Rapid Uncertainty Propagation for High-fidelity Prognostics Using SROMPy and Python

James E. Warner[1], Patrick E. Leser[2], and Jacob D. Hochhalter[3]

[1,2,3] *NASA Langley Research Center, Hampton, VA, 23681, USA*
james.e.warner@nasa.gov
patrick.e.leser@nasa.gov
jacob.d.hochhalter@nasa.gov

ABSTRACT

This work introduces a practical approach for accelerating probabilistic, high-fidelity prognostics using the stochastic reduced order model (SROM) method and its availability in the open-source Python package, `SROMPy`. SROMs are used as an efficient Monte Carlo simulation (MCS) method, providing low-dimensional representations of random model inputs that enable rapid and non-intrusive uncertainty propagation. This study represents the first application of the SROM approach in the field of prognostics and health management and serves as a tutorial demonstration of the `SROMPy` software package. The relative ease of applying SROMs with `SROMPy` for uncertainty propagation is demonstrated on an example of probabilistic, non-planar crack growth simulation. Results show that the SROM approach agrees well with results from MCS while providing the potential for orders of magnitude computational speedup. The complete source code and input data required to reproduce the results in this paper are available online to facilitate further evaluation and adoption of the SROM method by researchers in the field.

1. INTRODUCTION

The application of damage prognostics to real-world systems requires high-fidelity simulations (e.g., finite element (FE) analysis) to predict how damage will evolve in complex geometries. Furthermore, system uncertainties must be quantified and propagated through these models in order to provide reliable, probabilistic predictions. However, traditional Monte Carlo simulation (MCS) is impractical to use in these settings due to the number of model evaluations required for convergence, and the runtime required for each evaluation. Therefore, more advanced methods for propagating uncertainty are required to make probabilistic, high-fidelity damage prognostics computationally tractable.

There are two primary approaches for alleviating this computational burden. The first involves reducing the time for a single model evaluation by replacing an expensive computational model with an efficient surrogate model using machine learning (Warner et al., 2017; Leser et al., 2016; Sankararaman, Ling, Shantz, & Mahadevan, 2011). The second involves reducing the number of model evaluations required for convergence by using more sophisticated stochastic methods for uncertainty propagation. These two approaches may be combined, provided a satisfactory level of accuracy is retained.

This work aims to accelerate high-fidelity damage prognostics using the second approach of propagating uncertainty with fewer model evaluations. The most popular class of techniques for doing so are spectral methods, which express stochastic solutions as orthogonal polynomials of the random input parameters (i.e., polynomial chaos expansions). Two well-established methods following this approach are the stochastic collocation (Babuska, Nobile, & Tempone, 2007) and stochastic Galerkin method (Ghanem & Spanos, 2003). In their basic forms, stochastic collocation has the practical advantage of being a *non-intrusive* method (i.e., no modifications of the deterministic computational model are necessary), but the computational expense grows exponentially with the number of random inputs. Extensions of the approach have focussed on gaining efficiency by employing sparse grids (Nobile, Tempone, & Webster, 2008).

The stochastic reduced order model (SROM) approach (Grigoriu, 2009; Warner, Grigoriu, & Aquino, 2013; Grigoriu, 2011) was developed as an effective alternative to spectral methods, and has since been successfully applied to a range of stochastic problems (Sarkar, Warner, Aquino, & Grigoriu, 2014; Warner, Aquino, & Grigoriu, 2015; Emery, Field, Foulk, Karlson, & Grigoriu, 2015). This work, however, represents the first application of the approach in the field of prognostics and health management. A SROM provides a low-dimensional discrete approximation to random

model parameters by selecting optimal representative samples, and then using them to propagate uncertainty with relatively few model evaluations. The method is conceptually simple, non-intrusive, efficient, proven to converge to the exact solution (Grigoriu, 2011), and has been shown to offer advantages over the stochastic collocation and stochastic Galerkin approaches for some problems (Field, Grigoriu, & Emergy, 2015). Furthermore, the SROM approach has been implemented in the open-source Python package, `SROMPy`[1], providing user-friendly utilization of the method (Warner, 2018a).

The goal of this study is to demonstrate the practicality of the SROM approach for accelerating uncertainty propagation in damage prognostics and the relative ease of applying the method with `SROMPy` and Python. These goals are highlighted in the context of probabilistic prognosis for non-planar fatigue crack growth. Given uncertainty in initial damage state and crack growth rate parameters, the SROM approach was used to generate probabilistic end of life estimates in just a fraction of the time of MCS, while retaining a high degree of accuracy. It was shown that this analysis can be carried out using relatively few lines of Python code by leveraging the functionality of `SROMPy`. In order to facilitate reproducibility and encourage further evaluation of SROM-accelerated prognostics, the complete source code and input data for the example are available online[2].

The following section provides the relevant background for the study, starting with a formulation of the SROM approach for uncertainty propagation and then giving a brief overview of the `SROMPy` Python module that implements it. Then, an example of probabilistic, non-planar crack growth using the SROM method is presented, detailing the use of `SROMPy` to model random input parameters and then propagate uncertainty through the crack growth model. A comparison with MCS is provided here to demonstrate the accuracy and computational speedup provided by the SROM approach. Finally, the study is concluded in the summary section.

2. BACKGROUND

This section gives the theoretical and practical background regarding the use of `SROMPy` for uncertainty propagation in damage prognostics. Generally speaking, uncertainty propagation is the problem of characterizing statistics of a quantity of interest, $\mathbf{Y} \in \Gamma' \subset \mathbb{R}^{d'}$, that depends on random parameters, $\mathbf{X} \in \Gamma \subset \mathbb{R}^{d}$, through a deterministic model $\mathcal{M}$:

$$\mathbf{Y} = \mathcal{M}(\mathbf{X}). \tag{1}$$

Here, d and Γ are the dimension and range of the random input vector, respectively, and d' and Γ' are the dimension and range of the random output vector, respectively. In this work, $\mathbf{X}$ represents variables that describe crack growth rate and current damage state, $\mathcal{M}$ is a high-fidelity crack growth simulation, and $\mathbf{Y}$ is the end of life of the specimen being analyzed. The SROM theory and `SROMPy` capabilities are described in the context of the general stochastic problem in Eq. (1) before being applied specifically to probabilistic prognostics.

2.1. SROM Theory

SROMs can be viewed as a "smart" Monte Carlo method for stochastic problems. The approach efficiently discretizes the stochastic space and significantly reduces the computational complexity associated with propagating uncertainty relative to MCS, while retaining the benefits of a non-intrusive method. The approach is particularly beneficial for problems that depend on time-consuming computational models, e.g., the high-fidelity crack growth simulation used in this study.

To solve Eq. (1), the model inputs $\mathbf{X}$ are first approximated by a SROM $\tilde{\mathbf{X}}$, a simple random vector taking values $\tilde{\mathbf{x}}^{(1)}, ..., \tilde{\mathbf{x}}^{(m)}$ with probabilities $p^{(1)}, ..., p^{(m)}$. Then, the outputs $\mathbf{Y}$ are estimated using local piecewise-constant or piecewise-linear approximations of the model $\mathcal{M}$. These two steps are elaborated on in the following sections. As many details were omitted for the sake of brevity, consult the relevant references (Grigoriu, 2009; Warner et al., 2013; Emery et al., 2015) for further explanation.

2.1.1. Constructing a SROM for Model Inputs

When using the SROM approach, it is assumed that the statistics of the model inputs $\mathbf{X}$ are known a priori. Let the following expressions represent the marginal cumulative distribution functions (CDFs), moments of order q, and correlation matrix of $\mathbf{X}$

$$F_i(x_i) = P(X_i \leq x_i) \tag{2}$$

$$\mu_i(q) = E[X_i^q] \tag{3}$$

$$\mathbf{r} = E[\boldsymbol{X}\boldsymbol{X}^T], \tag{4}$$

where $E[\cdot]$ is the expectation operator.

The SROM approximation, $\tilde{\mathbf{X}}$, is a discrete random vector defined by

$$\tilde{\mathbf{X}} \equiv \{(\tilde{\mathbf{x}}^{(k)}, p^{(k)}),\ k = 1, ..., m\}, \tag{5}$$

where $\tilde{\mathbf{x}}^{(k)}$ denote samples of $\mathbf{X}$, and $p^{(k)} = P(\mathbf{X} = \tilde{\mathbf{x}}^{(k)})$ (Grigoriu, 2009). Note that $p^{(k)} \geq 0\ \forall k$, and $\sum_{k=1}^{m} p^{(k)} = 1$ so that $\tilde{\mathbf{X}}$ has a valid probabilistic description. It then follows that the statistics of $\tilde{\mathbf{X}}$ corresponding to those of $\mathbf{X}$ in

[1] Publicly available at https://github.com/nasa/SROMPy (Warner, 2018b)
[2] See https://github.com/nasa/SROMPy/examples/phm18

Equations (2) - (4) are given by

$$\tilde{F}_i(x_i) = \sum_{k=1}^{m} p^{(k)} \mathbf{1}\left(\tilde{x}_i^{(k)} \leq x_i\right) \tag{6}$$

$$\tilde{\mu}_i(q) = \sum_{k=1}^{m} p^{(k)} (\tilde{x}_i^{(k)})^q \tag{7}$$

$$\tilde{r}(i,j) = \sum_{k=1}^{m} p^{(k)} \tilde{x}_i^{(k)} \tilde{x}_j^{(k)}, \tag{8}$$

where $\mathbf{1}(\text{condition})$ is the indicator function, evaluating to 1 if the condition is true and 0 otherwise. In this context, the number of sample-probability pairs that define the SROM, m, is referred to as the SROM size.

The defining SROM samples and probabilities are chosen such that $\tilde{\mathbf{X}}$ is an optimal representation of $\mathbf{X}$ in a statistical sense. This is done through the solution of the following optimization problem:

$$\begin{aligned} \tilde{\mathbf{X}} \equiv \underset{\{\tilde{\mathbf{x}}\},\mathbf{p}}{\operatorname{argmin}} & \left(\sum_{i=1}^{3} \alpha_i e_i(\{\tilde{\mathbf{x}}\}, \mathbf{p})\right) \\ \text{s.t. } & \sum_{k=1}^{m} p^{(k)} = 1 \text{ and } p^{(k)} \geq 0,\ k = 1, ..., m, \end{aligned} \tag{9}$$

where e_1, e_2, and e_3 quantify the error between the SROM and target CDFs, moments, and correlation matrix, respectively. The weighting factors, α_i, are used to ensure that each error component have similar order of magnitude or to emphasize the SROM's ability to represent a particular statistic of $\mathbf{X}$. While the particular form of the error terms is problem dependent, one common example is

$$e_1(\{\tilde{\mathbf{x}}\}, \mathbf{p}) = \frac{1}{2} \sum_{i=1}^{d} \int_{\mathbb{R}} \left(\tilde{F}_i(x) - F_i(x)\right)^2 dx \tag{10}$$

$$e_2(\{\tilde{\mathbf{x}}\}, \mathbf{p}) = \frac{1}{2} \sum_{i=1}^{d} \sum_{q=1}^{\bar{q}} \left(\tilde{\mu}_i(q) - \mu_i(q)\right)^2 \tag{11}$$

$$e_3(\{\tilde{\mathbf{x}}\}, \mathbf{p}) = \frac{1}{2} \sum_{i,j=1}^{d} \left(\tilde{r}_{ij} - r_{ij}\right)^2, \tag{12}$$

where $\bar{q}$ is the largest moment of interest. Note that Eq. (12) allows the SROM approach to naturally handle correlated input variables.

An additional strength of the SROM approach is that it can still be applied if an explicit probability law for $\mathbf{X}$ is unavailable and there is only access to a collection of N independent, equally likely samples, $\{\hat{\mathbf{x}}^{(k)}\}_{k=1}^{N}$, of the vector. In this case, the following empirical estimators for the statistics of $\mathbf{X}$ can be used in the error terms in Equations (10)-(12):

$$\hat{F}_i(x_i) = \frac{1}{N} \sum_{k=1}^{N} \mathbf{1}\left(\hat{x}_i^{(k)} \leq x_i\right) \tag{13}$$

$$\hat{\mu}_i(q) = \frac{1}{N} \sum_{k=1}^{N} (\hat{x}_i^{(k)})^q \tag{14}$$

$$\hat{r}(i,j) = \frac{1}{N} \sum_{k=1}^{N} \hat{x}_i^{(k)} \hat{x}_j^{(k)} \tag{15}$$

2.1.2. Propagating Uncertainty to Outputs via SROMs

After a SROM is generated to represent the input parameters according to the previous section, it is then used to form a SROM-based surrogate model for uncertainty propagation. The surrogate is either a piecewise constant or piecewise linear response surface that maps samples of $\mathbf{X}$ directly to $\mathbf{Y}$ that can be used in place of the original model, $\mathcal{M}$, for MCS. The first step for either case is to evaluate Eq. (1) for each sample defining the input SROM, $\tilde{\mathbf{X}}$, i.e.,

$$\tilde{\mathbf{y}}^{(k)} = \mathcal{M}(\tilde{\mathbf{x}}^{(k)}), \text{ for } k = 1, ..., m, \tag{16}$$

to produce the collection of SROM output samples, $\{\tilde{\mathbf{y}}^{(k)}\}_{k=1}^{m}$.

For the simpler piecewise constant approximation, the surrogate model is then given by

$$\tilde{\mathbf{Y}}_C(\mathbf{X}) = \sum_{k=1}^{m} \mathbf{1}\left(\mathbf{X} \in \Gamma_k\right) \tilde{\mathbf{y}}^{(k)}, \tag{17}$$

where $\{\Gamma_k,\ k = 1, ..., m\}$ is a partition of the range Γ such that $P(\mathbf{X} \in \Gamma_k) = p^{(k)}$. Specifically, $\{\Gamma_k\}$ is a Voronoi tessellation of Γ with centers at the samples $\tilde{\mathbf{x}}^{(k)}$. In practice, however, the partition does not have to be constructed explicitly and Eq. (17) can be implemented in a straightforward manner. Here, a given sample of $\mathbf{X}$ is allocated to a particular cell Γ_k if it is closest to the input sample, $\tilde{\mathbf{x}}^{(k)}$. The surrogate model, $\tilde{\mathbf{Y}}_C$, then simply assumes the value of the corresponding SROM output sample, $\tilde{\mathbf{y}}^{(k)}$.

Equation (17) represents a closed-form expression that can be rapidly evaluated using MCS to produce a collection of output samples, $\{\hat{\mathbf{y}}^{(k)}\}_{k=1}^{N}$, forming an estimator for the true model output. Alternatively, $\tilde{\mathbf{Y}}_C(\mathbf{X})$ can be used to directly estimate the statistics of $\mathbf{Y}$ using SROM approximations analogous to Equations (6)-(8). For instance, the distributions and moments of the output can be approximated as

$$P(Y \leq y) \approx P(\tilde{Y}_C \leq y) = \sum_{k=1}^{m} p^{(k)} \mathbf{1}\left(\tilde{y}^{(k)} \leq y\right) \tag{18}$$

$$E[Y^q] \approx E[\tilde{Y}_C^q] = \sum_{k=1}^{m} p^{(k)} (\tilde{y}^{(k)})^q. \tag{19}$$

A more effective piecewise linear surrogate model, $\tilde{\mathbf{Y}}_L$, can be constructed by considering a first order Taylor series expansion over each cell in $\{\Gamma_k\}$ as follows

$$\tilde{\mathbf{Y}}_L(\mathbf{X}) = \sum_{k=1}^{m} \mathbf{1}\left(\mathbf{X} \in \Gamma_k\right) \left[\tilde{\mathbf{y}}^{(k)} + \nabla\tilde{\mathbf{y}}^{(k)} \cdot \left(\mathbf{X} - \tilde{\mathbf{x}}^{(k)}\right)\right], \tag{20}$$

where $\nabla\tilde{\mathbf{y}}^{(k)}$ denotes the gradient of the output with respect to the components of $\mathbf{X}$ evaluated at sample k. Equation (20) improves upon the accuracy of the simple piecewise constant approximation in Eq. (17), but with the added expense for computing gradients numerically using the finite difference method, requiring $m(d+1)$ model evaluations versus m via Eq. (16) alone. Note that Eq. (20) can be generalized to higher order approximations by including addition terms of the Taylor expansion (Field et al., 2015), but only piecewise constant and linear models are considered here.

2.1.3. SROM Method Summary

Given a probabilistic description of random input parameters to a computational model, the procedure for propagating uncertainty through the model with the SROM approach is summarized as follows:

1. Construct input SROM
   - Solve Eq. (9)
2. Execute computational model for each SROM sample
   - Evaluate Eq. (16)
3. Generate SROM output approximation
   - Piecewise constant - use Eq. (17).
   - Piecewise linear - use Eq. (20).
     - Calculate gradients using finite difference.

Once the closed-form SROM output approximation has been formed in step 3, it can be efficiently sampled using MCS to estimate the output statistics (e.g., distributions and moments). Alternatively, Eqs. (18) and (19) can be used directly for the case of the piecewise constant approximation.

Regarding the efficiency of the SROM approach, let $t_{\tilde{\mathbf{Y}}_C}$ and $t_{\tilde{\mathbf{Y}}_L}$ denote the total computation time required to generate the piecewise constant and piecewise linear SROM approximations, respectively. It follows that

$$t_{\tilde{\mathbf{Y}}_C} \approx t_{\tilde{X}} + mt_{\mathcal{M}} \tag{21}$$

$$t_{\tilde{\mathbf{Y}}_L} \approx t_{\tilde{X}} + m(d+1)t_{\mathcal{M}}, \tag{22}$$

where $t_{\tilde{X}}$ is the time required to form $\tilde{\mathbf{X}}$ in step (1) and $t_{\mathcal{M}}$ is the time for a single evaluation of the model $\mathcal{M}$. Here, the added computational expense for $\tilde{\mathbf{Y}}_L$ is due to the additional model evaluations required to estimate gradients.

The primary advantage of the SROM approach is that the solution times in Equations (21) and (22) can be substantially lower than traditional MCS by using far fewer model evaluations. To this end, let N represent the number of samples required for convergence with MCS, such that the total computational time is approximately $Nt_{\mathcal{M}}$. Assuming that $t_{\tilde{X}} << t_{\mathcal{M}}$, representing the common case where $\mathcal{M}$ is a computationally intensive high-fidelity simulation[3], then the computational speedup, S, for the piecewise constant and piecewise linear SROM approximation relative to MCS can be approximated as

$$S_{\tilde{\mathbf{Y}}_C} \approx \frac{N}{m} \tag{23}$$

$$S_{\tilde{\mathbf{Y}}_L} \approx \frac{N}{m(d+1)}, \tag{24}$$

respectively. Since in many typical cases $m << N$, the speedup values can be orders of magnitude in size.

2.2. `SROMPy` Overview

The `SROMPy` Python module (Warner, 2018b) is the first publicly-available software package that implements the SROM approach to uncertainty propagation. This section provides a brief overview of the software's fundamental capabilities[4], including a simple example of approximating a normal random variable with a SROM for illustration. A basic understanding of the Python programming language is assumed. The interested reader can consult the technical report (Warner, 2018a) that accompanied the release of the software or the user documentation that comes with the source code (Warner, 2018b) for more information.

2.2.1. Target Random Quantities

In order to solve an uncertainty propagation problem (Eq. (1)), a user must first choose a probabilistic description for the model inputs $\mathbf{X}$. `SROMPy` provides built-in options for representing random quantities for both the case when $\mathbf{X}$ follows known analytical probability distributions, and when only independent samples of it are available.

Currently, scalar random variables following beta, gamma, and normal probability distributions are directly supported in the software. Random vectors whose components follow standard distributions and have known correlations are supported in `SROMPy` through the implementation of translation random vectors, a method for modeling non-Gaussian random vectors (Grigoriu, 1995; Arwade, 2005). When only a collection of samples, $\{\hat{\mathbf{x}}^{(k)}\}_{k=1}^{N}$, is available to describe $\mathbf{X}$, `SROMPy` provides an implementation of a random vector with sample-based statistics (Eq.(13) - (15)). While only a small subset of all possible probability distributions are currently available in SROMPy, it is straightforward to extend the pack-

[3] The time required to solve Eq. (16) to generate $\tilde{\mathbf{X}}$ is typically on the order of seconds or minutes.

[4] Details provided in this paper correspond to version 1.0 of SROMPy. Modifications may occur with future versions.

age to model new random variables with SROMs (Warner, 2018a).

2.2.2. SROM Functionality

The goal of `SROMPy` is to allow users to easily model random quantities, $\mathbf{X}$, using SROMs, $\tilde{\mathbf{X}}$, and use them to efficiently propagate uncertainty through computational models, as described in Sections 2.1.1 and 2.1.2. These capabilities are provided through the classes[5] `SROM` and `SROMSurrogate`, respectively.

The `SROM` class is the fundamental component of the `SROMPy` package whose primary role is to select optimal SROM samples and probabilities through the solution of Eq. (9)[6]. The class also implements Equations (6) - (8) to calculate SROM statistics and provides access to the optimal samples so that a user can input them to their computational model as in Eq. (16).

Once a user has generated output samples via Eq. (16), the `SROMSurrogate` class is used to construct the SROM approximation to the model output. Specifically, this class provides an implementation of the piecewise constant and piecewise linear approximations in Equations (17) and (20), respectively, that can be sampled to estimate output statistics. There are also methods to evaluate the statistics directly for the piecewise constant case, e.g., Equations (18) and (19).

2.2.3. Postprocessing

SROMPy provides a few simple utilities for comparing statistics of a SROM versus a target quantity it is approximating with the `Postprocessor` class. For example, the class can be used to calculate and output errors in the SROM moment estimates, or to automatically produce CDF comparison plots.

2.2.4. Example - SROM for a Normal Random Variable

Below is a simple example that illustrates a small subset of `SROMPy` functionality from each of the three previous sections. In particular, a SROM $\tilde{X}$ with size $m = 10$ is generated to model a normal random variable, $X \sim N(\mu, \sigma) \in \mathbb{R}^1$, with mean, $\mu = 3.0$, and standard deviation $\sigma = 1.5$. The CDF of the optimized SROM is then compared to that of the target.

The Python source code that implements this example using `SROMPy` is shown below.

```
from target import NormalRandomVariable
from srom import SROM
from postprocess import Postprocessor

#Initialize Normal random variable
normal = NormalRandomVariable(mean=3, std_dev=1.5)

#Initialize SROM and optimize to match the target:
srom = SROM(size=10, dim=1)
srom.optimize(normal)

#Compare SROM and Normal CDFs (produces Figure 1):
pp = Postprocessor(srom, normal)
pp.compare_CDFs()
```

[5] Python data structures

[6] `SROMPy` uses a BFGS optimization algorithm implemented in the `scipy` Python module (Jones, Oliphant, Peterson, et al., 2001).

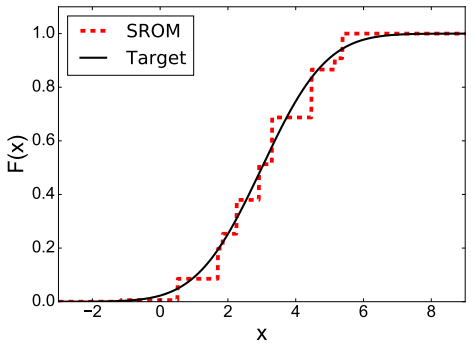


Figure 1. Comparison of a SROM and target (Normal) random variable CDF for $m = 10$.

Executing this code produces the CDF comparison plot shown in Figure (1), showing the piecewise constant SROM approximation versus the target normal CDF.

The above source code demonstrates the relative ease of deploying SROM capabilities with `SROMPy`, where an arbitrary random variable X is modeled with a SROM $\tilde{X}$ using just a few lines of code. The first block of code imports the relevant `SROMPy` classes needed for the example. Next, the `NormalRandomVariable` class is used to represent the target random variable in this example, $X \sim N(3.0, 1.5)$. The `SROM` class is then used to approximate the normal random variable with a SROM. Here, a call to the `optimize` method executes the solution of the optimization problem in Eq. (9) to determine the optimal SROM samples and probabilities. Finally, the `Postprocessor` class is used to generate the CDF comparison plot seen in Figure 1.

3. Example - Probabilistic Prognostics

3.1. Experimental Setup

The SROM approach is demonstrated here in the context of probabilistic prognosis for non-planar fatigue crack growth using the `SROMPy` software package. The experimental setup and crack growth model were borrowed from previous work done in (Leser et al., 2016, 2017), where the focus in the current study is on accelerating the uncertainty propagation process using SROMs. The experiment used an edge-notched, AA2024-T3 specimen with two holes drilled in it, as shown in Figure 2. The crack was grown from the notch under a

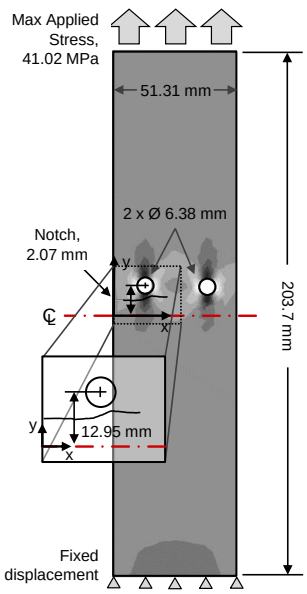


Figure 2. Diagram of the two-hole specimen.

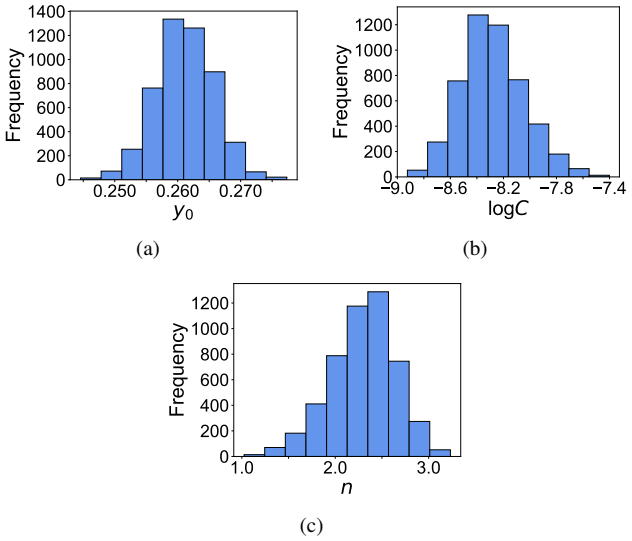


Figure 3. Histograms of the input parameter samples: (a) y_0, (b) log$C$, (c) n

constant amplitude of 41.0 MPa, with a load ratio $R = 0.1$, at 10 Hz.

A high-fidelity, finite element (FE)-based fracture modeling approach was developed to predict fatigue crack growth in the edge-notched specimen. The fracture mechanics code FRANC3D (Fracture Analysis Consultants Inc., 2011) was used to propagate geometrically-explicit cracks in a FE mesh via re-meshing while the parallel FE code ScIFEN (Warner, Bomarito, Heber, & Hochhalter, 2016) was used to evaluate the crack driving forces for a given crack configuration. The crack growth rate, $\frac{da}{dN}$, was calculated at each step according to

$$\frac{da}{dN} = C(\Delta K)^n, \tag{25}$$

where ΔK is the stress intensity factor (SIF), a is the crack size, N is the cycle number, and C and n are empirical parameters (Paris & Erdogan, 1963). The high-fidelity fracture simulation grows a crack according to Eq. (25) by iterating between remeshing and FE calculations until the stress intensity factor reaches a critical value, representing failure of the specimen. More details on the model can be found in (Leser et al., 2017).

This example will be used to demonstrate the SROM approach to propagate uncertainty through the high-fidelity crack growth model and generate a probabilistic prediction of failure in the edge-notched specimen. It is assumed that the uncertainty arises from the initial damage state (the y-coordinate of the notch, y_0) and the simplified crack growth model used (the empirical parameters, C and n). In terms of the general stochastic model in Eq. (1), the random input vector is $\mathbf{X} = [y_0, C, n] \in \mathbb{R}^3$ and the quantity of interest is the specimen's end of life (EOL), or the cycle at which failure occurs, i.e., $Y = [EOL] \in \mathbb{R}^1$. The model, $\mathcal{M}$, represents the iteration between fracture mechanics and FE calculations until the failure criteria is met.

This example demonstrates a common practical case where an explicit probability distribution to describe $\mathbf{X}$ was not available and was instead estimated using a Bayesian calibration and Markov chain Monte Carlo (MCMC) approach (Leser et al., 2017), resulting in a set of 5,000 independent samples $\{\hat{\mathbf{x}}^{(k)}\}_{k=1}^{N}$. Histograms for each input parameter's samples can be seen in Figure 3. A MCS prediction was made by running the crack growth simulation for each of the input parameter samples. The resulting collection of EOL output samples, $\{\hat{\mathbf{y}}^{(k)}\}_{k=1}^{N}$, and its distribution were treated as the reference solution. Several SROM approximations of varying fidelity were generated and compared to the MCS solution to assess accuracy and efficiency. These comparisons are first shown in the following section, followed by a demonstration of the `SROMPy` package for one particular case, detailing the necessary source code to solve the problem.

3.2. SROM-Accelerated EOL Predictions

Given the uncertainty in input parameters $\mathbf{X} = [y_0, C, n]$ (Figure 3), the CDF of $Y = [EOL]$ was estimated using both the piecewise constant SROM approximation $\tilde{Y}_C$ from Eq. (17) and the piecewise linear SROM approximation $\tilde{Y}_C$ from Eq. (20). Three different SROM sizes were tested, $m = 5, 10$, and 20, to observe the performance of the two different approximations for increasing model size. The SROM-accelerated predictions were compared to the MCS reference

solution CDF to assess the accuracy and efficiency of each case.

To facilitate probabilistic prognosis with the SROM approach, a SROM must first be generated to approximate the random input parameters by solving the optimization problem in Eq. (9). The optimization was performed for the three cases ($m = 5, 10, 20$). Since the random model inputs are described by the collection of samples in this example, $\{\hat{\mathbf{x}}^{(k)}\}_{k=1}^{N}$, the sample-based statistics in Equations (13)-(15) were used in the objective function error terms (Equations (10)-(12)). Equal weight was assigned to each component of the objective function in Eq. (9) for simplicity (e.g., $\alpha_1 = \alpha_2 = \alpha_3 = 1.0$). In each case, the optimization problem took under 10 seconds to solve using a 2.8 GHz Intel Core i7 processor. The resulting optimal SROMs obtained for each model size are compared to the target random inputs in Figure 4, showing the CDFs for (a) y_0, (b) log$C$, and (c) n. It can be seen that the accuracy of the SROM CDF estimate improves with increasing model size. In particular, despite the simple piecewise constant approximation, the three input parameter distributions were well represented by the SROM with size $m = 20$.

With SROMs constructed for the inputs with each model size, the output to the crack growth simulation, $Y = [EOL]$, was approximated using a SROM surrogate model as described in Section 2.1.2. The simulation was executed for each sample defining the input SROMs according to Eq. (16) to produce a collection of corresponding EOL samples, $\{\tilde{y}^{(k)}\}_{k=1}^{m}$. Note that this step requires only 5, 10, and 20 crack growth model evaluations for each case (as compared to evaluating the model 5,000 times for every sample to produce the MCS solution). These EOL samples were then used to construct the piecewise constant SROM surrogate model, $\tilde{\mathbf{Y}}_C(\mathbf{X})$ (Eq. (17)), to approximate the EOL distribution. A comparison of the SROM CDF approximations using $\tilde{\mathbf{Y}}_C(\mathbf{X})$ and the reference MCS solution is shown in Figure 5(a) for $m = 5, 10$, and 20. It can be seen that with just a small fraction of the crack growth simulations relative to MCS, the SROM approach can provide a reasonably accurate estimate of the EOL distribution that improves with increasing SROM model size.

The piecewise linear SROM surrogate, $\tilde{\mathbf{Y}}_L(\mathbf{X})$ (Eq. (20)), was then constructed to illustrate the improvement in accuracy it provides. Here, additional crack growth simulations were performed to estimate the required gradients, $\nabla\tilde{\mathbf{y}}^{(k)}$, using the finite difference method, i.e.,

$$(\nabla\tilde{y}^{(k)})_i = \frac{\partial y(\mathbf{X})}{\partial x_i}\Big|_{\mathbf{X}=\tilde{\mathbf{x}}^{(k)}} \approx \frac{\mathcal{M}(\tilde{\mathbf{x}}^{(k)} + 1_i\Delta_i) - \mathcal{M}(\tilde{\mathbf{x}}^{(k)})}{\Delta_i}, \tag{26}$$

for the gradient with respect to the i^{th} input. Here, Δ_i represents a small perturbation to the i^{th} input parameter.

With the gradients and EOL samples calculated, the surrogate model, $\tilde{\mathbf{Y}}_L(\mathbf{X})$, was used to generate EOL samples from the entire collection of the given input samples, $\{\hat{\mathbf{x}}^{(k)}\}_{k=1}^{5000}$. A comparison of the resulting SROM CDFs and the MCS reference solution are provided in Figure 5(b) for each model size. It can be seen that the piecewise linear SROM approximation provides a substantial increase in accuracy, with all three cases providing nearly identical EOL predictions compared to the MCS solution. Note that for the $m = 5$ case, only 20 crack growth simulations were required in total compared with $5,000$ used to generate the MCS CDF, but very little discrepancy is observed between the two solutions.

Table 1 shows the approximate computational speedup relative to the MCS solution achieved by the different SROM approximations. Recall that the values of $S_{\tilde{\mathbf{Y}}_C}$ and $S_{\tilde{\mathbf{Y}}_L}$ assume that the time to construct the input SROM (< 10 seconds) is negligible compared to the time to run a crack growth simulation (~ 3 hours), which is valid for this example. The potential for orders of magnitude computational speedup is observed by generating probabilistic EOL predictions with only a fraction of the simulations used by MCS. In particular, using the piecewise linear model $\tilde{\mathbf{Y}}_L$ with $m = 5$ provides a $250X$ computational speedup while producing a highly accurate approximation to the MCS estimate, as shown in Figure 5(b).

There are two important points to make regarding the computational speedup results above. First, no rigorous convergence study was performed on the MCS predictions in this study, so it is possible that a satisfactory reference solution could require fewer or more model evaluations. Therefore, the values reported in Table 1 should be interpreted as potential computational speedup with the SROM approach, with further study needed to provide a more explicit assessment for this application. Second, while the results reported here were performed in serial (on one computer processor), multiple processors can be easily used for both MCS and the SROM approach since all model evaluations are independent of one another. The use of parallel computing in this manner would then decrease the computation times by a factor equal to the number of available processors.

3.3. `SROMPy` Demo

The purpose of this section is to demonstrate the relative ease of applying the SROM approach to probabilistic prognostics using `SROMPy`. The three steps for propagating uncertainty outlined in Section 2.1.3 will be followed to generate a SROM EOL approximation, showing the Python source code to carry out each step with `SROMPy`[7]. Here, it is assumed that two data files are provided: (1) `input_samples_MC.txt`, containing the 5,000 independent samples of the three input parameters, $[y_0, C, n]$, and (2) `eol_samples_MC.txt`, con-

[7]All source code and data for this example can be found here: https://github.com/nasa/SROMPy/examples/phm18.

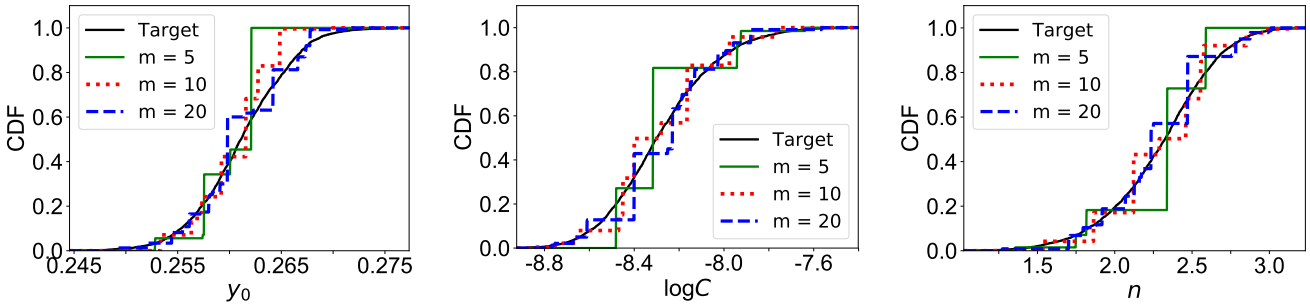


Figure 4. A comparison of the SROM and target CDFs for the input variables: (a) y_0, (b) log$C$, (c) n for different SROM sizes.

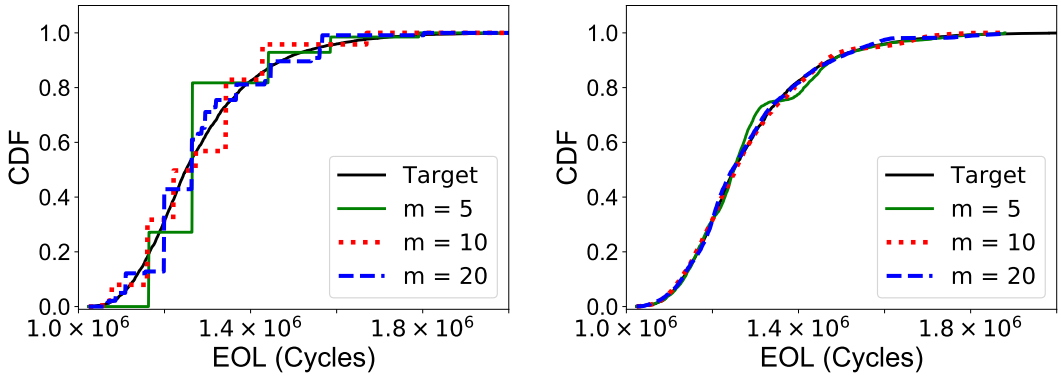


Figure 5. Predictions of EOL CDFs using the (a) piecewise constant and (b) piecewise linear SROM surrogate model for different model sizes.

Table 1. Approximate computational speedups, $S_{\tilde{\mathbf{Y}}_C}$ (Eq. (23)) and $S_{\tilde{\mathbf{Y}}_L}$ (Eq. (24)), provided by the piecewise constant and piecewise linear SROM EOL approximations, respectively, for different model sizes.

m	5	10	20
$S_{\tilde{\mathbf{Y}}_C}$	1000	500	250
$S_{\tilde{\mathbf{Y}}_L}$	250	125	62.5

taining the 5,000 corresponding EOL samples calculated by running the crack growth simulation for each input sample. The latter will be used to generate the MCS reference solution for comparison. Furthermore, the implementation of the crack growth model is assumed to be encapsulated in a Python object called `model`[8]. The example will be solved for the specific case of using a piecewise constant SROM approximation (Eq. (17)) to the EOL of the tensile specimen for a fixed SROM size, $m = 20$. A basic understanding of Python syntax is assumed. The full source code can be seen in the appendix of this paper.

3.3.1. Step 1: Construct input SROM

The first step in the probabilistic prognostics analysis is to define the target random vector for the model inputs, $\mathbf{X}$, and then generate a SROM approximation, $\tilde{\mathbf{X}}$, for it. The `SROMPy` implementation begins by importing the necessary Python modules for the analysis in its entirety, which will be elaborated on throughout the section:

```
import numpy
from postprocess import Postprocessor
from srom import SROM, SROMSurrogate
from target import SampleRV
```

The random input vector, $\mathbf{X}$, is then created from the data file that contains the input samples, `input_samples_MC.txt`, as follows:

```
#Define target random vector from samples
samplesfile = "input_samples_MC.txt"
MCsamples = numpy.genfromtxt(samplesfile)
target = SampleRV(MCsamples)
```

Here, the open-source Python module `numpy` (van der Walt, Colbert, & Varoquaux, 2011) is used to load the sample data into the array `MCsamples`. This array is then used to initialize `target`, representing $\mathbf{X}$, using the `SROMPy` class `SampleRV`.

With the target random input vector defined, `SROMPy` can be used to easily solve the optimization problem in Eq. (9) to form the SROM $\tilde{\mathbf{X}}$:

```
#Define SROM and determine optimal parameters
srom_size = 20
input_srom = SROM(size=srom_size, dim=3)
input_srom.optimize(target)
```

The SROM, $\tilde{\mathbf{X}}$, with a size $m = 20$ is represented by `input_srom`, which is optimized to match the target random vector. It can be seen that the complexities of solving Eq. (9) are encapsulated in the simple function, `optimize`, provided by the `SROM` class. To verify the SROM approximation of the input variables, the CDFs of $\tilde{\mathbf{X}}$ and $\mathbf{X}$ are then compared with the following code:

```
#Compare the CDFs (produces Figure 3)
pp = Postprocessor(input_srom, target)
pp.compare_CDFs(variablenames=
                    [r'log$C$', r'$y_{0}$', r'$n$'])
```

The resulting comparison plots can be seen in Figure 6.

3.3.2. Step 2: Execute model for each SROM sample

With a SROM generated for the random model inputs, the next step in the probabilistic prognostics analysis with `SROMPy` is to execute the crack growth model for each SROM sample (Eq. (16)). This is performed with Python as follows:

```
#Run the model for each input SROM sample:
srom_eols = numpy.zeros(srom_size)
(srom_samples, srom_probs)=input_srom.get_params()
for i, sample in enumerate(srom_samples):
    srom_eols[i] = model.evaluate(sample)
```

This code produces the collection of model output (EOL) samples, $\{\tilde{y}^{(k)}\}_{k=1}^{m}$, represented by the variable `srom_eols`. Here, the function `get_params()` returns the defining SROM samples and probabilities (Eq. (5)) in the variables `srom_samples` and `srom_probs`, respectively. The `evaluate` function of `model` runs a crack growth simulation for the given inputs in the `sample` variable and returns the corresponding EOL.

3.3.3. Step 3: Generate SROM output approximation

Now `SROMPy` can be used to straightforwardly produce a piecewise constant SROM approximation (Eq. (17)) to the EOL. This is done by using the SROM constructed in Step 1 (`input_srom`) and the output samples of EOL generated in Step 2 (`srom_eols`) as inputs to the `SROMSurrogate` class:

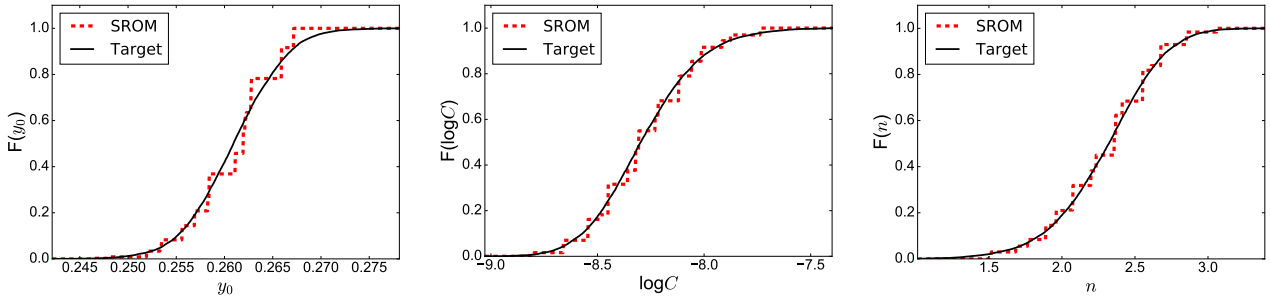
```
#Generate SROM surrogate for the EOL
eol_srom = SROMSurrogate(input_srom, srom_eols)
```

Here, the variable `eol_srom` now represents $\tilde{\mathbf{Y}}_C(\mathbf{X})$ in Eq. (17) whose statistics can be used to approximate those of the MCS reference solution.

To generate the reference solution, the EOL sample data in the file `eol_samples_MC.txt` is loaded and used to initialize the `SampleRV` class:

```
#Make random variable with MC EOL solution
mc_eol_file = "eol_samples_MC.txt"
MC_eols = numpy.genfromtxt(mc_eol_file)
eol_mc = SampleRV(MC_eols)
```

[8] Since the crack growth model depends on a commercial code, it cannot be included with the source code for this study. It is shown in the demo here to illustrate a typical SROM workflow. However, the input and output data from the model are provided online so that the uncertainty propagation results can be reproduced.



(a) (b) (c)

Figure 6. A comparison of the SROM and target CDFs for the input variables using `SROMPy`: (a) y_0, (b) log$C$, (c) n.

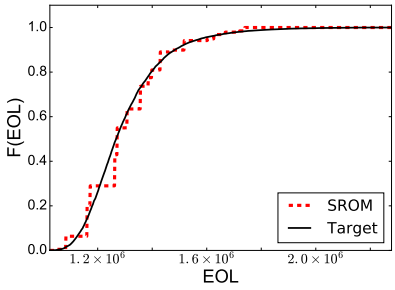


Figure 7. Prediction of the EOL CDF using a piecewise constant SROM surrogate using `SROMPy`.

The CDFs of the MCS solution, `eol_mc`, and the SROM approximation, `eol_srom`, can then be compared using the `Postprocessor` as follows:

```
#Compare final EOL solutions SROM vs MC:
# (produces Figure 7)
pp = Postprocessor(eol_srom, eol_mc)
pp.compare_CDFs(variablenames=["EOL"])
```

This code produces the comparison plot in Figure 7.

This example demonstrates that `SROMPy` can be used to provide accurate uncertainty propagation for prognostics with relatively few lines of code. Of the source code shown above, nearly half was devoted to generating the reference MCS solution and producing the comparison plots. The main responsibility of the user is in Step 2, generating outputs of their computational model for the SROM input samples produced by `SROMPy`. Note that while there are many more capabilities and advanced options in `SROMPy` for fine-tuning analyses that were not covered here in this simple case, the source code shown above can be easily generalized and extended to other prognostics problems.

4. SUMMARY

This study presented a practical uncertainty propagation method that enables efficient high-fidelity prognostics using stochastic reduced order models (SROMs). The open-source Python package, `SROMPy`, was also introduced in this work as a user-friendly option for adopting the SROM approach. The method was demonstrated on an example of probabilistic, non-planar crack growth simulation in an aluminum tensile specimen. The SROM approach produced end of life predictions that were highly accurate with respect to a Monte Carlo simulation (MCS) solution, while showing the potential to provide orders of magnitude computational speedup. Furthermore, the source code to generate a SROM solution for the example using `SROMPy` was included and discussed, illustrating the ease of applying the method for an arbitrary high-fidelity model. The complete source code and input data required to reproduce the results in this paper are available online to facilitate further evaluation and adoption of the SROM method in the field of prognostics and health management (PHM).

While demonstrated here in the context of prognostics for non-planar crack growth, the SROM approach has potential utility in a range of PHM applications as a general alternative to MCS for uncertainty propagation. By significantly reducing the number of required model evaluations relative to MCS, the SROM approach can allow expensive high-fidelity simulations to be used for probabilistic prognostics instead of less accurate (but faster) surrogate models. Alternatively, SROMs can be potentially combined with a surrogate model to enable real-time prognostics, an avenue of future research. The availability of the approach in `SROMPy` and the source code that accompanies this paper provides a basis for follow-on investigations.

REFERENCES

Arwade, S. J. (2005). Translation vectors with non-identically distributed components. *Probabilistic Engineering Mechanics*, *20*, 158-167.

Babuska, I., Nobile, F., & Tempone, R. (2007). A stochastic collocation method for elliptic partial differential equations with random input data. *SIAM J. Numer. Anal.*, *45*(3), 1005-1034.

Emery, J. M., Field, R. V., Foulk, J. W., Karlson, K. N., & Grigoriu, M. D. (2015). Predicting laser weld reliability with stochastic reduced-order models. *International Journal for Numerical Methods in Engineering*, *103*, 914-936.

Field, R. V., Grigoriu, M. D., & Emergy, J. M. (2015). On the efficacy of stochastic collocation, stochastic Galerkin, and stochastic reduced order models for solving stochastic problems. *Probabilistic Engineering Mechanics*, *41*, 60-72.

Fracture Analysis Consultants Inc. (2011). *FRANC3D Reference Manual, Version 6.*

Ghanem, R. G., & Spanos, P. D. (2003). *Stochastic finite elements: A spectral approach, revised edition*. New York, NY: Dover Publications, Inc.

Grigoriu, M. (1995). *Applied non-Gaussian processes: Examples, theory, simulation, linear random vibration, and matlab solutions*. Englewoods Cliffs, NJ: Prentice Hall.

Grigoriu, M. (2009). Reduced order models for random functions. Application to stochastic problems. *Applied Mathematical Modelling*, *33*, 161-175.

Grigoriu, M. (2011). A method for solving stochastic equations by reduced order models and local approximations. *Journal of Computational Physics*, *231*, 6495-6513.

Jones, E., Oliphant, T., Peterson, P., et al. (2001). *SciPy: Open source scientific tools for Python.*

Leser, P. E., Hochhalter, J. D., Warner, J. E., Newman, J. A., Leser, W. P., Wawrzynek, P. A., & Yuan, F. (2017). Probabilistic fatigue damage prognosis using surrogate models trained via three-dimensional finite element analysis. *Structural Health Monitoring*, *16*(3), 291-308.

Leser, P. E., Newman, J. A., Warner, J. E., Hochhalter, J. D., Leser, W. P., & Yuan, F. (2016, October). Probabilistic prognosis of non-planar fatigue crack growth. In *Annual conference of the prognostics and health management society.* Denver, CO.

Nobile, F., Tempone, R., & Webster, C. G. (2008). A sparse grid stochastic collocation method for partial differential equations with random input data. *SIAM J. Numer. Anal.*, *46*(5), 2309-2345.

Paris, P., & Erdogan, F. (1963). A critical analysis of crack propagation laws. *Journal of Basic Engineering*, *85*(4), 528-534.

Sankararaman, S., Ling, Y., Shantz, C., & Mahadevan, S. (2011). Uncertainty quantification in fatigue crack growth prognosis. *International Journal of Prognostics and Health Management*, *2*(1), 1-15.

Sarkar, S., Warner, J. E., Aquino, W., & Grigoriu, M. (2014). Stochastic reduced order models for uncertainty quantification of intergranular corrosion rates. *Corrosion Science*, *80*, 257-268.

van der Walt, S., Colbert, S. C., & Varoquaux, G. (2011). The NumPy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, *13*(2), 22-30. Retrieved from http://aip.scitation.org/doi/abs/10.1109/MCSE. doi: 10.1109/MCSE.2011.37

Warner, J. E. (2018a). Stochastic reduced order models with python (SROMPy). NASA/TM-2018-219824.

Warner, J. E. (2018b). *Stochastic reduced order models with python (SROMPy), Version 1.0. https://github.com/nasa/srompy.*

Warner, J. E., Aquino, W., & Grigoriu, M. (2015). Stochastic reduced order models for inverse problems under uncertainty. *Computer Methods in Applied Mechanics and Engineering*, *285*, 488-514.

Warner, J. E., Bomarito, G. B., Heber, G., & Hochhalter, J. D. (2016). Scalable implementation of finite elements by NASA - implicit (ScIFEi). NASA/TM-2016-219180.

Warner, J. E., Bomarito, G. F., Hochhalter, J. D., Leser, W. P., Leser, P. E., & Newman, J. A. (2017). A computationally-efficiency probabilistic approach to model-based damage diagonsis. *International Journal of Prognostics and Health Management*, *8*(2), 1-17.

Warner, J. E., Grigoriu, M., & Aquino, W. (2013). Stochastic reduced order models for random vectors. Application to random eigenvalue problems. *Probabilistic Engineering Mechanics*, *31*, 1-11.

APPENDIX - SROMPY DEMO SOURCE CODE

```
import numpy
from postprocess import Postprocessor
from srom import SROM, SROMSurrogate
from target import SampleRV

#Define target random vector from samples
samplesfile = "input_samples_MC.txt"
MCsamples = numpy.genfromtxt(samplesfile)
target = SampleRV(MCsamples)
#Define SROM and determine optimal parameters
srom_size = 20
input_srom = SROM(size=srom_size, dim=3)
input_srom.optimize(target)

#Compare the input CDFs (produces Figure 3)
pp = Postprocessor(input_srom, target)
pp.compare_CDFs(variablenames=
                    [r'log$C$', r'$y_{0}$', r'$n$'])
#Run the model for each input SROM sample:
srom_eols = numpy.zeros(srom_size)
(srom_samples, srom_probs)=input_srom.get_params()
for i, sample in enumerate(srom_samples):
    srom_eols[i] = model.evaluate(sample)

#Generate SROM surrogate for the EOL
eol_srom = SROMSurrogate(input_srom, srom_eols)

#Make random variable with MC EOL solution
mc_eol_file = "eol_samples_MC.txt"
MC_eols = numpy.genfromtxt(mc_eol_file)
eol_mc = SampleRV(MC_eols)
#Compare final EOL solutions SROM vs MC:
# (produces Figure 7)
pp = Postprocessor(eol_srom, eol_mc)
pp.compare_CDFs(variablenames=["EOL"])
```