1	0.138548	1669.8
0.039161	1822.3	0.06175	1791.3	0.08521	1783	0.112342	1746.2	0.139072	1662.8
0.039686	1818.9	0.062406	1787.6	0.085551	1783.5	0.112972	1743.5	0.139386	1658.2
0.040185	1817.9	0.063063	1786	0.086287	1784.2	0.11397	1741.7	0.139911	1652.8
0.040632	1817.5	0.06343	1786.2	0.086839	1783.2	0.114364	1741.4	0.140409	1648.5
0.041236	1818.1	0.063851	1787	0.087285	1783.2	0.115126	1741.2		
0.041682	1816.8	0.064403	1786.4	0.087889	1782.1	0.115573	1740.4		
0.042208	1816.3	0.065007	1786.9	0.088441	1781.1	0.116387	1739.7		

Table B-4 Flow stress versus plastic strain of RHA for initial temperature 298 K and plastic strain rate 0.001/s

Strain	Stress (MPa)	Strain	Stress (MPa)	Strain	Stress (MPa)	Strain	Stress (MPa)	Strain	Stress (MPa)
0.000028	1064.6	0.031693	1298.4	0.068066	1346	0.103861	1362.6	0.1416	1374.4
0.000291	1081	0.032377	1299.3	0.069038	1346	0.104307	1362.6	0.142573	1375.7
0.000527	1094.4	0.033612	1302.3	0.069774	1346.9	0.105017	1362.2	0.144018	1375.7
0.000816	1107.3	0.034321	1304.1	0.070379	1346.5	0.105621	1361.7	0.144649	1375.3
0.001105	1120.7	0.035084	1305.8	0.071062	1346.9	0.1062	1361.7	0.145542	1375.3
0.001525	1135.3	0.03603	1307.1	0.071955	1347.3	0.106936	1362.2	0.14641	1374.4
0.001946	1140.9	0.036713	1309.3	0.072639	1346.9	0.107592	1362.2	0.147172	1376.6
0.00276	1152.2	0.037317	1310.1	0.073401	1348.2	0.108696	1362.6	0.148039	1376.6
0.003338	1159.5	0.038027	1311.8	0.073953	1348.2	0.109406	1363	0.148959	1376.6
0.00389	1166.4	0.03871	1312.3	0.074715	1347.4	0.110299	1363.9	0.149984	1378.3
0.004415	1173.3	0.039472	1313.1	0.075477	1348.7	0.110983	1365.6	0.150799	1379.2
0.005072	1180.2	0.040208	1314.4	0.076265	1350.4	0.111587	1365.2	0.151377	1378.7
0.005729	1185.8	0.041207	1317.9	0.077606	1350.4	0.112323	1364.8	0.153032	1377.4
0.00636	1192.3	0.041864	1317.5	0.078421	1350.4	0.113033	1364.8	0.154189	1376.2
0.007148	1197.4	0.042784	1319.2	0.079104	1350	0.113663	1365.2	0.154846	1377
0.007858	1203.1	0.04352	1321.4	0.079629	1350.8	0.114452	1365.6	0.155582	1377.5
0.008462	1206.5	0.044203	1320.9	0.080418	1351.3	0.115661	1364.4	0.156554	1377.5
0.009014	1210.8	0.044755	1320.9	0.081233	1351.7	0.116791	1365.7	0.157264	1378.3
0.009592	1215.6	0.045543	1323.1	0.081679	1352.6	0.117658	1365.2	0.158236	1379.2
0.010197	1218.2	0.046594	1325.7	0.082415	1352.6	0.118447	1365.2	0.159103	1378.8
0.010749	1220.7	0.047173	1326.1	0.082993	1353.4	0.119288	1365.7	0.159734	1379.6
0.011353	1225.9	0.048014	1326.5	0.083598	1353	0.119997	1366.5	0.160654	1378.8
0.011957	1228.9	0.048986	1328.3	0.084202	1354.3	0.120733	1368.3	0.161732	1378.3
0.012641	1232	0.049879	1329.1	0.085122	1354.7	0.121653	1368.7	0.162625	1378.4
0.013376	1236.3	0.050747	1330.9	0.086095	1354.7	0.122573	1367	0.163335	1378.4
0.014664	1241	0.05164	1332.2	0.086699	1355.6	0.124255	1367.4	0.164255	1380.1
0.015426	1246.6	0.052297	1331.7	0.08754	1356	0.124938	1366.5	0.165201	1381.4
0.016346	1249.7	0.052954	1332.2	0.088145	1356.5	0.125937	1365.3	0.165936	1381.8
0.016977	1252.3	0.053717	1331.7	0.088959	1356.9	0.126489	1365.7	0.166593	1381.8
0.017581	1254.8	0.054531	1332.6	0.089721	1358.6	0.127119	1366.1	0.167303	1380.5
0.018317	1258.3	0.055293	1333	0.090352	1359.1	0.127855	1366.1	0.169169	1381
0.019158	1260.9	0.056187	1334.8	0.091167	1358.2	0.12867	1367.9	0.169958	1381
0.019841	1263.9	0.056713	1334.3	0.091666	1359.1	0.129038	1367.9	0.170825	1381.4
0.02063	1265.6	0.057527	1335.6	0.092849	1358.6	0.129406	1368.3	0.171587	1381
0.021365	1269.1	0.058684	1337.4	0.09348	1357.4	0.130352	1369.2	0.172375	1381.4
0.022338	1271.2	0.05963	1339.1	0.094531	1358.7	0.131035	1369.6	0.173348	1382.3
0.023179	1274.3	0.060392	1338.7	0.095319	1356.9	0.132165	1370.5	0.17453	1382.3
0.023941	1277.3	0.061233	1339.1	0.096187	1359.5	0.133138	1370.9	0.175582	1381.4
0.025018	1281.2	0.061916	1340	0.097317	1359.1	0.134793	1371.8	0.176607	1383.2
0.025938	1283.8	0.062442	1340	0.098552	1359.5	0.135687	1371.3	0.177421	1383.2
0.026726	1285.5	0.063178	1340.4	0.099656	1360	0.136922	1372.2	0.178262	1384
0.027567	1287.2	0.063913	1341.3	0.100628	1360	0.1375	1373.5	0.179182	1384.5
0.028697	1291.5	0.064754	1341.7	0.10097	1360	0.138446	1373.5	0.180023	1384.5
0.029696	1293.3	0.065517	1341.7	0.101732	1360.4	0.139314	1373.5		
0.030511	1295.4	0.066226	1342.6	0.102468	1360.8	0.140155	1374.4		
0.031089	1297.6	0.066857	1342.2	0.103177	1362.1	0.140786	1374.8		

Table B-5 Flow stress versus plastic strain of RHA for initial temperature 298 K and plastic strain rate 0.1/s

Strain	Stress (MPa)	Strain	Stress (MPa)	Strain	Stress (MPa)	Strain	Stress (MPa)	Strain	Stress (MPa)
0.000212	1068.9	0.028986	1310.9	0.067382	1363.7	0.104517	1381.6	0.14499	1398.1
0.000475	1087.9	0.029801	1310.9	0.06825	1364.1	0.105017	1382	0.145489	1397.7
0.000606	1099.5	0.030458	1312.7	0.069196	1365	0.106173	1383.7	0.146698	1397.7
0.000842	1116.4	0.031036	1315.3	0.070037	1366.3	0.106856	1383.7	0.147723	1398.1
0.000973	1133.2	0.031824	1316.6	0.07093	1368.5	0.107592	1383.7	0.148275	1398.6
0.001315	1149.1	0.032955	1319.2	0.071771	1368.5	0.108617	1384.2	0.148827	1399
0.001604	1159.9	0.033769	1320	0.072744	1369.3	0.109905	1384.6	0.149642	1400.3
0.001945	1171.6	0.034453	1320.9	0.073795	1370.2	0.111035	1385.5	0.150351	1398.6
0.002339	1178.9	0.035819	1324.3	0.074741	1370.2	0.111692	1384.2	0.151587	1399
0.002681	1185.4	0.036397	1324.3	0.07574	1371.1	0.112481	1385	0.152769	1398.1
0.003259	1193.1	0.036897	1326.1	0.076502	1372.4	0.113348	1385.9	0.153715	1398.1
0.0036	1195.7	0.038184	1330.4	0.077605	1373.7	0.114268	1386.8	0.154609	1399
0.004415	1206.5	0.039183	1333.4	0.078499	1373.2	0.115082	1387.2	0.155135	1399
0.005151	1211.2	0.039787	1333.8	0.079445	1373.7	0.116055	1388.1	0.155897	1399.5
0.005703	1216.8	0.040523	1335.1	0.080286	1373.3	0.116817	1387.6	0.156344	1400.3
0.006281	1221.2	0.0421	1336.9	0.08118	1374.1	0.117842	1388.5	0.157447	1399.9
0.006754	1224.6	0.043046	1337.7	0.082205	1375.8	0.118551	1387.2	0.158499	1401.2
0.007411	1230.2	0.043624	1339	0.083177	1374.6	0.119471	1387.2	0.158919	1400.8
0.00791	1234.1	0.044571	1340.3	0.084176	1374.6	0.120391	1386.4	0.159313	1400.3
0.008567	1238.4	0.04599	1341.2	0.085017	1375.4	0.121127	1385.5	0.160916	1400.3
0.009119	1243.2	0.046515	1340.8	0.0857	1376.3	0.121863	1387.2	0.16181	1401.2
0.009671	1246.2	0.047251	1342.5	0.086462	1376.3	0.122861	1386.8	0.162414	1401.6
0.010222	1250.1	0.047856	1342.5	0.087145	1376.7	0.124018	1386.4	0.163623	1402.9
0.011431	1254.8	0.048539	1344.7	0.087934	1376.7	0.124649	1388.5	0.164517	1402.5
0.012299	1257.8	0.049722	1345.5	0.088775	1377.2	0.125411	1388.5	0.165595	1402.5
0.012982	1261.3	0.050352	1346.8	0.089642	1376.7	0.126698	1390.7	0.166383	1402.9
0.013691	1263.4	0.051272	1349.4	0.090142	1377.2	0.127671	1389.8	0.167355	1403.4
0.014322	1265.6	0.051903	1349.4	0.091061	1378.5	0.128459	1390.7	0.168223	1404.2
0.01511	1266.9	0.052639	1351.1	0.091902	1379.3	0.129484	1391.6	0.169195	1403.8
0.01603	1270.8	0.053427	1350.7	0.092796	1378.5	0.130772	1391.6	0.169931	1403.8
0.016766	1273.8	0.054216	1352.9	0.093689	1378.9	0.131193	1391.6	0.17093	1403.8
0.017528	1277.7	0.055162	1352.4	0.094636	1378.5	0.132112	1392.4	0.172007	1404.3
0.018763	1282	0.056029	1355.5	0.095214	1378.9	0.132848	1392.9	0.172795	1405.1
0.019815	1285	0.056686	1356.8	0.095897	1378.5	0.134451	1393.8	0.173558	1404.3
0.020393	1286.8	0.057947	1357.6	0.096659	1378.9	0.135292	1393.3	0.174372	1404.7
0.021391	1287.6	0.058736	1359.4	0.097264	1380.2	0.136659	1395.1	0.175318	1405.1
0.022311	1291.9	0.059656	1359.4	0.098026	1380.2	0.13771	1394.2	0.175897	1405.6
0.023231	1296.3	0.060549	1360.2	0.099209	1382	0.13863	1394.6	0.17679	1406.4
0.023967	1298.8	0.061679	1359.8	0.100102	1379.4	0.13955	1395.1	0.177657	1406
0.024703	1298.9	0.062573	1360.7	0.100864	1380.2	0.140259	1395.5	0.178446	1406.4
0.025859	1302.3	0.063519	1359.8	0.101889	1381.1	0.141153	1395.9	0.179155	1406.9
0.026595	1304.5	0.064307	1362.8	0.102599	1382	0.142204	1397.2	0.179786	1406.9
0.027672	1306.2	0.065332	1363.7	0.103282	1381.1	0.142809	1396.8	0.180233	1408.2
0.028277	1310.1	0.066278	1362.8	0.103887	1380.7	0.144149	1397.7		

Table B-6 Flow stress versus plastic strain of RHA for initial temperature 298 K and plastic strain rate 3500/s

Strain	Stress (MPa)	Strain	Stress (MPa)	Strain	Stress (MPa)	Strain	Stress (MPa)	Strain	Stress (MPa)
0.028009	1290.9	0.057762	1379.4	0.093368	1429.5	0.123083	1441.7	0.158363	1430.6
0.02843	1295	0.058157	1380.5	0.09371	1429.8	0.123398	1442.7	0.158626	1433.9
0.028772	1298.3	0.05863	1381.4	0.094235	1431	0.123793	1444.4	0.158916	1437.9
0.029141	1302.1	0.059024	1382.1	0.094787	1432.6	0.124187	1445.8	0.1591	1439.5
0.02943	1305.9	0.059418	1383.6	0.095339	1434.1	0.125448	1444.6	0.159521	1443.7
0.029694	1309.3	0.059944	1386.1	0.096023	1436.3	0.125868	1445.9	0.159916	1446.8
0.030088	1310.4	0.060759	1389	0.096469	1436.4	0.127365	1442.9	0.160232	1448.5
0.030535	1311.5	0.061573	1389.1	0.097389	1438.6	0.127733	1441.6	0.160941	1451.2
0.031112	1309.5	0.062204	1389.4	0.098335	1438.7	0.128337	1439.2	0.161441	1452.3
0.03148	1307.7	0.063386	1390.1	0.098676	1437.8	0.129098	1436.2	0.161887	1451.2
0.0319	1307.2	0.063885	1389.8	0.099307	1437.3	0.129702	1435.3	0.162517	1448.1
0.03232	1306.6	0.064279	1390.5	0.099753	1435.1	0.130227	1433	0.163147	1444.2
0.032662	1306.1	0.064936	1390.4	0.100278	1432.7	0.130962	1430.8	0.163567	1440.7
0.033318	1306.9	0.06554	1390.7	0.100803	1429.2	0.131382	1429	0.164039	1436.3
0.033765	1306.7	0.066355	1390.8	0.101721	1420.8	0.131986	1427.4	0.164327	1432.5
0.034264	1308.2	0.067143	1391.3	0.10193	1418.6	0.132459	1426.3	0.164825	1428.9
0.034659	1308.9	0.067695	1390.9	0.10235	1414.8	0.133693	1421.5	0.165324	1425.6
0.035079	1310	0.068457	1391.4	0.102874	1409.4	0.134796	1422.5	0.165822	1422.3
0.035447	1311.2	0.069087	1391.7	0.10332	1404	0.135716	1425.3	0.166295	1421.9
0.035868	1312.3	0.069639	1392.5	0.103845	1399.8	0.136505	1428.3	0.16682	1420.8
0.036577	1313.8	0.070375	1393.9	0.104396	1398.6	0.136926	1430.7	0.167346	1419.7
0.037181	1313.4	0.0709	1395.2	0.105027	1401.2	0.137294	1434	0.168055	1419.4
0.037864	1313.7	0.071347	1396.1	0.1055	1402.8	0.137557	1436.2	0.16858	1420.1
0.038469	1314.6	0.072083	1397.5	0.105764	1406.7	0.138004	1439.3	0.168948	1420.3
0.039126	1315.8	0.07274	1398.6	0.106264	1413	0.138504	1443.1	0.1695	1420.9
0.039625	1315.9	0.07337	1398.9	0.10658	1416.9	0.138689	1444.7	0.17021	1422.1
0.04015	1316.4	0.074054	1400.3	0.106896	1421.8	0.138925	1447	0.170709	1423.2
0.040728	1316.9	0.074527	1400.4	0.107476	1430.7	0.139372	1449.8	0.171549	1422
0.041096	1318.5	0.075105	1402	0.107634	1432.8	0.139767	1451.4	0.171917	1422.5
0.0422	1321.6	0.075736	1403.9	0.107818	1435.7	0.140293	1454	0.172364	1421.9
0.04249	1323.7	0.076156	1404.1	0.108003	1438.9	0.14116	1457.2	0.172915	1418.9
0.042937	1325.5	0.076603	1405.4	0.108292	1442.4	0.142184	1453.8	0.173308	1416.1
0.043331	1328.3	0.077102	1407.1	0.108845	1447.4	0.142815	1452.9	0.173728	1413
0.043778	1331.4	0.077628	1408	0.109108	1450.9	0.143287	1449.2	0.174306	1410.7
0.044146	1333.4	0.078337	1408.9	0.109714	1456.6	0.14368	1447.5	0.174726	1409.7
0.044646	1335.2	0.078811	1410.7	0.110029	1459.6	0.1441	1443.5	0.175172	1409
0.045145	1337.8	0.079047	1411.9	0.110371	1460.6	0.144651	1439.7	0.175698	1411.2
0.045593	1341.4	0.079416	1415	0.111028	1462.7	0.145229	1437.7	0.175909	1412.3
0.046223	1343.8	0.079705	1417.4	0.111527	1460.8	0.145649	1435.9	0.176356	1415.3
0.046539	1346.3	0.080152	1420.7	0.11221	1459.4	0.146174	1433.7	0.17654	1418
0.046934	1349	0.080547	1422.9	0.112656	1455.6	0.146646	1431.8	0.176777	1420.5
0.047407	1351.2	0.080994	1424.7	0.113128	1450.7	0.147093	1431.6	0.177041	1427
0.047985	1354.2	0.081493	1427.5	0.113521	1446.2	0.148353	1429	0.177226	1431.8
0.048275	1357.1	0.081861	1429	0.114019	1439.3	0.149246	1426.2	0.177411	1435.9
0.048722	1359.7	0.082361	1430.7	0.114465	1435.2	0.150244	1425.1	0.177753	1441.6
0.049011	1361.8	0.082939	1432.2	0.114937	1430.6	0.150822	1423.5	0.178043	1445.5
0.04951	1362.3	0.083938	1433.9	0.115435	1425.6	0.151321	1421.6	0.178307	1449.5
0.050088	1362.8	0.084647	1433.3	0.115881	1421.5	0.152029	1419.9	0.178518	1452.8
0.05043	1365.2	0.085303	1432.3	0.116327	1417.4	0.152135	1420.1	0.178597	1454.8
0.050877	1366.2	0.085934	1432.6	0.117298	1412.5	0.153263	1414	0.178887	1459.1
0.051403	1369.6	0.086669	1430.7	0.118348	1412.1	0.153867	1411.6	0.179308	1465.7
0.052191	1369.7	0.087352	1430.5	0.118769	1413	0.154392	1410.5	0.179624	1468.5
0.052822	1370.4	0.087851	1430.5	0.119426	1415.6	0.154838	1408	0.180255	1470.4
0.053374	1372.1	0.088508	1429.2	0.119768	1418.1	0.155574	1408.8	0.180676	1472
0.053847	1372.7	0.089007	1428.8	0.120031	1419.7	0.155784	1409.6	0.18107	1472.1
0.054477	1372.6	0.090031	1428.1	0.120558	1424.7	0.15631	1411.4	0.1817	1469.7
0.054977	1373.5	0.090793	1427.8	0.121057	1427.4	0.156494	1412.6	0.182277	1466.3
0.055423	1374.5	0.09145	1427.6	0.121426	1430.3	0.157125	1415.8	0.182775	1460.5
0.055975	1375.8	0.091739	1428.4	0.121899	1433.4	0.157467	1418.6	0.183116	1456.8
0.056737	1376	0.092265	1428.8	0.122346	1436.6	0.157783	1421.8	0.183351	1448.8
0.057263	1377.3	0.092921	1428.6	0.122846	1440.5	0.157941	1423.8	0.183586	1443.4

Table B-7 Flow stress versus plastic strain of RHA for initial temperature 298 K and plastic strain rate 7000/s

Strain	Stress (MPa)	Strain	Stress (MPa)	Strain	Stress (MPa)	Strain	Stress (MPa)	Strain	Stress (MPa)
0.03648	1359	0.058189	1417.2	0.08481	1462.9	0.116208	1484.7	0.154932	1485
0.036769	1360.7	0.058662	1417.8	0.08531	1465.7	0.116418	1485.5	0.155746	1483
0.037032	1364.1	0.058925	1416.6	0.085862	1468.5	0.117022	1485.7	0.156297	1479.6
0.037427	1366.5	0.059424	1416.2	0.086177	1470.4	0.117601	1486	0.157164	1477.6
0.037821	1369.3	0.059975	1415.4	0.086546	1472.6	0.118179	1487.8	0.157689	1477.5
0.038216	1372.5	0.060474	1414.7	0.087072	1476.6	0.119362	1491.8	0.158293	1477.7
0.03869	1377.2	0.061079	1414.2	0.087466	1479.1	0.119913	1491.3	0.159003	1477.4
0.039111	1380.4	0.061499	1413.2	0.087861	1480.9	0.120334	1491.8	0.159397	1477.1
0.039532	1383.9	0.061945	1412.6	0.088124	1482.9	0.120991	1492.4	0.159764	1477.7
0.039874	1387.2	0.062313	1412.4	0.088623	1484.9	0.121516	1493.1	0.16029	1479.1
0.040268	1390.4	0.063154	1411.6	0.089359	1488.5	0.121989	1493.5	0.160869	1481.5
0.040715	1393.9	0.06381	1412.7	0.0902	1490.4	0.122646	1494.4	0.161184	1482.2
0.041136	1396.6	0.064284	1414	0.090779	1491.9	0.123356	1494.5	0.16171	1483.9
0.0414	1400.4	0.065046	1415.8	0.091567	1492.7	0.124643	1494.4	0.162183	1485.4
0.041715	1404.1	0.06544	1417.2	0.091882	1491.8	0.125588	1492.6	0.162788	1487.8
0.042005	1407.4	0.065808	1419.5	0.092565	1491.5	0.126376	1492.9	0.163602	1489.2
0.042347	1411.1	0.06615	1420.7	0.093011	1489.3	0.126954	1492.4	0.164181	1490.5
0.042742	1413	0.066518	1421.7	0.093458	1487.8	0.127453	1491.3	0.164943	1491.2
0.043084	1415.4	0.066807	1422.5	0.093983	1485.8	0.128267	1490.4	0.166178	1493.7
0.043636	1420.4	0.067307	1425	0.094586	1483.7	0.128767	1490.8	0.166414	1492.2
0.044031	1424	0.068043	1428	0.095059	1482.6	0.129213	1489.7	0.167228	1491.3
0.044768	1432	0.068437	1429.9	0.095374	1480.2	0.129844	1489.9	0.16778	1489.8
0.045215	1434.6	0.0687	1431.1	0.095794	1477.5	0.13079	1490.7	0.168699	1490.2
0.045557	1437.9	0.069147	1432.9	0.096476	1474.9	0.13121	1491.6	0.169198	1488.3
0.045899	1440.3	0.069462	1434	0.097054	1472.1	0.13234	1493.3	0.169933	1486.1
0.046267	1441.4	0.069857	1435.9	0.097868	1468.5	0.133812	1494.8	0.170458	1484.2
0.046766	1443.9	0.070199	1437	0.098419	1466.9	0.1346	1494.7	0.171246	1480.4
0.047134	1444.2	0.070593	1438	0.098682	1466.1	0.135204	1493.4	0.171824	1479.6
0.047554	1443.2	0.071066	1439.3	0.09918	1463.4	0.135651	1494.3	0.172296	1477
0.048105	1439.9	0.071749	1440.7	0.099758	1462.5	0.136202	1493.8	0.172847	1474.4
0.048499	1437.2	0.072143	1440.9	0.100467	1462.2	0.136727	1493.4	0.173451	1471.3
0.048761	1432.7	0.072853	1441.9	0.100887	1462	0.137253	1493	0.174028	1468.6
0.049207	1428.3	0.073457	1441.4	0.101387	1460.9	0.137831	1493.6	0.174632	1466.9
0.049548	1424.5	0.074271	1440.2	0.101859	1461.3	0.139013	1493.6	0.175183	1464.7
0.049888	1420.3	0.074718	1439.5	0.102622	1462.1	0.139407	1493.8	0.175866	1464.4
0.050203	1416.6	0.075742	1438	0.103383	1461.8	0.140432	1493.4	0.176864	1462.1
0.050544	1412.4	0.076267	1437.2	0.104119	1463.8	0.141193	1492.2	0.177521	1461.9
0.050832	1408.2	0.076687	1436.6	0.104513	1464	0.141771	1490.6	0.177915	1460.9
0.051173	1403.6	0.077265	1436.2	0.105722	1465.7	0.142349	1490.4	0.178466	1459.4
0.051671	1398.8	0.078027	1435.8	0.10609	1465.9	0.142848	1491.5	0.178887	1459.1
0.052091	1397.4	0.078579	1435.4	0.106537	1466	0.143505	1490.5	0.179464	1458.6
0.052537	1395.9	0.078841	1435.4	0.107299	1466.4	0.144372	1489.6	0.180147	1457.2
0.053089	1394.3	0.079314	1435.1	0.107588	1466.7	0.145423	1490.6	0.18104	1453.4
0.053457	1394.6	0.079761	1436.5	0.108113	1467.5	0.146158	1490.6	0.181512	1450.8
0.053904	1396.4	0.080024	1437.3	0.108691	1468.6	0.146605	1490.3	0.182115	1446.3
0.054193	1398.9	0.080313	1437.7	0.1099	1471	0.147735	1491.5	0.182798	1442.8
0.054798	1401.3	0.080707	1438.7	0.1104	1473	0.148444	1492.3	0.183349	1439.5
0.055166	1404	0.080996	1440.3	0.111451	1474.9	0.149101	1493.2	0.183899	1433.6
0.055482	1406	0.081549	1444.6	0.112371	1475.8	0.149758	1492.2	0.184476	1428.8
0.055823	1408.4	0.082154	1446.5	0.11287	1477	0.150415	1492.3	0.184975	1424.8
0.056244	1410.2	0.082863	1449.9	0.113527	1477.5	0.151439	1492.6	0.185578	1420.3
0.056665	1411.7	0.083153	1451.8	0.114	1479.1	0.151912	1491.5		
0.057059	1413.6	0.083495	1453.7	0.114526	1480.7	0.152779	1489.4		
0.057453	1415	0.083968	1458.2	0.11513	1482.1	0.153619	1487.4		
0.057795	1416.6	0.084494	1460.6	0.115525	1483	0.15417	1486.2		

Table B-8 Flow stress versus plastic strain of RHA for initial temperature 473 K and plastic strain rate 3000/s

Strain	Stress (MPa)	Strain	Stress (MPa)	Strain	Stress (MPa)	Strain	Stress (MPa)	Strain	Stress (MPa)
0.030984	1177.8	0.05687	1235.8	0.087142	1287	0.118271	1280.2	0.148874	1270.6
0.031406	1185.6	0.057501	1237.4	0.087667	1286.7	0.118928	1280.9	0.149452	1271.4
0.031722	1192	0.058105	1240.3	0.088193	1286.4	0.119769	1281.7	0.149873	1272.3
0.032065	1200.8	0.05871	1244.5	0.088718	1285.3	0.120347	1282.4	0.150503	1272.6
0.03246	1205.3	0.058974	1247	0.089243	1285	0.120846	1281.8	0.150871	1274
0.032829	1209	0.0595	1251.6	0.089742	1281.6	0.121424	1281.8	0.151581	1276
0.033407	1211.7	0.059895	1256	0.091055	1278.7	0.122028	1282.5	0.151897	1276.7
0.033986	1214.7	0.060421	1259	0.092158	1277.2	0.122501	1283.1	0.152265	1277.7
0.034328	1217.6	0.060841	1262.1	0.092579	1279	0.123132	1284.5	0.152843	1280.3
0.034774	1219.6	0.061367	1265.1	0.093472	1281.8	0.1235	1285.2	0.153737	1283.2
0.0353	1221.8	0.062392	1265.8	0.094051	1283.8	0.124315	1287.5	0.154736	1286
0.035852	1224.1	0.06276	1266.9	0.09476	1285.6	0.124788	1289.2	0.155708	1287.3
0.036509	1226.2	0.063102	1267.3	0.09497	1286.1	0.124867	1289.1	0.155997	1287.7
0.037088	1227.5	0.063889	1266.3	0.095707	1289.5	0.125366	1290.7	0.156444	1287.8
0.037744	1228.8	0.064625	1265.3	0.096259	1290.7	0.126206	1284.6	0.156785	1287.7
0.038165	1229.9	0.065097	1263.5	0.096653	1292.1	0.126888	1281.5	0.157573	1286.7
0.038717	1230.9	0.065754	1262.2	0.097047	1292.4	0.127492	1278.4	0.158203	1284.4
0.039479	1233.6	0.066279	1261.5	0.097546	1291	0.128043	1277.3	0.158833	1281.7
0.040031	1233.4	0.066647	1260.2	0.098203	1290.9	0.128831	1275.5	0.159463	1279.8
0.04053	1234.8	0.067251	1260.2	0.098597	1291.5	0.129356	1274.8	0.159988	1276.5
0.04103	1237.5	0.067671	1259.7	0.099306	1289.4	0.129645	1274.8	0.16046	1272.9
0.041529	1238.9	0.068302	1261.3	0.100172	1288.2	0.130355	1275.7	0.160854	1270.9
0.042081	1240.7	0.068722	1262.4	0.101196	1286.1	0.130749	1276	0.161457	1266.4
0.042686	1243.3	0.069143	1264.3	0.102405	1285.7	0.131143	1276.2	0.162034	1263.8
0.043264	1245	0.069643	1267.2	0.10293	1285.8	0.131458	1277.3	0.162612	1261.3
0.043764	1248.1	0.070563	1269.7	0.103298	1285.3	0.132247	1281.2	0.163085	1260.3
0.044316	1250.7	0.070799	1271	0.103902	1285.2	0.132589	1282.3	0.163767	1259.4
0.044999	1252.7	0.071641	1274.7	0.104375	1284.2	0.133062	1285.8	0.16424	1260.2
0.045525	1254.9	0.072114	1277.7	0.104821	1284	0.133404	1288	0.165134	1261.6
0.046103	1256.2	0.072482	1278.8	0.105373	1284.1	0.133851	1289.3	0.166081	1267.4
0.046734	1257.5	0.073113	1281.6	0.10582	1283.9	0.134456	1291.2	0.166712	1270.6
0.047076	1259.1	0.073586	1283.4	0.106266	1284.8	0.13527	1290.1	0.16758	1274.2
0.047523	1261.8	0.073928	1283.8	0.106897	1286.7	0.135953	1290.3	0.167921	1276
0.047943	1263.8	0.074821	1285.8	0.107187	1287.9	0.137004	1289.7	0.16829	1278.1
0.048364	1265.3	0.075767	1287	0.10766	1291.2	0.137686	1285.4	0.169131	1283.2
0.048837	1267.1	0.076686	1284.9	0.107923	1292.8	0.138421	1284	0.169446	1283.1
0.049284	1268.2	0.077737	1283.9	0.108528	1295.5	0.139209	1283	0.170103	1284.1
0.049809	1269.6	0.078341	1282.7	0.109867	1292.1	0.13976	1280.8	0.170629	1283.4
0.050283	1271.8	0.078787	1280.5	0.110261	1290.4	0.140469	1279.8	0.171548	1280.3
0.050598	1272.7	0.079575	1279.5	0.110812	1288.1	0.140942	1278.1	0.172545	1277.9
0.051071	1272.4	0.080284	1277.8	0.111337	1285.1	0.14152	1276.9	0.172992	1276.6
0.051622	1270.5	0.080993	1276.4	0.112072	1283	0.141888	1277.2	0.173516	1273
0.052147	1266.9	0.08165	1275.2	0.112518	1282	0.14257	1275.9	0.174199	1269.9
0.05275	1260.4	0.082044	1275.1	0.113227	1279.1	0.14328	1275.7	0.175354	1264.4
0.053301	1257.2	0.082858	1275.2	0.113936	1279.4	0.143726	1275.2	0.17601	1260.3
0.053878	1251.9	0.083515	1276.3	0.114436	1279.9	0.144304	1274	0.176901	1253.3
0.05435	1246.4	0.084172	1277.4	0.114882	1280.8	0.145013	1272.3	0.177478	1246.1
0.054795	1240.1	0.084671	1280	0.115539	1280.7	0.146116	1272.4	0.178002	1239.2
0.055215	1236.7	0.085592	1285.1	0.11588	1279.9	0.146878	1271.8	0.178657	1232.2
0.05574	1234.8	0.086433	1286.4	0.116695	1280.3	0.147535	1271.3	0.179181	1224
0.056344	1234.8	0.086853	1287	0.117457	1279.3	0.148139	1270.9		

Table B-9 Flow stress versus plastic strain of RHA for initial temperature 673 K and plastic strain rate 3000/s

Strain	Stress (MPa)	Strain	Stress (MPa)	Strain	Stress (MPa)	Strain	Stress (MPa)	Strain	Stress (MPa)
0.020894	1007.3	0.045945	1103.7	0.077839	1123.5	0.117924	1113.9	0.155884	1111.3
0.021262	1009.8	0.046575	1102.6	0.07868	1124.1	0.118791	1112.9	0.156435	1112.2
0.021499	1013	0.047126	1100.3	0.079547	1126.3	0.119579	1110.1	0.156961	1112.7
0.021736	1016.6	0.047625	1099.3	0.079757	1126	0.120261	1108.5	0.157512	1112.1
0.022052	1020.5	0.048124	1098.8	0.080755	1123.7	0.120944	1107	0.158064	1111.4
0.022395	1025.2	0.048571	1099.5	0.081596	1122.7	0.121496	1107.9	0.158616	1111.9
0.02271	1028.2	0.049176	1102.1	0.082541	1120.4	0.1221	1108.3	0.159535	1110.9
0.023078	1030.8	0.049754	1102.3	0.082961	1120.3	0.122915	1109.7	0.160506	1108.3
0.02342	1032.1	0.050463	1103.1	0.083355	1118.3	0.123572	1110.5	0.160717	1108.4
0.023788	1033	0.050963	1105.4	0.084012	1118.3	0.123888	1112.8	0.16119	1109.3
0.024234	1031.7	0.051752	1111.5	0.085824	1116.6	0.124282	1114.6	0.161716	1112.4
0.024812	1028.1	0.05212	1113.5	0.086849	1121	0.125097	1118.7	0.162137	1116.7
0.025258	1023.8	0.052619	1114.2	0.087427	1119.5	0.12557	1120.4	0.162953	1125
0.025808	1017.7	0.053145	1116.1	0.08811	1121.9	0.126517	1123.9	0.163295	1128.2
0.026333	1012.1	0.053618	1117.2	0.088925	1125.3	0.127016	1126	0.163716	1132.9
0.026857	1007.3	0.054038	1118.7	0.089293	1125.7	0.128014	1126.3	0.164163	1136.1
0.027461	1002.5	0.054853	1119.8	0.089635	1126.4	0.128881	1126.8	0.164873	1139.6
0.027933	1000.7	0.05551	1120.3	0.090134	1128.2	0.129564	1123	0.165713	1135.3
0.02838	1002.8	0.056166	1119.1	0.09087	1130.1	0.130115	1119.3	0.166369	1130.2
0.028748	1004.9	0.056797	1118	0.091475	1132.2	0.130771	1116.7	0.166815	1125.2
0.029116	1008.3	0.057506	1115.9	0.092132	1135.8	0.131453	1113.6	0.167313	1119.9
0.029485	1013.8	0.058136	1113.6	0.092894	1135.7	0.132136	1112.4	0.167574	1111.9
0.029933	1018.8	0.058713	1110.9	0.093997	1133.2	0.132582	1111.2	0.168229	1102.4
0.030117	1023.3	0.059107	1107.9	0.094968	1128.5	0.133108	1111.7	0.168648	1096.8
0.030328	1028.5	0.059657	1103.6	0.095466	1125.6	0.133712	1113.3	0.169146	1091.4
0.030592	1033.7	0.060104	1101.9	0.096149	1121.2	0.134212	1115.7	0.169828	1087.7
0.030829	1038.9	0.060944	1098.9	0.096778	1117	0.134554	1117.6	0.170484	1086.3
0.03104	1042	0.061363	1095.6	0.097618	1114.4	0.135027	1120.5	0.171089	1088.9
0.031356	1047.5	0.062046	1092.8	0.098301	1113.6	0.135816	1121.8	0.171563	1092.7
0.031724	1051.3	0.062781	1091.5	0.098958	1115.6	0.136525	1124.4	0.172168	1099.7
0.03204	1054.8	0.063333	1089.3	0.099589	1118.8	0.137314	1126.9	0.172617	1108.3
0.032382	1059	0.063779	1088.4	0.100326	1121.9	0.137865	1125.1	0.172775	1111.8
0.032724	1062	0.064173	1088.3	0.100826	1126.8	0.138574	1122.4	0.173355	1122
0.033093	1064.1	0.064856	1087.9	0.101326	1132.1	0.139545	1118.2	0.174119	1134.6
0.033644	1064.7	0.065408	1089.4	0.101852	1136.1	0.140227	1111.1	0.174383	1138.3
0.034275	1065.7	0.065671	1090.7	0.102561	1136.5	0.141171	1105.1	0.175146	1146.1
0.034853	1066.3	0.066302	1094	0.103192	1138.5	0.141828	1103.9	0.17533	1145.9
0.035405	1067.8	0.066538	1095	0.104085	1139	0.142458	1103.6	0.176144	1144.6
0.035904	1068.1	0.066801	1096.7	0.10482	1135.8	0.142879	1105	0.176878	1133.9
0.036482	1070.8	0.067327	1099.3	0.105608	1133.3	0.143668	1109	0.177244	1125.9
0.037139	1071.7	0.067774	1101.7	0.106106	1130.7	0.14422	1112.1	0.177558	1119.7
0.037691	1072.8	0.068142	1102.9	0.106788	1124.1	0.144957	1118.7	0.178108	1110.7
0.038033	1074.5	0.068484	1103.7	0.107418	1117.9	0.145352	1122.8	0.178553	1103.3
0.038533	1077.7	0.068825	1104.1	0.108205	1112.4	0.145799	1126	0.179156	1097.9
0.03898	1080.9	0.069903	1106.5	0.108861	1110.8	0.14643	1130.5	0.179733	1094
0.039401	1084.2	0.070848	1105.4	0.109491	1109.7	0.146956	1132.9	0.180206	1093.9
0.0399	1087	0.07203	1104.9	0.110148	1109.7	0.147612	1129.9	0.180758	1095.8
0.040216	1089.9	0.072556	1103.9	0.11091	1110.4	0.148584	1128.4	0.1811	1097.5
0.040742	1093.5	0.073107	1103.6	0.111436	1112.1	0.149266	1122.7	0.181495	1101.8
0.041163	1096.8	0.073449	1103.6	0.11183	1113.6	0.149764	1118.4	0.18189	1107.1
0.041504	1097.6	0.074079	1104.5	0.112698	1117.6	0.150367	1114	0.182232	1113.2
0.041846	1099.7	0.074657	1105.9	0.113276	1119.3	0.150945	1110.3	0.182522	1117.2
0.042293	1101.7	0.074999	1107.9	0.113959	1121.2	0.151863	1105.6	0.182917	1122.9
0.042661	1103	0.075394	1109.5	0.114406	1121.8	0.152414	1102.3	0.183154	1123.7
0.043161	1104.5	0.075815	1112.6	0.115036	1121.1	0.153124	1103	0.183521	1122.2
0.04366	1106	0.076682	1116.4	0.115798	1120.2	0.153912	1104.3	0.184125	1118.3
0.044212	1107	0.07684	1117	0.116244	1119.7	0.154228	1106.2	0.184623	1110.8
0.044816	1107.5	0.077313	1120.1	0.116822	1117.9	0.154911	1108	0.184937	1105.8
0.04542	1105.6	0.077497	1120.7	0.117531	1115.1	0.15541	1110.4		

Table B-10 Flow stress versus plastic strain of RHA for initial temperature 873 K and plastic strain rate 3500/s

Strain	Stress (MPa)	Strain	Stress (MPa)	Strain	Stress (MPa)	Strain	Stress (MPa)	Strain	Stress (MPa)
0.024813	728.8	0.048262	834.8	0.08079	880.2	0.116274	859.3	0.151119	925.7
0.025024	734.6	0.048892	833.8	0.081551	879.5	0.116852	861.1	0.151592	926.4
0.025314	737.7	0.049286	833.9	0.08205	876.3	0.117694	864.2	0.152643	928.7
0.025577	740.5	0.04989	833.8	0.082496	874.7	0.118299	869	0.153405	927.3
0.025761	744.6	0.050573	833.2	0.083179	870.9	0.118746	872.5	0.154376	926.3
0.026051	749	0.051151	831.4	0.084071	866.1	0.119325	876.2	0.155112	924.5
0.026341	753.8	0.051597	831	0.084938	865.7	0.120219	881.5	0.155978	922.6
0.026604	757.4	0.052595	830.2	0.085831	864.9	0.120587	885	0.156503	921.7
0.02692	760.5	0.053121	830.1	0.086566	866.2	0.121455	892.6	0.157291	920.5
0.02742	764.3	0.053594	830.1	0.087328	867.5	0.121876	895.3	0.157685	920.2
0.027709	768.3	0.05425	830.8	0.08788	868.2	0.122323	898.7	0.158499	917.9
0.027946	771.9	0.054828	830.7	0.088958	871.6	0.122876	904	0.159103	917.3
0.028262	775	0.055196	831.6	0.089693	873.3	0.123192	906.4	0.159734	917.5
0.028499	777.8	0.055853	832.3	0.09014	873.2	0.123902	913	0.160496	919.3
0.028893	780.8	0.056431	833.4	0.090692	874.7	0.124823	919	0.161021	918.8
0.029235	782.7	0.056852	834.7	0.09148	877.2	0.125664	919.7	0.161941	920.5
0.029577	782.8	0.057351	836.3	0.092085	879.8	0.126189	919.9	0.162545	920.3
0.030023	781.6	0.057693	836.8	0.0929	880.6	0.127003	916.9	0.163438	921.7
0.030443	779.1	0.058611	833.5	0.093425	882.1	0.127895	914.9	0.163911	922.3
0.030968	778.3	0.059294	831.7	0.094108	881.9	0.128762	912.5	0.164568	922.8
0.031651	775.6	0.059819	829.6	0.09466	882.9	0.129497	908.8	0.16533	923.2
0.032123	773.1	0.060213	828	0.095475	885.3	0.130442	906.7	0.16596	923.6
0.032543	771.1	0.061237	825.5	0.095895	885.7	0.131125	905.7	0.166512	923.1
0.033147	769.8	0.062234	823.4	0.096394	887.2	0.131597	905.3	0.167011	922.6
0.033541	769.5	0.062944	825.3	0.096946	888.3	0.132307	905	0.16772	922
0.033987	769.9	0.06368	827.9	0.097577	890.1	0.132727	906.1	0.168245	921.5
0.034461	773.7	0.063995	828.8	0.098312	888.2	0.133515	906.9	0.169033	920
0.034777	776.8	0.064548	833.2	0.098916	885.3	0.133909	906.1	0.169663	918.6
0.035119	780.7	0.064969	836.1	0.099598	882.3	0.134698	909.2	0.170189	918.5
0.035409	785.9	0.065416	840.2	0.100044	878.7	0.135434	910.4	0.170609	917.7
0.035751	792.3	0.06589	845.4	0.100621	875.5	0.135933	910.6	0.17116	917.9
0.036093	796.2	0.066442	850.2	0.101356	870.9	0.136432	912.4	0.172369	919.3
0.036383	799.7	0.066942	855.4	0.10196	867.2	0.137195	915.9	0.173341	918.4
0.036962	806.8	0.067152	856.5	0.102721	864.2	0.137799	916	0.173604	918.5
0.037225	809.9	0.067757	860	0.103482	861.5	0.138351	916.3	0.174418	918.8
0.03762	813.3	0.068335	860.6	0.104165	860.9	0.139402	920.5	0.175048	918.6
0.037909	816	0.06915	864.8	0.104875	864.2	0.139848	919	0.175652	917.6
0.038225	818.6	0.070359	868.9	0.1054	864.5	0.140321	918.9	0.17623	916
0.038488	821.3	0.071279	868.1	0.106005	868.6	0.141215	923.7	0.17686	915.8
0.038883	824.3	0.071725	867.3	0.106689	875.4	0.142108	920.2	0.177754	917.4
0.039303	826.4	0.072277	868	0.107058	877.8	0.142843	919.1	0.178253	918.1
0.03975	827.7	0.073144	868	0.107215	878	0.143394	917.8	0.179015	920.2
0.040276	828.5	0.073538	867.6	0.107847	882.9	0.144051	918	0.179646	922.2
0.040906	829.6	0.073984	868.4	0.108635	882.1	0.144813	918.7	0.180198	923.8
0.041642	830.2	0.074615	868.7	0.10937	880.7	0.145443	918.5	0.18075	925.8
0.04222	830.6	0.074851	869.3	0.109895	878.7	0.146153	918.2	0.181223	927.2
0.042798	831.3	0.075771	871.2	0.110787	876.3	0.146547	917.4	0.181985	929.3
0.04356	831.9	0.076244	872.4	0.11147	872.9	0.14744	917.7	0.182589	928
0.044269	832.9	0.076796	873.1	0.112336	867.9	0.147912	916.9	0.183352	930.1
0.045162	833	0.077532	874.8	0.11307	863.7	0.148649	919.5	0.184402	930.6
0.045793	833.7	0.078215	876.2	0.113701	862.8	0.149358	921.5		
0.046265	832.8	0.079082	877.8	0.114252	860	0.149726	923		
0.046659	833.7	0.079345	878.8	0.115066	858.1	0.149936	923.5		
0.047343	834.4	0.079949	880.2	0.11588	857.7	0.150567	924.8		

Appendix C. Stan Specification Files

These are the Stan specification files that have been used for Bayesian analyses of the Johnson-Cook[1] and the Zerilli-Armstrong model for body-centered cubic materials.[2] Comments in these files of the form `//!{...}` can be ignored, since they are meant to be read by tools that extract source code fragments.

C.1 Specification File `jc.stan`

```
//!{funcstart}
functions {
  vector jc(vector epsilon_p, real log_epsilon_p_dot, vector T_star,
            real A, real B, real n, real C, real m) {

    int length_epsilon_p = num_elements(epsilon_p);
    vector[length_epsilon_p] sigma;

    real edot_factor = (1.0 + C*log_epsilon_p_dot);

    // The exponentiation operator "^" doesn't vectorize, so I need a
    // "for" loop here.
    for (i in 1:length_epsilon_p) {
      sigma[i] = (A + B*(epsilon_p[i])^n)*edot_factor*
        (1.0 - (T_star[i])^m);
    }

    return sigma;
  }
}
//!{funcend}

//!{datastart}
data {
  int<lower=1> num_curves;
  int<lower=0> curve_sizes[num_curves];
  vector[num_curves] epsilon_p_dot;

  vector[sum(curve_sizes)] epsilon_p;
  vector[sum(curve_sizes)] sigma;
  vector[sum(curve_sizes)] T;

  real<lower=0.0> T_melt;
  real<lower=0.0> T_room;

  real<lower=0.0> epsilon_p_dot_0;

```

[1]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.

[2]Zerilli FJ, Armstrong RW. Dislocation-mechanics-based constitutive relations for material dynamics calculations. Journal of Applied Physics. 1987;61(5):1816-1825.

```
  real<lower=0.0> A_guess_mean; real<lower=0.0> A_guess_sd;
  real<lower=0.0> B_guess_mean; real<lower=0.0> B_guess_sd;
  real<lower=0.0> C_guess_mean; real<lower=0.0> C_guess_sd;
  real<lower=0.0> m_guess_mean; real<lower=0.0> m_guess_sd;

  real<lower=0.0> n_alpha; real<lower=0.0> n_beta;

  vector<lower=0.0>[2] sd_sigma_guess_mean;
  vector<lower=0.0>[2] sd_sigma_guess_sd;
}
//!{dataend}

//!{transdatastart}
transformed data {
  vector[num_curves] log_epsilon_p_dot = log(epsilon_p_dot/epsilon_p_dot_0);
  vector[sum(curve_sizes)] T_star = (T - T_room)/(T_melt - T_room);
}
//!{transdataend}

//!{paramstart}
parameters {
  real<lower=0.0> A;
  real<lower=0.0> B;
  real<lower=0.0, upper=1.0> n;
  real<lower=0.0> C;
  real<lower=0.0> m;

  real<lower=0.0> sd_sigma[2];
}
//!{paramend}

//!{modelstart}
model {
  A ~ normal(A_guess_mean, A_guess_sd)T[0.0,];
  B ~ normal(B_guess_mean, B_guess_sd)T[0.0,];
  n ~ beta(n_alpha, n_beta);
  C ~ normal(C_guess_mean, C_guess_sd)T[0.0,];
  m ~ normal(m_guess_mean, m_guess_sd)T[0.0,];

  for (i in 1:2) {
    sd_sigma[i] ~
      normal(sd_sigma_guess_mean[i],
             sd_sigma_guess_sd[i])T[0.0,];
  }

  {
    int start_ind = 1;
    for (curve_ind in 1:num_curves) {
      int end_ind = start_ind + curve_sizes[curve_ind] - 1;

      real curr_sd_sigma = (epsilon_p_dot[curve_ind] <= 1.0
                            ? sd_sigma[1]
                            : sd_sigma[2]);

      sigma[start_ind:end_ind] ~ normal(jc(epsilon_p[start_ind:end_ind],
                                           log_epsilon_p_dot[curve_ind],
                                           T_star[start_ind:end_ind],
                                           A, B, n, C, m),
                                        curr_sd_sigma);

      start_ind = end_ind + 1;
    }
  }
}
//!{modelend}
```

C.2 Specification File za_bcc.stan

```
functions {

  vector za_bcc(vector epsilon_p, real log_epsilon_p_dot, vector T,
                real C0, real C1, real C3, real C4, real C5, real n) {

    int length_epsilon_p = num_elements(epsilon_p);
    vector[length_epsilon_p] sigma;

    real C3_C4_fac = -C3 + C4*log_epsilon_p_dot;

    // The exponentiation operator "^" doesn't vectorize, so I need a
    // "for" loop here.
    for (i in 1:length_epsilon_p) {
      sigma[i] = C0 + C1*exp(C3_C4_fac*(T[i])) + C5*(epsilon_p[i])^n;
    }

    return sigma;
  }

}

data {
  int<lower=1> num_curves;
  int<lower=0> curve_sizes[num_curves];

  vector[num_curves] epsilon_p_dot;

  vector[sum(curve_sizes)] epsilon_p;
  vector[sum(curve_sizes)] sigma;
  vector[sum(curve_sizes)] T;

  real<lower=0.0> C0_guess_mean;
  real<lower=0.0> C0_guess_sd;

  real<lower=0.0> C1_guess_mean;
  real<lower=0.0> C1_guess_sd;

  real<lower=0.0> C3_guess_mean;
  real<lower=0.0> C3_guess_sd;

  real<lower=0.0> C4_guess_mean;
  real<lower=0.0> C4_guess_sd;

  real<lower=0.0> C5_guess_mean;
  real<lower=0.0> C5_guess_sd;

  real<lower=0.0> n_alpha;
  real<lower=0.0> n_beta;

  real<lower=0.0> sd_sigma_guess_mean[2];
  real<lower=0.0> sd_sigma_guess_sd[2];
}

transformed data {
  vector[num_curves] log_epsilon_p_dot = log(epsilon_p_dot);
}

parameters {
  real<lower=0.0> C0;
  real<lower=0.0> C1;
  real<lower=0.0> C3;
  real<lower=0.0> C4;
  real<lower=0.0> C5;
  real<lower=0.0, upper=1.0> n;

  real<lower=0.0> sd_sigma[2];
}

model {
  C0 ~ normal(C0_guess_mean, C0_guess_sd)T[0.0,];
  C1 ~ normal(C1_guess_mean, C1_guess_sd)T[0.0,];
  C3 ~ normal(C3_guess_mean, C3_guess_sd)T[0.0,];
  C4 ~ normal(C4_guess_mean, C4_guess_sd)T[0.0,];
  C5 ~ normal(C5_guess_mean, C5_guess_sd)T[0.0,];

  n ~ beta(n_alpha, n_beta);

  for (i in 1:2) {
    sd_sigma[i] ~
      normal(sd_sigma_guess_mean[i],
             sd_sigma_guess_sd[i])T[0.0,];
  }

  {
    int start_ind = 1;
    for (curve_ind in 1:num_curves) {
      int end_ind = start_ind + curve_sizes[curve_ind] - 1;

      real curr_sd_sigma = (epsilon_p_dot[curve_ind] <= 1.0
                            ? sd_sigma[1]
                            : sd_sigma[2]);

      sigma[start_ind:end_ind] ~ normal(za_bcc(epsilon_p[start_ind:end_ind],
                                               log_epsilon_p_dot[curve_ind],
                                                        T[start_ind:end_ind],
                                                        C0, C1, C3, C4, C5, n),
                                                  curr_sd_sigma);

      start_ind = end_ind + 1;
    }
  }
}
```

Appendix D. Python Modules for Bayesian Analysis

These are the contents of some of the Python module files that have been used for Bayesian analyses of strength models. Comments of the form `#!{...}` can be ignored, since they are meant to be read by tools that extract source code fragments. Documentation of the parameters and return values of functions follows the guidelines of the Numpydoc docstring guide.[1]

D.1 Module File `jc.py`

```python
def jc(epsilon_p, log_epsilon_p_dot, T_star,
       A, B, n, C, m):
    """Flow stress according to the Johnson-Cook model

    Parameters
    ----------

    epsilon_p
        Strain

    log_epsilon_p_dot
        Natural logarithm of the normalized strain rate (i.e. strain
        rate divided by the reference strain rate)

    T_star
        Normalized temperature, usually (T - T_room)/(T_melt - T_room),
        where T_melt and T_room are the melting and room temperatures

    A, B, n, C, m
        The Johnson-Cook parameters
    """

    return ((A + B*(epsilon_p**n))*
            (1.0 + C*log_epsilon_p_dot)*(1 - T_star**m))
```

D.2 Module File `jc_pymc3.py`

```python
from jc import jc
import numpy as np
import pymc3 as pm

def make_jc_model(epsilon_p, sigma,
                  epsilon_p_dot, T,
                  T_melt, T_room, epsilon_p_dot_0,
                  prior_params):

    """Create a PyMC3 model conforming to the Zerilli-Armstrong (BCC) model

    Parameters
    ----------
```

[1]Numpydoc maintainers. Numpydoc docstring guide. c2017 [accessed 2018 May]. `https://numpydoc.readthedocs.io/en/latest/format.html`

```

    epsilon_p : list of 1-d NumPy array
        Strain values for all curves, where `epsilon_p[0]` contains
        strain values for the first curve, `epsilon_p[1]` contains
        strain values for the second curve, etc.

    sigma : list of 1-d NumPy array
        Stress values for all curves, where `sigma[0]` contains stress
        values for the first curve, `sigma[1]` contains stress values
        for the second curve, etc.

    epsilon_p_dot : 1-d array_like
        List or array where element `i` contains the strain rate for
        curve `i`

    T : list of 1-d array_like
        Temperature values for all curves, where `T[0]` contains
        temperature values for the first curve, `T[1]` contains
        temperature values for the second curve, etc.

    T_melt : float
        Melting temperature

    T_room : float
        Room temperature

    epsilon_p_dot_0 : float
        Reference strain rate, usually 1.0 per second.

    prior_params : dict
        Dictionary with the following keys: "A_guess_mean",
        "B_guess_mean", "C_guess_mean", "m_guess_mean",
        "sd_sigma_guess_mean", "A_guess_sd", "B_guess_sd",
        "C_guess_sd", "m_guess_sd", "sd_sigma_guess_sd", "n_alpha",
        and "n_beta". The values corresponding to
        "sd_sigma_guess_mean" and "sd_sigma_guess_sd" are lists or 1-d
        arrays with 2 elements, where both elements are positive
        numbers. Values corresponding to other keys are positive
        scalars.

    Returns
    -------

    A PyMC3 model
    """

    PosNormal = pm.Bound(pm.Normal, lower = 0.0)

    model = pm.Model()

    num_curves = len(epsilon_p)
    T_melt_minus_T_room = T_melt - T_room
    log_epsilon_p_dot = np.log(np.asarray(epsilon_p_dot)/epsilon_p_dot_0)

    with model:

        # Priors
        A = PosNormal("A",
                      mu = prior_params["A_guess_mean"],
                      sd = prior_params["A_guess_sd"])

        B = PosNormal("B",
                      mu = prior_params["B_guess_mean"],
                      sd = prior_params["B_guess_sd"])

        n = pm.Beta("n",
                    alpha = prior_params["n_alpha"],
                    beta = prior_params["n_beta"])

        C = PosNormal("C",
                      mu = prior_params["C_guess_mean"],
                      sd = prior_params["C_guess_sd"])

        m = PosNormal("m",
                      mu = prior_params["m_guess_mean"],
                      sd = prior_params["m_guess_sd"])

        sd_sigma = PosNormal("sd_sigma",
                             mu = np.asarray(prior_params["sd_sigma_guess_mean"]),
                             sd = np.asarray(prior_params["sd_sigma_guess_sd"]),
                             shape = 2)

        for i in range(num_curves):
            T_star = (T[i] - T_room)/T_melt_minus_T_room

            pm.Normal("sigma_curve{}".format(i),
                      mu = jc(epsilon_p[i],
                              log_epsilon_p_dot[i], T_star,
                              A, B, n, C, m),
                      sd = (sd_sigma[0]
                            if (epsilon_p_dot[i] <= 1.0)
                            else sd_sigma[1]),
                      observed = sigma[i])

    return model
```

D.3 Module File `za_bcc.py`

```
import numpy

def za_bcc(epsilon_p, log_epsilon_p_dot, T,
           C0, C1, C3, C4, C5, n, exp_func = numpy.exp):
    """Flow stress according to the Zerilli-Armstrong (BCC) model

    Parameters
    ----------

    epsilon_p
```

```
        Strain

    log_epsilon_p_dot
        Natural logarithm of the strain rate

    T
        Temperature

    C0, C1, C3, C4, C5, n
        The Zerilli-Armstrong (BCC) parameters. (Note that there is no
        C2 parameter, since that is for the Zerilli-Armstrong FCC
        model.)

    exp_func : function, optional
        Object representing the exponential function
    """

    return (C0 + C1*exp_func((-C3 + C4*log_epsilon_p_dot)*T) +
            C5*epsilon_p**n)
```

D.4 Module File `za_bcc_pymc3.py`

```
#!{importstart}
import numpy as np
import pymc3 as pm
from za_bcc import za_bcc
#!{importend}

def make_za_bcc_model(epsilon_p, sigma,
                      epsilon_p_dot, T,
                      prior_params):
    """Create a PyMC3 model conforming to the Zerilli-Armstrong (BCC) model

    Parameters
    ----------

    epsilon_p : list of 1-d NumPy array
        Strain values for all curves, where `epsilon_p[0]` contains
        strain values for the first curve, `epsilon_p[1]` contains
        strain values for the second curve, etc.

    sigma : list of 1-d NumPy array
        Stress values for all curves, where `sigma[0]` contains stress
        values for the first curve, `sigma[1]` contains stress values
        for the second curve, etc.

    epsilon_p_dot : 1-d array_like
        List or array where element `i` contains the strain rate for
        curve `i`

    T : list of 1-d array_like
        Temperature values for all curves, where `T[0]` contains
        temperature values for the first curve, `T[1]` contains
        temperature values for the second curve, etc.

    prior_params : dict
        Dictionary with the following keys: "C0_guess_mean",
        "C1_guess_mean", "C3_guess_mean", "C4_guess_mean",
        "C5_guess_mean", "sd_sigma_guess_mean", "C0_guess_sd",
        "C1_guess_sd", "C3_guess_sd", "C4_guess_sd", "C5_guess_sd",
        "sd_sigma_guess_sd", "n_alpha", and "n_beta". The values
        corresponding to "sd_sigma_guess_mean" and "sd_sigma_guess_sd"
        are lists or 1-d arrays with 2 elements, where both elements
        are positive numbers. Values corresponding to other keys are
        positive scalars.

    Returns
    -------

    A PyMC3 model
    """

    #!{boundnormstart}
    PosNormal = pm.Bound(pm.Normal, lower = 0.0)
    #!{boundnormend}

    #!{miscvarsstart}
    num_curves = len(epsilon_p)
    log_epsilon_p_dot = np.log(epsilon_p_dot)
    #!{miscvarsend}

    #!{withmodelstart}
    model = pm.Model()

    with model:
    #!{withmodelend}

        #!{priorsstart}
        C0 = PosNormal("C0",
                       mu = prior_params["C0_guess_mean"],
                       sd = prior_params["C0_guess_sd"])

        C1 = PosNormal("C1",
                       mu = prior_params["C1_guess_mean"],
                       sd = prior_params["C1_guess_sd"])

        C3 = PosNormal("C3",
                       mu = prior_params["C3_guess_mean"],
                       sd = prior_params["C3_guess_sd"])

        C4 = PosNormal("C4",
                       mu = prior_params["C4_guess_mean"],
                       sd = prior_params["C4_guess_sd"])

        C5 = PosNormal("C5",
                       mu = prior_params["C5_guess_mean"],
                       sd = prior_params["C5_guess_sd"])

        n = pm.Beta("n",
                    alpha = prior_params["n_alpha"],
                    beta = prior_params["n_beta"])

        sd_sigma = PosNormal("sd_sigma",
                             mu = np.asarray(prior_params["sd_sigma_guess_mean"]),
                             sd = np.asarray(prior_params["sd_sigma_guess_sd"]),
                             shape = 2)
        #!{priorsend}

        #!{likstart}
        for i in range(num_curves):
            pm.Normal("sigma_curve{}".format(i),
                      mu = za_bcc(epsilon_p[i],
                                  log_epsilon_p_dot[i], T[i],
                                  C0, C1, C3, C4, C5, n,
                                  exp_func = pm.math.exp),
                      sd = (sd_sigma[0]
                            if (epsilon_p_dot[i] <= 1.0)
                            else sd_sigma[1]),
                      observed = sigma[i])
        #!{likend}

    #!{retstart}
    return model
    #!{retend}
```

List of Symbols, Abbreviations, and Acronyms

β_{TQ}	Taylor-Quinney coefficient
$\boldsymbol{\theta}$	vector of Bayesian model parameters
$\boldsymbol{\theta}_{err}$	vector of nuisance parameters
$\boldsymbol{\theta}_{mdl}$	vector of parameters of the predictive part of a Bayesian model
$\dot{\epsilon}_p$	plastic strain rate
$\dot{\epsilon}_{p0}$	reference plastic strain rate, $1/\mathrm{s}$
ϵ_p	plastic strain
$\mathbf{D}$	experimental data or other known quantity on which parameter vector $\boldsymbol{\theta}$ is supposed to depend
$\mathbf{e}$	material state (e.g., a combination of the plastic strain, strain rate, and temperature at a point)
ρ	density
σ_{JC}	flow stress according to the Johnson-Cook model
σ_{mdl}	flow stress according to some predictive model
$\sigma_{ZA,BCC}$	flow stress according to the 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
$c(T)$	specific heat as function of temperature
C_i	fitting parameter of Zerilli-Armstrong (BCC) model, where $i \in \{0, 1, 3, 4, 5\}$

f_{area}	fraction such that $f_{area}\sigma_1^{i_c}\epsilon_{p,1}^{i_c}$, where $(\epsilon_{p,1}^{i_c}, \sigma_1^{i_c})$ is the first point of a stress-strain curve, equals the area under the missing part of a stress-strain curve over the interval $[0, \epsilon_{p,1}^{i_c}]$
i_c	index associated with a stress-strain curve
m	fitting parameter of Johnson-Cook model that represents thermal softening exponent
N	number of data points
n	fitting parameter of Johnson-Cook and Zerilli-Armstrong models that represents strain hardening exponent
n_c	number of stress-strain curves
n_α, n_β	parameters of the beta distribution used as a prior for fitting parameter n
N_{i_c}	number of data points for stress-strain curve i_c
$p(x\|d_1, d_2, \dots)$	PDF of a quantity x given quantities d_1, d_2, $\dots$
$p(x)$	PDF or prior PDF of a quantity x
SD_σ	standard deviation of the noise in a flow stress measurement
$SD_{\sigma,1}$	standard deviation of the noise in a flow stress measurement from a quasi-static experiment
$SD_{\sigma,2}$	standard deviation of the noise in a flow stress measurement from a high-strain-rate experiment
T	temperature
T^*	normalized temperature in Johnson-Cook model
T_{melt}	melting temperature
T_{room}	room temperature
1-D	one-dimensional
ARL	CCDC Army Research Laboratory

BCC	body-centered cubic
HDI	highest density interval
HMC	Hamiltonian Monte Carlo
IPM	interval predictor model
MCMC	Markov Chain Monte Carlo
MIDAS	Material Implementation, Database, and Analysis Source
NUTS	no U-turn sampler
PDF	probability density function
PFP	pushed forward posterior
PPD	posterior predictive distribution
RHA	rolled homogeneous armor

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
1 (PDF)	DIR CCDC ARL FCDD RLD CL TECH LIB
1 (PDF)	GOVT PRINTG OFC A MALHOTRA