Hindawi Publishing Corporation
Science and Technology of Nuclear Installations
Volume 2015, Article ID 839249, 17 pages
http://dx.doi.org/10.1155/2015/839249

Research Article

Demonstration of Emulator-Based Bayesian Calibration of Safety Analysis Codes: Theory and Formulation

Joseph P. Yurko,[1,2] Jacopo Buongiorno,[1] and Robert Youngblood[3]

[1]*MIT, 77 Massachusetts Avenue, Cambridge, MA 02139, USA*
[2]*FPoliSolutions, LLC, 4618 Old William Penn Highway, Murrysville, PA 15668, USA*
[3]*INL, P.O. Box 1625, Idaho Falls, ID 83415-3870, USA*

Correspondence should be addressed to Joseph P. Yurko; jyurko@fpolisolutions.com

Received 16 January 2015; Revised 1 April 2015; Accepted 28 May 2015

Academic Editor: Francesco Di Maio

System codes for simulation of safety performance of nuclear plants may contain parameters whose values are not known very accurately. New information from tests or operating experience is incorporated into safety codes by a process known as calibration, which reduces uncertainty in the output of the code and thereby improves its support for decision-making. The work reported here implements several improvements on classic calibration techniques afforded by modern analysis techniques. The key innovation has come from development of code surrogate model (or code emulator) construction and prediction algorithms. Use of a fast emulator makes the calibration processes used here with Markov Chain Monte Carlo (MCMC) sampling feasible. This work uses Gaussian Process (GP) based emulators, which have been used previously to emulate computer codes in the nuclear field. The present work describes the formulation of an emulator that incorporates GPs into a factor analysis-type or pattern recognition-type model. This "function factorization" Gaussian Process (FFGP) model allows overcoming limitations present in standard GP emulators, thereby improving both accuracy and speed of the emulator-based calibration process. Calibration of a friction-factor example using a Method of Manufactured Solution is performed to illustrate key properties of the FFGP based process.

1. Introduction

Propagating input parameter uncertainty for a nuclear reactor system code is a challenging problem due to often nonlinear system response to the numerous parameters involved and lengthy computational times, issues that compound when a statistical sampling procedure is adopted, since the code must be run many times. Additionally, the parameters are sampled from distributions that are themselves uncertain. Current industry approaches rely heavily on expert opinion for setting the assumed parameter distributions. Observational data is typically used to judge if the code predictions follow the expected trends within reasonable accuracy. All together, these shortcomings lead to current uncertainty quantification (UQ) efforts relying on overly conservative assumptions, which ultimately hurt the economic performance of nuclear energy.

This work adopts a Bayesian framework that allows reducing computer code predictive uncertainty by calibrating parameters directly to observational data; this process is also known as solving the inverse problem. Unlike the current heuristic calibration approach, Bayesian calibration is systematic and statistically rigorous, as it calibrates the parameter distributions to the data, not simply tune point values. With enough data, any biases from expert opinion on the starting parameter distributions can be greatly reduced. Multiple levels of data are easier to handle as well, since Integral and Separate Effect Test (IET and SET) data can be used simultaneously in the calibration process. However, implementing Bayesian calibration for safety analysis codes is very challenging. Because the posterior distribution cannot be obtained analytically, approximate Bayesian inference with sampling is required. Markov Chain Monte Carlo (MCMC) sampling algorithms are very powerful and have become increasingly widespread over the last decade [1]. However, for even relatively fast computer models practical implementation of Bayesian inference with MCMC would simply take too long because MCMC samples must be drawn in series.

As an example, a computer model that takes 1 minute to run but needs 10^5 MCMC samples would take about 70 days to complete. A very fast approximation to the system code is thus required to use the Bayesian approach. Surrogate models (or *emulators*) that emulate the behavior of the input/output relationship of the computer model but are computationally inexpensive allow MCMC sampling to be possible. An emulator that is 1000x faster than the computer model would need less than two hours to perform the same number of MCMC samples. As the computer model run time increases, the surrogate model becomes even more attractive because MCMC sampling would become impractically lengthy.

Gaussian Process- (GP-) based emulators have been used to calibrate computer code for a variety of applications. Please consult [2–5] for specific cases as well as reviews of other sources. This work applies a relatively new class of statistical model, the function factorization with Gaussian Process (FFGP) priors model, to emulate the behavior of the safety analysis code. The FFGP model builds on the more commonly used GP emulator but overcomes certain limiting assumptions inherent in the GP emulator, as will be explained later. The FFGP model is therefore better suited to emulate the complex time series output produced by the system code. The surrogate is used in place of the system code to perform the parameter calibration, thereby allowing the observational data to directly improve the current state of knowledge.

The rest of this paper is organized as follows. An overview of the entire emulator-based Bayesian calibration process is described in Section 2. Section 3 discusses the emulators in detail. The first half of Section 3 summarizes the important expressions related to GP emulators. Most of these expressions can be found in numerous other texts and references on GP models, including [6, 7]. They are repeated in this paper for completeness as well as providing comparison to the FFGP expressions in the latter half of Section 3. Section 4 presents a method of manufactured solutions-type demonstration problem that highlights the benefits of the FFGP model over the standard GP model.

2. Overview of Emulator-Based Bayesian Calibration

As already stated, the emulator-based approach replaces the potentially very computationally expensive safety analysis code (also known as a simulator, computer code, system code, or simply the code) with a computationally inexpensive surrogate. Surrogate models are used extensively in a wide range of engineering disciplines, most commonly in the form of response surfaces and look-up tables. Reference [4] provides a thorough review of many different types of surrogate models. The present work refers to the surrogates as *emulators* to denote that they provide an estimate of their own uncertainty when making a prediction [5]. An emulator is therefore a probabilistic response surface which is a very convenient approach because the emulator's contribution to the total uncertainty can be included in the Bayesian calibration process. An uncertain (noisy) emulator would therefore limit the parameter posterior precision, relative to calibrating the parameters using the long-running computer code itself. Obviously, it is desirable to create an emulator that is as accurate as possible relative to the computer code, which limits the influence of error and uncertainty on the results.

The emulator-based approach begins with choosing the input parameters and their corresponding prior distributions. If the emulator was not used in place of the system code, the Bayesian calibration process would start in the exact same manner. The priors encode the current state of knowledge (or lack thereof) about each of the uncertain input parameters. Choice of prior for epistemically uncertain variables is controversial and relies heavily on expert opinion. Justification for the priors used in the applications of this work is given later on, but choice of the priors is not the focus of this work. Additionally, the choice of the specific input parameters to be used for calibration may be controversial. Dimensionality reduction techniques might be used to help screen out unimportant input parameters [4]. Some screening algorithms such as the Reference Distribution Variable Selection (RDVS) algorithm use GPs to identify statistically significant input parameters [8]. In the nuclear industry specifically, expert opinion-based Phenomena Identification and Ranking Tables (PIRTs) are commonly used to down-select the most important physical processes that influence a Figure of Merit (FOM) [9]. More recently, Quantitative PIRTs, or QPIRTs, have been used in place of the traditional expert opinion PIRTs to try to remove bias and to capture relevant physical processes as viewed by the computer code [10, 11]. No matter the approach, the set of input parameters and their corresponding prior distribution must be specified.

In the emulator-based approach, the prior has the additional role of aiding in choosing the training set on which the emulator is based. As the phrase implies, the training set is the sequence of computer code evaluations used to build or train the emulator. Once trained on selected inputs and outputs, the emulator reflects the complex input/output relationship, so training is clearly an essential piece of the emulator-based approach. There are numerous methods and decision criteria for the selection of the training set; see [4, 5] for more details. Reference [12] provides an excellent counter point for the dangers of not using enough points in generating the training set. This work does not focus on choosing the "optimal" or "best" training set, which is an active area of research. The input parameter prior is used to set bounds on the input parameter values; Latin Hypercube Sampling (LHS) is then used to create a "space filling" design within those bounds. Although not guaranteed to produce the best possible training set, this method adequately covers the prior range of possible input parameter values. An active area of research is how to enhance the training set during the calibration process itself, in order to focus more on the posterior range of possible values.

With the training input values chosen, the computer code is run the desired number of times to generate the training output. The complete training set is then the training input values with their corresponding training output. The emulator is then built by learning specific characteristics that allow the emulator to represent the input/output relationship encoded in the training set. The specific characteristics that must be learned depend on the type of emulator being used.

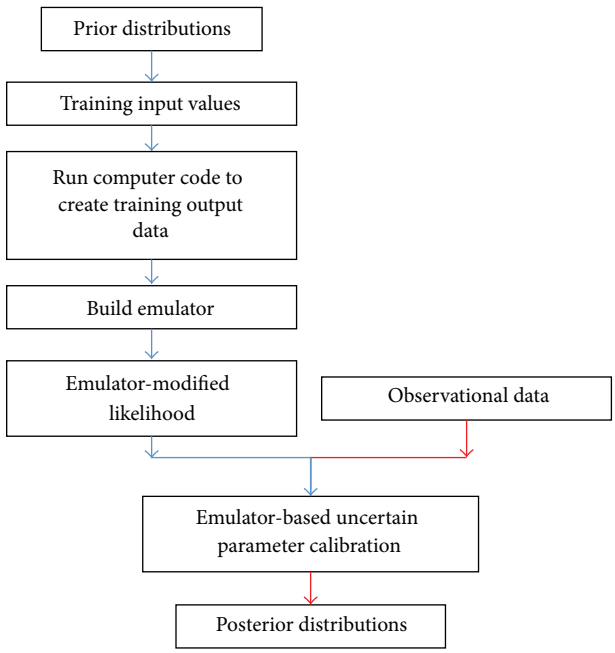

Figure 1: Emulator-based Bayesian calibration flow chart.

Training algorithms for the standard GP emulator and FFGP emulator are described in Section 3.

Once trained, the emulator is used in place of the computer code in the MCMC sampling via an emulator-modified likelihood function. The modified likelihood functions are presented in Section 3 for each of the emulators used in this work. Regardless of the chosen type of emulator, the emulator-based calibration process results in uncertain input parameter posterior distributions and posterior-approximated predictions, conditioned on observational data. A flow chart describing the key steps in the emulator-based Bayesian calibration process is shown in Figure 1.

The emulator-based Bayesian calibration process presented in this work fixes the emulator once it is trained. Alternatively, the emulator could be constructed simultaneously with the calibration of the uncertain input parameters [2, 3]. The key difference between the two approaches is that the emulator-modified likelihood function in [2, 3] is not fixed since the emulator is not fixed. Formally, the alternative approach bases the emulator-modified likelihood function around the emulator prior predictive distribution whereas the work presented in this paper bases the emulator-modified likelihood function around the emulator posterior predictive distribution. The difference between the posterior and prior predictive distributions is described in detail in Section 3.2.4. The alternative approach therefore makes emulator predictions conditioned on both the training data and the observational data simultaneously. In some sense, the alternative approach is more of a data or information "fusion" method rather than a calibration focused approach. The drawback of the alternative "data fusion" approach is that the emulator is not built until after the entire Bayesian calibration process is complete. Thus, if multiple levels of data such as from IETs and SETs are present, the emulators for all of the IETs and SETs must be calibrated simultaneously, which considerably complicates and slows the MCMC sampling. For those reasons, this work does not use the "data fusion" approach but fixes the emulator before starting the calibration of the uncertain input parameters.

3. Gaussian Process-Based Emulators

3.1. Overview. The emulators used in this work are based on Gaussian Process (GP) models and are considered Bayesian nonparametric statistical models. Nonparametric models offer considerably more flexibility than parametric models because the input/output functional relationship does not have to be assumed *a priori* by the user. The training data dictates the input/output relationship, just as a look-up table functions. As stated earlier, the emulator is a probabilistic model; therefore, the emulators are essentially probabilistic look-up tables. Nonparametric models are however considerably more computationally intensive than parametric models, because the training data is never discarded. If a large number of training runs are required to accurately capture the input/output trends, a nonparametric model might be considerably slower to run than a parametric model of the same data (e.g., a curve that fits the data).

The underlying principles of the GP model were developed in the 1960s in the geostatistics field where it was known as *Kriging* [4]. Since then Kriging has been widely used for optimization, but starting in the late 1980s and early 1990s, [13–15] popularized the approach as Bayesian approximations to deterministic computer codes. In the early 2000s, Kennedy and O'Hagan used the GP model to facilitate Bayesian calibration of computer codes [16]. Their work served as the foundation for this paper and many of the references cited in the previous section.

The machine learning community has also extensively used GP models for both regression and classification (regression is used for continuous functions while classification is used for discrete data) [6, 7]. Even with all of their flexibility, GP models are still somewhat limited by certain underlying assumptions to be discussed later, as well as the limitation in handling very large datasets (just as with any nonparametric model). In order to overcome these limitations and handle more complicated input/output relationships, many different approaches have been developed [6]. One such approach is based on combining GP models with factor analysis techniques; this is referred to as Gaussian Process Factor Analysis (GPFA) models [17, 18]. The work presented here uses the factor analysis based approach in order to handle very large datasets following the formulation of Schmidt [17].

3.2. Standard Gaussian Process (GP) Emulators

3.2.1. Formulation. Within the Bayesian framework, a Gaussian Process (GP) prior is placed on the computer code's unknown output. The computer code, such as RELAP, is actually deterministic, meaning that the same output will result if the same input parameters and settings are used over

and over. The output, however, is in some sense unknown until the computer code is run, and it will therefore be treated as a random variable. A GP is a collection of random variables, any finite number of which have a jointly Gaussian distribution [6]. A Gaussian Process is simply a multivariate normal (MVN) distribution and is used presently as a prior distribution on the computer code input/output functional relationship.

The input $\mathbf{x}$ will be all D inputs to the computer code that the GP is trying to emulate: $\mathbf{x} = [x_1, x_2, \ldots, x_d, \ldots, x_D]^{\mathrm{T}}$. The superscript T denotes the transpose of the vector. The output, $f(\mathbf{x})$, as stated above, is considered a random variable. A GP is completely specified by its mean function and covariance function. The mean function $m(\mathbf{x})$ and covariance function $k(\mathbf{x}, \mathbf{x}')$ are defined as [6]

$$\begin{aligned} m(\mathbf{x}) &= \mathbb{E}\left[f(\mathbf{x})\right], \\ k\left(\mathbf{x}, \mathbf{x}'\right) &= \mathbb{E}\left[\left(f(\mathbf{x}) - m(\mathbf{x})\right)\left(f\left(\mathbf{x}'\right) - m\left(\mathbf{x}'\right)\right)\right]. \end{aligned} \tag{1}$$

The GP is then defined as

$$f(\mathbf{x}) \sim \mathcal{GP}\left(m(\mathbf{x}), k\left(\mathbf{x}, \mathbf{x}'\right)\right). \tag{2}$$

An important aspect of (2) is that the covariance between the outputs is written *only* as a function of the inputs. This is a key assumption in the simplicity of standard GP models, since all the covariance between two outputs depends only on the values of the inputs that produced those two outputs.

Following [6], as well as many other sources, the mean function is usually taken to be zero. Besides being the simplest approach to use, a zero mean function gives no prior bias to the trend in the data, since no mean trend is assumed. Covariance functions themselves depend on a set of *hyperparameters*; therefore, even though the GP is a nonparametric model, these hyperparameters specify the covariance function and must be learned from the training data. However, the GP model is still considered a nonparametric model, because a prediction still requires regressing the training dataset. Numerous covariance functions exist, ranging from very simple forms to very complex neural net-like functions [6]. Different forms have various advantages/disadvantages for different datasets, but the most common type used in the literature is the squared-exponential (SE) covariance function. The SE covariance function is usually parameterized as

$$k\left(\mathbf{x}_p, \mathbf{x}_q\right) = \sigma_f^2 \exp\left(-\frac{1}{2}\left(\mathbf{x}_p - \mathbf{x}_q\right)^{\mathrm{T}} M\left(\mathbf{x}_p - \mathbf{x}_q\right)\right), \tag{3}$$

where the subscripts p and q denote (potentially) two different values for the D-dimensional input vector $\mathbf{x}$. The hyperparameters in (3) are the signal variance σ_f^2 and the matrix M, which is a symmetric matrix that is usually parameterized as a diagonal matrix:

$$M = \mathrm{diag}(l)^{-2}. \tag{4}$$

Each diagonal element of M is a separate hyperparameter, l_d, which serves as the characteristic length scale for the dth input. Loosely speaking, the length scale represents how far the input value must move along a particular axis in input space for the function values to become uncorrelated [6]. Since each input parameter has its own unique length scale, this formulation implements what is known as automatic relevance determination (ARD), since the inverse of the length scale determines how relevant that input is. If the length has a very large value, the covariance will become almost independent of that input. Linkletter et al. [8] used ARD to screen out unimportant inputs using GP models.

Strictly speaking, the GP model can interpolate the training data exactly if no noise is allowed between the training data and the GP prior. However, implementation of an interpolating GP model might be difficult due to ill-conditioning issues [5, 6], which will be discussed later on. Allowing some hopefully very small noise between the GP prior and training data removes the numerical issues and turns the model into a GP regression (GPR) model. The GP prior is therefore actually placed on a latent (hidden) function, $f(\mathbf{x})$, that must be inferred from the noisy data y [6]. This viewpoint brings to light the signal processing nature of the GPR framework, since the latent function is the true signal that must be inferred from the noisy data. In emulating computer codes, the training output is not noisy, but this setup provides a useful mathematical framework. The computer model output of interest, y, is then related to the GP latent function $f(\mathbf{x})$ as

$$y = f(\mathbf{x}) + \epsilon, \tag{5}$$

where ϵ is the error or noise. The error can take a variety of forms, but if a Gaussian likelihood model is used with independent and identically distributed (IID) noise, with zero mean and variance σ_n^2, the remaining calculations are all analytically tractable. More complicated likelihood models can be used and are often required to handle very complex datasets, but the remaining calculations would no longer have analytical expressions.

At this point, some important notation needs to be defined. If there are a total of N training points, the inputs are stacked into an $N \times D$ matrix of all training input values:

$$X = \begin{bmatrix} \mathbf{x}_1^{\mathrm{T}} \\ \mathbf{x}_2^{\mathrm{T}} \\ \vdots \\ \mathbf{x}_N^{\mathrm{T}} \end{bmatrix}. \tag{6}$$

Each row of X contains the D input parameters for that particular training case run. The training outputs, y, are stacked into a vector of size $N \times 1$: $\mathbf{y} = [y_1, y_2, \ldots, y_N]^{\mathrm{T}}$. Since $f(\mathbf{x})$ has a GP prior and the likelihood function is also Gaussian, the latent variables can be integrated yielding a Gaussian distribution on the training output [6]:

$$\mathbf{y} \sim \mathcal{N}\left(\mathbf{0}, \mathbf{K}(X, X) + \sigma_n^2 \mathbb{I}\right). \tag{7}$$

In (7), $\mathbf{K}(X, X)$ is the training set covariance matrix and $\mathbb{I}$ is the identity matrix. The training set covariance matrix is

built by applying the covariance function between each pair of input parameter values [6]:

$$\mathbf{K}(X,X) = \begin{bmatrix} k(\mathbf{x}_1,\mathbf{x}_1) & k(\mathbf{x}_1,\mathbf{x}_2) & \cdots & k(\mathbf{x}_1,\mathbf{x}_N) \\ k(\mathbf{x}_2,\mathbf{x}_1) & k(\mathbf{x}_2,\mathbf{x}_2) & \cdots & k(\mathbf{x}_2,\mathbf{x}_N) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_N,\mathbf{x}_1) & k(\mathbf{x}_N,\mathbf{x}_2) & \cdots & k(\mathbf{x}_N,\mathbf{x}_N) \end{bmatrix}. \quad (8)$$

The training set covariance matrix is therefore a full matrix. If the SE covariance function in (3) is used, each diagonal element of $\mathbf{K}(X,X)$ is equal to the signal variance, σ_f^2. Evaluating the covariance function, however, requires the hyperparameter values to be known, which is accomplished by training the GP emulator.

3.2.2. Training. Training or building the emulator consists of learning the hyperparameters that define the covariance and likelihood functions. As discussed earlier, there are two types of hyperparameters in the SE covariance function, the signal variance, σ_f^2, and the length scales, l. The Gaussian likelihood function used presently consists of one hyperparameter, the likelihood noise (variance), σ_n^2. The complete set of hyperparameters is denoted by $\phi = \{\sigma_f^2, l, \sigma_n^2\}$.

Two ways to learn the hyperparameters will be discussed here: the empirical Bayes approach and the full Bayesian approach. "Full Bayesian" refers to inferring the hyperparameter posterior distribution given the training data. Due to the complexity of the relationship between the hyperparameters and the training output, sampling based Markov Chain Monte Carlo (MCMC) inference is required to perform the full Bayesian approach. The "empirical Bayes" method summarizes the hyperparameters with point estimates. The hyperparameter contribution to the output uncertainty is therefore neglected, but, as discussed by many authors, this is an acceptable approximation [4, 5]. The entire GP model is still considered Bayesian because the GP itself is a statement of the probability of the latent function, which can then make a statement of the probability of the output. The point estimates can be found either from sampling-based approaches or by optimization methods. With optimization procedures, the empirical Bayes approach would be much faster than the full Bayesian training approach. However, cross-validation is very important to ensure the optimizer did not get "stuck" at a local optimum [6].

However, this work used a hybrid approach to training. MCMC sampling was used to draw samples of the hyperparameter posterior distribution just as in the full Bayesian approach. The hyperparameters were then summarized as point estimates at the posterior mean values. Using point estimates greatly reduced the computer memory required to make predictions (which are described later). The sampling based approach removed having to perform cross-validation since the point estimates correspond to the values that on average maximize the posterior density.

The prior distribution on the set of hyperparameters, known as the *hyperprior,* must be specified as part of the MCMC sampling procedure. The simplest hyperprior would be the "flat" improper hyperprior, $p(\phi) \propto 1$; however for GP models the input and output can be scaled to facilitate meaningful hyperprior specification. Following [2, 3, 5], the inputs used in this work are all scaled between 0 and 1, where 0 and 1 correspond to the minimum and maximum training set value, respectively. Additionally, the training output data are scaled to a standard normal, with mean 0 and variance 1. Since the signal variance, σ_f^2, defines the diagonal elements of the covariance matrix, it is biased to be near 1. The likelihood noise, σ_n^2, is biased to be a small value using a Gaussian distribution with prior mean of 10^{-6}. This hyperprior format biases the sampling procedure to try to find length scale values that match the training output within this noise tolerance. The length scale hyperpriors are more difficult to set, but the formulation from Higdon was used [2, 3, 8], which *a priori* biases the length scales to yield smooth input/output relationships. Only the training data can reduce the length scales; therefore only the training data can dictate if an input strongly influences the output variability.

Additionally, a small "nugget" or "jitter" term was added to the diagonal elements of $\mathbf{K}(X,X)$. The nugget term is rarely mentioned outside of footnotes in most references in the literature [6], but it is a very important part of practical implementations of GP models. The nugget adds a small amount of additional noise, preventing a GP model from interpolating the training set exactly. This additional noise may be very useful at preventing the training set covariance matrix from being ill-conditioned. There have been some detailed investigations into the nugget's influence on the training algorithm results [5], but for practical purposes the nugget is a simple way to make sure the covariance matrix is always invertible.

The hyperparameter posterior, up to a normalizing constant, can be written as

$$p(\phi \mid \mathbf{y}) \propto p(\mathbf{y} \mid \phi)\, p(\phi). \quad (9)$$

In (9), $p(\phi)$ is the hyperprior described previously and the likelihood function, $p(\mathbf{y} \mid \phi)$, is (7) rewritten to explicitly depend on the hyperparameters. Hyperparameter posterior samples were drawn using the Adaptive Metropolis (AM) MCMC algorithm [19]. The AM-MCMC algorithm improves the efficiency of the basic Random Walk Metropolis (RWM) sampling algorithm because the MCMC proposal distribution covariance matrix is empirically computed using the previous samples. Regardless of the type of MCMC algorithm used, the likelihood function must be evaluated for each MCMC sample. The log-likelihood, written up to a normalizing constant, is [6]

$$\log\left[p(\mathbf{y} \mid \phi)\right] \propto -\frac{1}{2}\mathbf{y}^{\mathrm{T}}\left[\mathbf{K}(X,X) + \sigma_n^2\mathbb{I}\right]^{-1}\mathbf{y} - \frac{1}{2}\log\left[\left|\mathbf{K}(X,X) + \sigma_n^2\mathbb{I}\right|\right]. \quad (10)$$

Equation (10) clearly shows that the training set covariance matrix must be inverted at each MCMC sample. This highlights why the nugget term is useful, if for a particular sample

the likelihood noise, σ_n^2, does not provide enough noise to allow the matrix to be inverted. The inversion of the training set covariance matrix is the most computationally expensive part of the training algorithm.

3.2.3. Predictions. Once the emulator is trained, predictions can be made at input values that were not part of the training set. If there are N_* new test or prediction points, the test input matrix, X_*, is size $N_* \times D$. Under the GP model framework, the latent function at those new test points has the same GP prior as the training points:

$$\mathbf{f}_* \sim \mathcal{N}\left(\mathbf{0}, \mathbf{K}\left(X_*, X_*\right)\right). \tag{11}$$

Comparing (11) with the training output GP prior in (7), the key difference is that the covariance matrix is evaluated at the test input values rather than the training input values. As written, the test prior provides very little useful information, since it has no information regarding the structure of the training dataset. The test latent output must therefore be conditioned on the training output. The joint prior is a multivariate normal distribution [6]:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} \mathbf{K}\left(X, X\right) + \sigma_n^2 \mathbb{I} & \mathbf{K}\left(X, X_*\right) \\ \mathbf{K}\left(X_*, X\right) & \mathbf{K}\left(X_*, X_*\right) \end{bmatrix}\right), \tag{12}$$

where $\mathbf{K}(X, X_*)$ is the cross-covariance matrix between the training and test input values. The cross-covariance matrix is size $N \times N_*$ and $\mathbf{K}(X_*, X)$ is its transpose. Standard multivariate normal theory easily allows computing the conditional distribution $p(\mathbf{f}_* \mid \mathbf{y})$ which gives the key predictive equations for the GPR model [6]:

$$\mathbf{f}_* \mid \mathbf{y} \sim \mathcal{N}\left(\bar{\mathbf{f}}_*, \operatorname{cov}\left(\mathbf{f}_*\right)\right), \tag{13}$$

with the posterior predictive mean given as

$$\bar{\mathbf{f}}_* \triangleq \mathbb{E}\left[\mathbf{f}_* \mid \mathbf{y}\right] = \mathbf{K}\left(X_*, X\right)\left[\mathbf{K}\left(X, X\right) + \sigma_n^2 \mathbb{I}\right]^{-1} \mathbf{y}, \tag{14}$$

and the posterior predictive covariance is

$$\begin{aligned} &\operatorname{cov}\left(\mathbf{f}_*\right) \\ &\quad = \mathbf{K}\left(X_*, X_*\right) \\ &\qquad - \mathbf{K}\left(X_*, X\right)\left[\mathbf{K}\left(X, X\right) + \sigma_n^2 \mathbb{I}\right]^{-1} \mathbf{K}\left(X, X_*\right). \end{aligned} \tag{15}$$

The posterior predictive distribution of the test targets, $\mathbf{y}_*$, is the same as the latent posterior predictive distribution except the additional likelihood noise is added:

$$\mathbf{y}_* \sim \mathcal{N}\left(\bar{\mathbf{f}}_*, \operatorname{cov}\left(\mathbf{f}_*\right) + \sigma_n^2 \mathbb{I}\right). \tag{16}$$

Equations (14) and (15) reveal the important features of the GP emulator. First, the posterior predictive covariance shrinks the prior test covariance as witnessed by the subtraction between the first and second terms on the right-hand side of (15). Second, when making predictions at the training points ($X_* = X$), the predictive uncertainty shrinks to the allowable error tolerance.

3.2.4. Gaussian Process-Based Calibration. Once the GP emulator is constructed, it can be used to calibrate the uncertain input parameters, in place of the computer code. Before going into detail of the emulator-based calibration calculations, Bayesian calibration of the computer code itself is reviewed. The computer code functional relationship is denoted as $\mathbf{y}(\mathbf{x}_{\text{cv}}, \theta)$, where $\mathbf{x}_{\text{cv}}$ is the set of control variables that are not uncertain and θ are the uncertain input parameters. Control variables are conditioned controlled by experimenters or in a transient could also include time. If the computer code could be used as part of the MCMC sampling, the uncertain parameter posterior distribution (up to a normalizing constant) could be written as

$$p\left(\theta \mid \mathbf{y}_o\right) \propto p\left(\mathbf{y}_o \mid \mathbf{y}\left(\mathbf{x}_{\text{cv},o}, \theta\right)\right) p\left(\theta\right). \tag{17}$$

In (17), $\mathbf{y}_o$ refers to the observational (experimental) data and $\mathbf{x}_{\text{cv},o}$ are the control variables' *locations* for the observational data. The computer code therefore acts as a (potentially very nonlinear) mapping function between the uncertain inputs and the observational data. As discussed previously, the computer code is computationally too expensive and the emulator is used in place of the computer code for Bayesian calibration. To facilitate the emulator-based calibration, the likelihood function between the computer prediction and the observational data is split into a hierarchical-like fashion. The total likelihood consists of two parts. The first component is the likelihood between the observational data and the prediction, $p(\mathbf{y}_o \mid \mathbf{y})$. The second part is the likelihood between the computer prediction and the uncertain inputs, $p(\mathbf{y} \mid \mathbf{x}_{\text{cv}}, \theta)$. The posterior distribution is now the joint posterior distribution between the uncertain inputs and the computer predictions, both conditioned on the observational data:

$$p\left(\mathbf{y}, \theta \mid \mathbf{y}_o\right) \propto p\left(\mathbf{y}_o \mid \mathbf{y}\right) p\left(\mathbf{y} \mid \mathbf{x}_{\text{cv},o}, \theta\right) p\left(\theta\right). \tag{18}$$

The likelihood between the observational data and the computer predictions, $p(\mathbf{y}_o \mid \mathbf{y})$, is the assumed likelihood model for the experiment. This work uses a Gaussian likelihood with known independent measurement errors at each of the observational data points. Assuming N_o independent data points, the likelihood function factorizes as

$$p\left(\mathbf{y}_o \mid \mathbf{y}\right) = \prod_{l=1}^{N_o} p\left(y_{o,l} \mid y_l\right) = \prod_{l=1}^{N_o} \mathcal{N}\left(y_l, \sigma_{\epsilon,l}^2\right), \tag{19}$$

where $\sigma_{\epsilon,l}^2$ is the measurement error variance for the lth observational data point. The likelihood between computer prediction and the inputs, $p(\mathbf{y} \mid \mathbf{x}_{\text{cv}}, \theta)$, is almost impossible to write analytically because of the very complex nature of the computer code. However, $p(\mathbf{y} \mid \mathbf{x}_{\text{cv}}, \theta)$ can be approximated using the emulator which leads to the emulator-modified likelihood function. As discussed in Section 2, there are two ways to accomplish this. The alternate "data fusion" approach of [2, 3] uses the GP prior distribution to approximate $p(\mathbf{y} \mid \mathbf{x}_{\text{cv}}, \theta)$. This work however uses the GP posterior predictive distribution, of the already built emulator, to approximate $p(\mathbf{y} \mid \mathbf{x}_{\text{cv}}, \theta)$. The training set is denoted as a whole as

$\mathscr{D} = \{\mathbf{y}, X\}$, and the hyperparameters are assumed to be already determined as part of the training algorithm. The joint posterior between the emulator estimated predictions $\mathbf{y}_*$ and the uncertain inputs is

$$p(\mathbf{y}_*, \theta \mid \mathbf{y}_o, \mathscr{D}, \phi) \propto p(\mathbf{y}_o \mid \mathbf{y}_*)\, p(\mathbf{y}_* \mid \{\mathbf{x}_{\mathrm{cv},o}, \theta\}, \mathscr{D}, \phi)\, p(\theta). \tag{20}$$

In (20), $p(\mathbf{y}_* \mid \{\mathbf{x}_{\mathrm{cv},o}, \theta\}, \mathscr{D}, \phi)$ is exactly the same as (16), except that it is explicitly written to depend on the training set and hyperparameters. Since the GP posterior predictive distribution is Gaussian and the likelihood between the observational data and computer prediction is also Gaussian, the emulator predictions can be integrated out of (20). The (integrated) posterior distribution on the uncertain inputs conditioned on the observational data is then

$$p(\theta \mid \mathbf{y}_o, \mathscr{D}, \phi) \propto p(\mathbf{y}_o \mid \{\mathbf{x}_{\mathrm{cv},o}, \theta\}, \mathscr{D}, \phi)\, p(\theta). \tag{21}$$

The likelihood between the uncertain inputs and the observational data is the *GP emulator-modified likelihood function* equal to the GP posterior predictive distribution with the measurement error added to the predictive variance:

$$\mathbf{y}_o \mid \{\mathbf{x}_{\mathrm{cv},o}, \theta\}, \mathscr{D}, \phi \sim \mathcal{N}\left(\bar{\mathbf{f}}_*, \mathrm{cov}(\mathbf{f}_*) + \sigma_n^2 \mathbb{I} + \Sigma_\epsilon\right). \tag{22}$$

In (22), Σ_ϵ is the measurement error covariance matrix which is assumed to be diagonal. If more complicated likelihood functions between the observational data and computer prediction were assumed, (21) and (22) would potentially be very different and even require approximations. Equation (22) also provides the direct comparison with the "data fusion" approach described in Section 2. The emulator-modified likelihood function given by equation 4 in [2] uses the GP prior mean and covariance matrix, while this work uses the GP posterior predictive mean and covariance matrix.

3.3. Function Factorization with Gaussian Process (FFGP) Priors Emulators. For very large datasets, the inverse of the training set covariance matrix might be too expensive to compute. Typically, "very large" corresponds to training sets with over 10,000 points. These situations can occur for several reasons, the obvious being that a large number of computer code evaluations are required. Training sets become very large when the goal is to emulate multiple outputs, especially for time series predictions. If 100 points in time are taken from a single computer code evaluation (referred to as a case run) and 100 cases are required to cover the ranges of the uncertain variables, the total training set consists of 10,000 points.

As stated previously, there are various solutions to this issue, most of which involve some form of a dimensionality reduction technique. The function factorization approach used in this work embeds the dimensionality reduction as part of the emulator through factor analysis techniques. The following sections describe the formulation and implementation of the function factorization model.

3.3.1. Formulation. The main idea of function factorization (FF) is to approximate a complicated function, $y(\mathbf{x})$, on a high dimensional space, $\mathscr{X}$, by the sum of products of a number of simpler functions, $f_{i,k}(\mathbf{x}_i)$, on lower dimensional subspaces, $\mathscr{X}^i$. The FF-model is [17]

$$y(\mathbf{x}) \approx \sum_{k=1}^{K} \prod_{i=1}^{I} f_{i,k}(\mathbf{x}_i). \tag{23}$$

In (23), I is the number of different factors and K is the number of different components within each factor. The function $f_{i,k}(\mathbf{x}_i)$ is therefore the latent (hidden) function of the kth component within the ith factor. These hidden patterns are not observed directly, but rather must be inferred from the training dataset. The patterns represent a hidden underlying trend within the training data that characterizes the input/output relationship. In the context of emulating safety analysis codes, the patterns correspond to trends between the inputs and the code output of interest, a temperature, for example. With two factors, factor 1 could be the time factor which captures the temperature response through time and factor two could be the trend due to an uncertain input or the interaction of several uncertain inputs. These hidden patterns are not observed directly but interact together to produce the observed temperature response. As will be discussed later, constructing the FF-model requires learning these hidden patterns from the observed training data.

The difference between a factor and component is more distinguishable when (23) is rewritten in matrix form. The training output data will now be denoted as a matrix $\mathbf{Y}$ of size $M \times N$. In the GP emulator discussion, N was the number of training points. In the FF-model framework, N refers to the number of computer code case runs and M is the number of points taken per case run. If one data point is taken per case run, $M = 1$, then the number of case runs equals the number of training points. With two factors, there are two sets of training inputs, $\mathbf{x}_1$ and $\mathbf{x}_2$. The inputs do not need to be the same size. If factor 1 corresponds to the number of points taken per case run, then $\mathbf{x}_1$ is size $M \times D_1$. Factor 2 would then correspond to the number of different case runs; thus $\mathbf{x}_2$ is size $N \times D_2$. The entire set of training input values will be denoted as $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2\}$ and the entire training set will be denoted as for the GP emulator, $\mathscr{D} = \{\mathbf{X}, \mathbf{Y}\}$. With 1-component for each factor the FF-model becomes a matrix product of two vectors $\mathbf{f}_1$ and $\mathbf{f}_2$:

$$\mathbf{Y} \approx \mathbf{f}_1 \mathbf{f}_2^{\mathrm{T}}. \tag{24}$$

For more than one component, each factor is represented as a matrix. The columns within each factor's matrix correspond to the individual components within that factor. For the 2-factor 2-component FF-model the factor matrices are $\mathbf{F}_1 = [\mathbf{f}_{1,1}^{\mathrm{T}}, \mathbf{f}_{1,2}^{\mathrm{T}}]^{\mathrm{T}}$ and $\mathbf{F}_2 = [\mathbf{f}_{2,1}^{\mathrm{T}}, \mathbf{f}_{2,2}^{\mathrm{T}}]^{\mathrm{T}}$. The FF-model is then [17]

$$\mathbf{Y} \approx \mathbf{F}_1 \mathbf{F}_2^{\mathrm{T}}. \tag{25}$$

The elements within each of the factor matrices are the latent variables which represent that factor's hidden pattern

and must be learned from the training dataset. Performing Bayesian inference on the FF-model requires specification of a likelihood function between the training output data and the FF-model as well as the prior specification on each factor matrix. In general, any desired likelihood function could be used, but this work focused on a simple Gaussian likelihood with a likelihood noise σ_n^2 and mean equal to the FF-model predictive mean. The likelihood function is therefore the same as the likelihood function between the latent GP variables as the training output, just with the FF-model replacing the GP latent variable. The prior on each component within each factor is specified as a GP prior. Because the FF-model uses a GP, the emulator is known as the FFGP model. As described in detail by Schmidt, this FFGP approach is a generalization of the nonnegative matrix factorization (NMF) technique [17]. Each GP prior is assumed to be a zero-mean GP with a SE covariance function, though in general different covariance functions could be used. The GP priors on the kth component for each of the two factors are written as

$$\begin{aligned} f_{1,k}(x_1) &\sim \mathcal{GP}\left(0, k_{1,k}(x_1, x_{1'}); \phi_{1,k}\right), \\ f_{2,k}(x_2) &\sim \mathcal{GP}\left(0, k_{2,k}(x_2, x_{2'}); \phi_{2,k}\right). \end{aligned} \tag{26}$$

The semicolon notation within each of the GP priors denotes that both priors depend on the respective set of hyperparameters. Each covariance function consists of a similar set of hyperparameters as those shown in (3), namely, the signal variance and the length scales. An additional nugget hyperparameter, σ_j^2, was included to prevent ill-conditioning issues, but rather than fixing its value it was considered unknown. The hyperparameters for the (i,k)th covariance function are denoted as in (26), $\phi_{i,k} = \{\sigma_f^2, l, \sigma_j^2\}_{i,k}$. Writing the GP priors in vector notation requires applying each of the covariance functions to their respective number of input pairs. Using notation consistent with Section 3.2.1, the GP priors on the kth component for both factors are

$$\begin{aligned} \mathbf{f}_{1,k} &\sim \mathcal{GP}\left(\mathbf{0}, \mathbf{K}_{1,k}(\mathbf{x}_1, \mathbf{x}_1); \phi_{1,k}\right), \\ \mathbf{f}_{2,k} &\sim \mathcal{GP}\left(\mathbf{0}, \mathbf{K}_{2,k}(\mathbf{x}_2, \mathbf{x}_2); \phi_{2,k}\right). \end{aligned} \tag{27}$$

Comparing (27) to the GP emulator formulation immediately highlights the key differences between the two emulator types. First, the GP emulator was able to specify a prior distribution on the output data itself, as given by (7), while the FFGP emulator specifies prior distribution on the latent patterns. As described in Section 3.2.1, (7) was actually derived by integrating the GP latent variables. The FFGP latent variables cannot be integrated however, and so the FFGP model requires learning the latent variables as well as the hyperparameters as part of the training algorithm. This adds significant complexity compared to the training of the standard GP emulator. However, this added complexity may enable an important computational benefit. The standard GP emulator covariance matrix consists of the covariance function applied to every input pair in the entire training set. For the present scenario there are a total of NM training points, which means the covariance matrix is size $NM \times NM$. In the FFGP framework, each factor's covariance matrix is constructed by evaluating the factor's covariance function only at each of that particular factor's input pairs. The factor 1 covariance matrix is therefore size $M \times M$ and the factor 2 covariance matrix is size $N \times N$. By decomposing the data into various patterns, the FFGP emulator is a dimensionality reduction technique that works with multiple smaller covariance matrices.

3.3.2. Training. Training the FFGP emulator requires learning all of the latent variables and hyperparameters. For notational simplicity, the following set of expressions will assume a 2-factor 1-component FFGP model. The joint posterior for FFGP models with more components is straightforward to write out. Denoting the set of all hyperparameters as $\Xi = \{\phi_{i,k}, \sigma_n^2\}$, the joint posterior distribution (up to a normalizing constant) between all latent variables and hyperparameters for a 2-factor 1-component FFGP model is

$$\begin{aligned} &p(\mathbf{f}_1, \mathbf{f}_2, \Xi \mid \mathcal{D}) \\ &\quad \propto p\left(\mathbf{Y} \mid \mathbf{f}_1, \mathbf{f}_2, \sigma_n^2\right) p(\mathbf{f}_1 \mid \phi_1)\, p(\mathbf{f}_2 \mid \phi_2)\, p(\Xi). \end{aligned} \tag{28}$$

The log-likelihood function (up to a normalizing constant) between the training output data and the FF-model is [17]

$$\begin{aligned} \log p\left(\mathbf{Y} \mid \mathbf{f}_1, \mathbf{f}_2, \sigma_n^2\right) \propto &-\frac{1}{2\sigma_n^2}\left\|\mathbf{Y} - \mathbf{f}_1\mathbf{f}_2^{\mathrm{T}}\right\|_F^2 \\ &-\frac{NM}{2}\log\sigma_n^2. \end{aligned} \tag{29}$$

In (29) $\|\cdot\|_F^2$ denotes the Frobenius norm. The log prior for each of the factor's priors is

$$\log p(\mathbf{f}_i \mid \phi_i) \propto -\frac{1}{2}\log\left|\mathbf{K}_i\right| - \frac{1}{2}\mathbf{f}_i^{\mathrm{T}}\mathbf{K}_i^{-1}\mathbf{f}_i, \quad i = 1, 2. \tag{30}$$

The two factors are assumed to be independent *a priori* in (28). With more components, the setup is the same if all components within each factor are also assumed independent *a priori*. Any correlation between any of the components as well as across the factors is induced by the training data through the likelihood function. Drawing samples from the joint posterior with MCMC does not require any assumptions about the posterior correlation structure. Therefore any data induced posterior correlation can be completely captured by the MCMC inference procedure.

Following Schmidt in [7], the Hamiltonian Monte Carlo (HMC) MCMC scheme was used to build the FFGP emulator. The HMC is a very powerful MCMC algorithm that accounts for gradient information to suppress the randomness of a proposal. See [7, 9, 16] for detailed discussions on HMC. The HMC algorithm is ideal for situations with a very large number of highly correlated variables, as is the case with sampling the latent variables presently.

This work has several key differences from Schmidt's training algorithm in [17], to simplify the implementation and increase the execution speed. Following [20], the latent variables and hyperparameter sampling were split into a

"Gibbs-like" procedure. A single iteration of the MCMC scheme first samples the latent variables given the hyperparameters and then samples the hyperparameters given the latent variables. The latent variables were sampled with HMC, but the hyperparameters can now be sampled from a simpler MCMC algorithm such as the RWM sampler. Although less efficient compared to the HMC scheme, the RWM performed adequately for this work.

The next key difference relative to Schmidt's training algorithm was to use an empirical Bayes approach and fix the hyperparameters as point estimates, similar to the hybrid style training algorithm of Section 3.2.2. The hyperparameter point estimates are denoted as $\widehat{\Xi}$. Once the hyperparameters are fixed, the HMC algorithm is restarted, but now the hyperparameters are considered known.

The end result of the HMC algorithm is a potentially very large number of samples of all of the latent variables. One last simplification relative to Schmidt's setup was to summarize the latent variable posteriors as Gaussians. Their posterior means and covariance matrices were empirically estimated from the posterior samples. All of the latent variables are denoted in stacked vector notation as $\widehat{\mathbf{f}}$ and the empirically estimated means of the latent variables are $\mathbb{E}[\widehat{\mathbf{f}} \mid \mathscr{D}, \widehat{\Xi}]$. The empirically estimated covariance matrix of all the latent variables is $\operatorname{cov}[\widehat{\mathbf{f}} \mid \mathscr{D}, \widehat{\Xi}]$. As will be shown in the next section, this assumption greatly simplified making predictions with the FFGP emulator and ultimately provided a very useful approximation that aided the overall goal of emulator-based Bayesian model calibration.

3.3.3. Predictions. The expressions required to make predictions with the FFGP emulator were summarized briefly in [21], but they will be described in detail here. Prediction with the FFGP emulator consists of two steps: first, make a prediction in the latent factor space and then combine the factor predictions together to make a prediction on the output directly. A latent space posterior prediction is very straightforward following MVN theory and is identical in procedure to posterior predictions with the GP emulator. The joint prior between the training latent variables and test latent variables is written out similar to (12). Before writing the joint prior, the two factors are stacked together into a single stacked vector, $\widehat{\mathbf{f}} = [\mathbf{f}_1^{\mathrm{T}}, \mathbf{f}_2^{\mathrm{T}}]^{\mathrm{T}}$. Because the factors are independent *a priori*, the stacked covariance matrix is a block diagonal matrix:

$$\widehat{\mathbf{K}} = \begin{bmatrix} \mathbf{K}_1 & 0 \\ 0 & \mathbf{K}_2 \end{bmatrix}. \tag{31}$$

If more components are used, the individual factor covariance matrices are themselves block diagonal matrices. The training latent variables prior in the stacked notation is

$$\widehat{\mathbf{f}} \sim \mathcal{N}\left(\mathbf{0}, \widehat{\mathbf{K}}_{\mathbf{f}}; \widehat{\Xi}\right). \tag{32}$$

The subscript $\mathbf{f}$ is used on the stacked covariance matrix to denote that it is the stacked training covariance matrix. The test latent variables prior in stacked notation is similar to (32):

$$\widehat{\mathbf{f}}_* \sim \mathcal{N}\left(\mathbf{0}, \widehat{\mathbf{K}}_{**}; \widehat{\Xi}\right). \tag{33}$$

The subscript $**$ is used on the stacked covariance matrix in (33) to denote that it is the stacked test covariance matrix. The cross-covariance matrix between the training and test points in stacked notation is defined as $\widehat{\mathbf{K}}_{\mathbf{f},*}$, which requires evaluating the covariance function between the training and test inputs within each factor. The stacked joint prior is now easily written as

$$\begin{bmatrix} \widehat{\mathbf{f}} \\ \widehat{\mathbf{f}}_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} \widehat{\mathbf{K}}_{\mathbf{f}} & \widehat{\mathbf{K}}_{\mathbf{f},*} \\ \widehat{\mathbf{K}}_{\mathbf{f},*}^{\mathrm{T}} & \widehat{\mathbf{K}}_{**} \end{bmatrix}; \widehat{\Xi}\right). \tag{34}$$

Equation (34) is identical in format to (12), except for one key difference. The joint prior is defined between the training and test latent variables, not between the training output and the test latent variables. Conditioning on the training latent variables, the test latent variable posterior predictive (conditional) distribution is

$$\widehat{\mathbf{f}}_* \mid \widehat{\mathbf{f}}, \widehat{\Xi} \sim \mathcal{N}\left(\mathbb{E}\left[\widehat{\mathbf{f}}_* \mid \widehat{\mathbf{f}}\right], \operatorname{cov}\left(\widehat{\mathbf{f}}_* \mid \widehat{\mathbf{f}}\right)\right). \tag{35}$$

The posterior predictive (conditional) mean is

$$\mathbb{E}\left[\widehat{\mathbf{f}}_* \mid \widehat{\mathbf{f}}, \widehat{\Xi}\right] = \widehat{\mathbf{K}}_{\mathbf{f},*}^{\mathrm{T}} \widehat{\mathbf{K}}_{\mathbf{f}}^{-1} \widehat{\mathbf{f}}, \tag{36}$$

and the posterior predictive (conditional) covariance matrix is

$$\operatorname{cov}\left(\widehat{\mathbf{f}}_* \mid \widehat{\mathbf{f}}, \widehat{\Xi}\right) = \widehat{\mathbf{K}}_{**} - \widehat{\mathbf{K}}_{\mathbf{f},*}^{\mathrm{T}} \widehat{\mathbf{K}}_{\mathbf{f}}^{-1} \widehat{\mathbf{K}}_{\mathbf{f},*}. \tag{37}$$

The goal is to make a prediction conditioned on the training dataset, not on particular values of the training latent variables. Therefore the training latent variables must be integrated out using their own posterior distribution computed during the training algorithm. The resulting predictive distribution will be approximated as a Gaussian with the mean estimated using the Law of Total Expectation [7]:

$$\mathbb{E}\left[\widehat{\mathbf{f}}_* \mid \mathscr{D}, \widehat{\Xi}\right] = \int \mathbb{E}\left[\widehat{\mathbf{f}}_* \mid \widehat{\mathbf{f}}, \widehat{\Xi}\right] p\left(\widehat{\mathbf{f}} \mid \mathscr{D}, \widehat{\Xi}\right) d\widehat{\mathbf{f}}. \tag{38}$$

Substituting in (36) gives

$$\mathbb{E}\left[\widehat{\mathbf{f}}_* \mid \mathscr{D}, \widehat{\Xi}\right] = \widehat{\mathbf{K}}_{\mathbf{f},*}^{\mathrm{T}} \widehat{\mathbf{K}}_{\mathbf{f}}^{-1} \int \widehat{\mathbf{f}} p\left(\widehat{\mathbf{f}} \mid \mathscr{D}, \widehat{\Xi}\right) d\widehat{\mathbf{f}}. \tag{39}$$

The expression within the integral of (39) is simply the mean of the (stacked) training latent variables, which was empirically estimated from the posterior MCMC samples from the training algorithm. Thus, the posterior predictive test latent variable means are

$$\mathbb{E}\left[\widehat{\mathbf{f}}_* \mid \mathscr{D}, \widehat{\Xi}\right] = \widehat{\mathbf{K}}_{\mathbf{f},*}^{\mathrm{T}} \widehat{\mathbf{K}}_{\mathbf{f}}^{-1} \mathbb{E}\left[\widehat{\mathbf{f}} \mid \mathscr{D}, \widehat{\Xi}\right]. \tag{40}$$

The Law of Total Covariance is used to estimate the posterior predictive covariance of the test latent variables. In words, the

Law of Total Covariance sums up the mean of the predictive conditional covariance with the covariance of the predictive conditions means, which is given as

$$\begin{aligned}\operatorname{cov}\left(\widehat{\mathbf{f}}_* \mid \mathscr{D}, \widehat{\Xi}\right) &= \mathbb{E}\left[\operatorname{cov}\left(\widehat{\mathbf{f}}_* \mid \widehat{\mathbf{f}}, \mathscr{D}, \widehat{\Xi}\right)\right] \\ &\quad + \operatorname{cov}\left(\mathbb{E}\left[\widehat{\mathbf{f}}_* \mid \widehat{\mathbf{f}}, \mathscr{D}, \widehat{\Xi}\right]\right).\end{aligned} \tag{41}$$

Substituting in (36) and (37) as well as rearranging yields

$$\begin{aligned}&\operatorname{cov}\left(\widehat{\mathbf{f}}_* \mid \mathscr{D}, \widehat{\Xi}\right) \\ &= \widehat{\mathbf{K}}_{**} \\ &\quad - \widehat{\mathbf{K}}_{\mathbf{f},*}^{\mathrm{T}}\left(\widehat{\mathbf{K}}_{\mathbf{f}}^{-1} - \widehat{\mathbf{K}}_{\mathbf{f}}^{-1}\operatorname{cov}\left(\widehat{\mathbf{f}} \mid \mathscr{D}, \widehat{\Xi}\right)\widehat{\mathbf{K}}_{\mathbf{f}}^{-1}\right)\widehat{\mathbf{K}}_{\mathbf{f},*}^{\mathrm{T}}.\end{aligned} \tag{42}$$

Equations (40) and (42) are the approximate posterior predictive test latent variable mean and covariance matrix. They are referred to as being approximate because the training latent variable posterior distribution was approximated as a Gaussian with empirically estimated means and covariance matrix from the training algorithm.

The FF-model predictions can now be estimated. The FF-model predictive distribution is approximated as a Gaussian, with the estimated FF-model predictive means stored in an $M_* \times N_*$ matrix denoted as $\mathbf{H}_*$. M_* is the number of predictive "locations" to be made per case, and N_* is the number of cases to predict. If the FFGP model is emulating a transient, M_* is the number of predictions per case and N_* is the number of prediction cases. In general, the FFGP emulator can therefore make prediction at a large number of case runs all at once, something a computer code cannot do unless multiple instances are run simultaneously. Within the present framework of Bayesian calibration of the uncertain inputs, a single MCMC iteration requires only one case to be predicted at a time. However, the following expressions are presented for multiple case predictions at once. The following expressions change notation back to using the matrix form of the latent variables which requires splitting the stacked latent variables into their respective factors:

$$\widehat{\mathbf{f}}_* = \left[\widehat{\mathbf{f}}_{1^*}^{\mathrm{T}}, \widehat{\mathbf{f}}_{2^*}^{\mathrm{T}}\right]^{\mathrm{T}}. \tag{43}$$

Then the stacked-factor vectors are reshaped into matrices:

$$\begin{aligned}\mathbf{F}_{1^*} &= \operatorname{vec}^{-1}\left(\widehat{\mathbf{f}}_{1^*}\right), \\ \mathbf{F}_{2^*} &= \operatorname{vec}^{-1}\left(\widehat{\mathbf{f}}_{2^*}\right).\end{aligned} \tag{44}$$

Additionally, the expressions will focus on the predictive FF-model distribution at a single point rather than in vector notation. This simplifies the notation considerably.

The FF-model approximate predictive mean requires computing the expectation of the product of two latent variable factors. At the (m_*, n_*)th predictive point the FF-model approximate predictive mean is

$$\mathbb{E}\left[\mathbf{H}_*\left(m_*, n_*\right)\right] = \sum_{k=1}^{K}\mathbb{E}\left[\mathbf{F}_{1^*}\left(m_*, k\right)\mathbf{F}_{2^*}\left(n_*, k\right)\right]. \tag{45}$$

The kth component in the summation in (45) is the standard result for the product of two correlated random variables:

$$\begin{aligned}&\mathbb{E}\left[\mathbf{F}_{1^*}\left(m_*, k\right)\mathbf{F}_{2^*}\left(n_*, k\right)\right] \\ &\quad = \mathbb{E}\left[\mathbf{F}_{1^*}\left(m_*, k\right)\right]\mathbb{E}\left[\mathbf{F}_{2^*}\left(n_*, k\right)\right] \\ &\qquad + \operatorname{cov}\left(\mathbf{F}_{1^*}\left(m_*, k\right), \mathbf{F}_{2^*}\left(n_*, k\right)\right).\end{aligned} \tag{46}$$

The FF-model approximate predictive variance is the variance of the summation of products of random variables plus the FF-model likelihood noise:

$$\begin{aligned}&\operatorname{var}\left(\mathbf{H}_*\left(m_*, n_*\right)\right) \\ &\quad = \sigma_n^2 + \operatorname{var}\left(\sum_{k=1}^{K}\left\{\mathbf{F}_{1^*}\left(m_*, k\right)\mathbf{F}_{2^*}\left(n_*, k\right)\right\}\right).\end{aligned} \tag{47}$$

Writing out the expression completely gives

$$\begin{aligned}\operatorname{var}\left(\mathbf{H}_*\left(m_*, n_*\right)\right) &= \sigma_n^2 + \sum_{k=1}^{K}\operatorname{var}\left(\mathbf{F}_{1^*}\left(m_*, k\right)\mathbf{F}_{2^*}\left(n_*, k\right)\right)\cdots \\ &\quad + 2\sum_{1\le k}\sum_{<k'\le K}\operatorname{cov}\left(\mathbf{F}_{1^*}\left(m_*, k\right)\mathbf{F}_{2^*}\left(n_*, k\right), \mathbf{F}_{1^*}\left(m_*, k'\right)\mathbf{F}_{2^*}\left(n_*, k'\right)\right).\end{aligned} \tag{48}$$

Both (46) and (48) reveal the FF-model approximate prediction depends on the covariance between all components and all factors. This covariance structure of the posterior test latent variables is induced by the training dataset through the posterior training latent variable covariance structure.

3.3.4. FFGP-Based Calibration. With the FFGP emulator posterior predictive distribution approximated as a Gaussian, a modified likelihood can be formulated in much the same way as the GP emulator-modified likelihood function described in Section 3.2.4. As stated earlier, a single case run is being emulated at each MCMC iteration when calibrating the uncertain inputs; therefore $N_* = 1$. Any number of points in time (or in general any number of control variable locations, or predictions per case) can be predicted, but for notational convenience it is assumed that the number of predictions per case equals the number of observational locations, $M_* = N_o$. At each MCMC iteration the FFGP emulator predictions are therefore size $N_o \times 1$.

The joint posterior between the FFGP emulator predictions and the uncertain input parameters is

$$p\left(\mathbf{H}_*, \theta \mid \mathbf{y}_o, \mathscr{D}, \widehat{\Xi}\right) \propto p\left(\mathbf{y}_o \mid \mathbf{H}_*\right) p\left(\mathbf{H}_* \mid \left\{\mathbf{x}_{\mathrm{cv},o}, \theta\right\}, \mathscr{D}, \widehat{\Xi}\right) p\left(\theta\right). \tag{49}$$

The likelihood function between the observational data and the predictions is assumed to be Gaussian with a known observational error matrix, just as in Section 3.2.4. Integrating out the FFGP predictions gives the uncertain input posterior distribution which looks very similar to the expression in (21):

$$p\left(\theta \mid \mathbf{y}_o, \mathscr{D}, \widehat{\Xi}\right) \propto p\left(\mathbf{y}_o \mid \left\{\mathbf{x}_{\mathrm{cv},o}, \theta\right\}, \mathscr{D}, \widehat{\Xi}\right) p\left(\theta\right). \tag{50}$$

Assuming the observational error matrix Σ_ϵ is diagonal, the FFGP modified likelihood function factorizes as

$$p\left(\mathbf{y}_o \mid \left\{\mathbf{x}_{\mathrm{cv},o}, \theta\right\}, \mathscr{D}, \widehat{\Xi}\right) = \prod_{l=1}^{N_o} p\left(y_{o,l} \mid \left\{x_{\mathrm{cv},o,l}, \theta\right\}, \mathscr{D}, \widehat{\Xi}\right). \tag{51}$$

The *FFGP-modified likelihood function* for each observational data point is then

$$p\left(y_{o,l} \mid \left\{x_{\mathrm{cv},o,l}, \theta\right\}, \mathscr{D}, \widehat{\Xi}\right) \approx \prod_{l=1}^{N_o} \mathcal{N}\left(\mathbb{E}\left[\mathbf{H}_*\left(l\right)\right], \mathrm{var}\left(\mathbf{H}_*\left(l\right)\right) + \sigma_\epsilon^2\right). \tag{52}$$

4. Calibration Demonstration: Friction Factor Model

4.1. Problem Statement. A method of manufactured solutions-type approach is used to verify that the emulator-based calibration process is working as expected. The metric of success is that calibration based on the emulator replicates the calibration results if the computer code itself is used in the same MCMC procedure. The "computer code" in this context is a simple expression that would not actually require an emulator and can therefore be easily used to calibrate any of its inputs. Synthetic "observational" data are generated by setting the uncertain inputs at true values and computing the corresponding output. If the calibration process works as intended, the true values of the uncertain inputs will be learned from the synthetic observational data, within the assumed measurement error tolerance.

A simple friction factor expression is used as the computer code:

$$f = \exp\left(b\right) \mathrm{Re}^{-\exp(c)}. \tag{53}$$

Note that f in (53) is the friction factor and not related to any of the emulator latent variables. The first demonstration below assumes that only b is uncertain, while the second demonstration assumes that both b and c are uncertain. Note that the friction factor expression in (53) is written in the above form to facilitate specifying Gaussian priors on the uncertain inputs. The typical friction factor expression ($f = B/\mathrm{Re}^C$) can be recovered by substituting in $B = \exp(b)$ and $C = \exp(c)$ into (53). Gaussian priors on b and c are therefore equivalent to specifying log-normal priors on B and C. The prior means on b and c equal McAdam's friction factor correlation values: log(0.184) and log(0.2), respectively. The prior variances on each are set so that 95% of the prior probability covers ±50% around the prior mean.

Each demonstration follows the emulator-based calibration steps outlined in Figure 1.

4.2. Demonstration for the Case of One Uncertain Parameter. With only b uncertain, the friction factor expression can be decomposed into the product of two separate functions. The first is a function of the Reynolds number and the second is a function of b:

$$f = g\left(b\right) g\left(\mathrm{Re}\right), \quad g\left(b\right) = \exp\left(b\right), \quad g\left(\mathrm{Re}\right) = \mathrm{Re}^{-\exp(c)}. \tag{54}$$

The 1-component FFGP emulator should be able to exactly model this expression, within the desired noise level, because the 1-component FFGP emulator is, by assumption, the product of two functions. The control variable is the Reynolds number; therefore the two factors in the FFGP model are the Reynolds number factor (factor 1) and the uncertain input factor (factor 2). The training data was generated assuming 15 case runs, $N = 15$, and 10 control variable locations per case, $M = 10$. These numbers were chosen based on the "rule of thumb" for GP emulators that requires at least 10 training points per input [5]. The b training values were selected at 15 equally spaced points in $\pm 2\sigma$. The Re training inputs were selected at 10 equally spaced points over an assumed Reynolds number range, between 5000 and 45000. The training data is shown in blue in Figure 2 along with the synthetic observational data in red. The measurement error is assumed to be 10% of the mean value of the friction factor. The Reynolds number is shown in scaled terms where 0 and 1 correspond to the minimum and maximum training value, respectively. Figure 2 clearly shows that the true value of b falls between two of the training case runs.

Even with only one uncertain parameter, there are actually two inputs to the computer code: Re and b. If the standard GP emulator was built, a space filling design would need to be used, such as Latin Hypercube Sampling (LHS), to generate input values that sufficiently cover the input space. The FFGP training set is simpler to generate for this demonstration because each factor's training input values can be generated independent of the other factor. This illustrates how the FFGP emulator decomposes, or literally factorizes, the training set into simpler, smaller subsets.

The 2-factor 1-component FFGP emulator is built following the training algorithm outlined in Section 3.3.2. The posterior results of the observation space training points are shown in Figure 3. The red dots are the training output

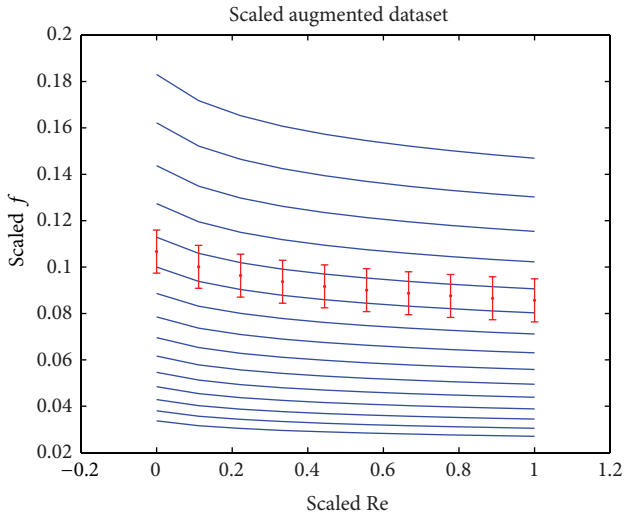

Figure 2: One uncertain parameter demonstration training set.

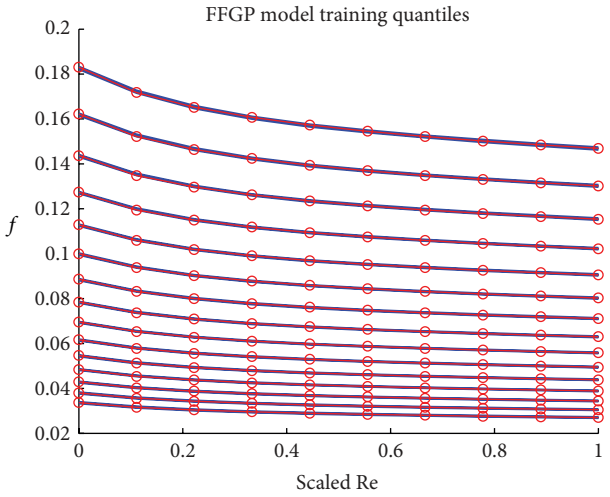

Figure 3: Posterior observational training predictions.

data and although difficult to see, the blue lines are the posterior quantiles on the FFGP training output predictions corresponding to the 5th, 25th, 50th, 75th, and 95th quantiles. The quantiles are tightly packed together representing that the FFGP emulator has very little uncertainty. This meets expectations since by assumption the 2-factor 1-component FFGP emulator should be able to exactly model the product of two functions.

The uncertain b input is calibrated using the FFGP modified likelihood function. The input is scaled between 0 and 1, which corresponds to a prior scaled mean of 0.5 and prior scaled variance of 0.25. Posterior samples were drawn using the RWM algorithm with the FFGP emulator-modified likelihood function. A total of 2×10^4 samples were drawn with the first half discarded as burn-in. Figure 4 shows the scaled posterior samples in blue with the true value displayed as the horizontal red line. The mixing rate is very high and the posterior samples are tightly packed around the true value.

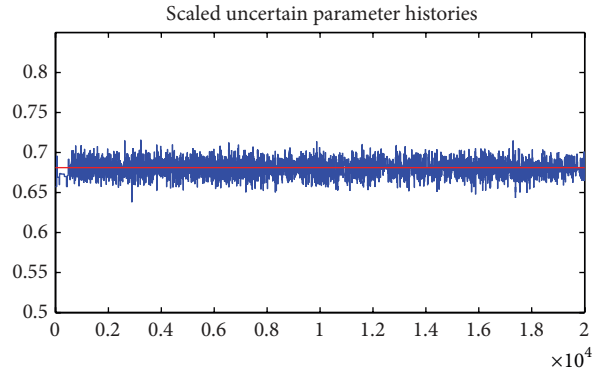

Figure 4: Scaled b posterior samples.

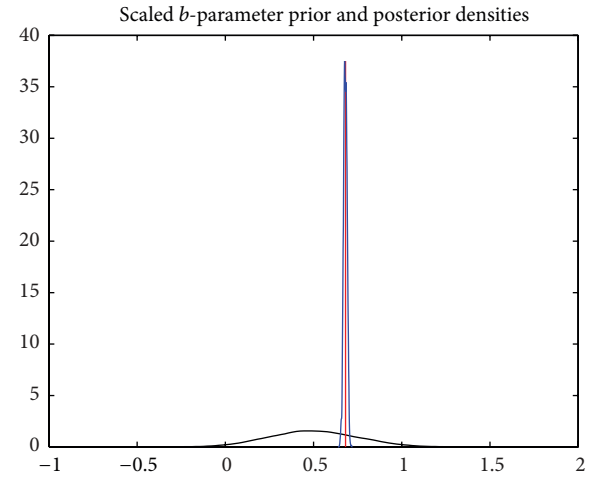

Figure 5: Estimated scaled b posterior and prior densities.

Figure 5 shows the estimated posterior distribution in blue relative to the relatively uncertain prior in black. The red line is the true value. Figure 5 illustrates how precise the posterior b distribution is, confirming that the 2-factor 1-component FFGP emulator is working as expected.

4.3. Demonstration for the Case of Two Uncertain Parameters. With both b and c uncertain, the uncertain input function $g(b, c)$ cannot be written explicitly. It is expected that the 2-factor 1-component FFGP model will no longer be able to exactly model this relationship since the friction factor is no longer a product of two simple functions. A 3-factor model could be used, but this work focused on 2-factor models for convenience. The 2-factor FFGP model requires additional components to gain the necessary flexibility to handle this. The downside of using only 2-factors requires the uncertain parameter factor (factor 2) to be trained with space filling designs, such as LHS. This work did not focus on finding the absolute "best" training set, which is an active area of research.

The LHS generated training dataset is shown in Figure 6. Fifty case runs were made with 25 points taken per case, $N = 50$ and $M = 25$. Using more training points helped

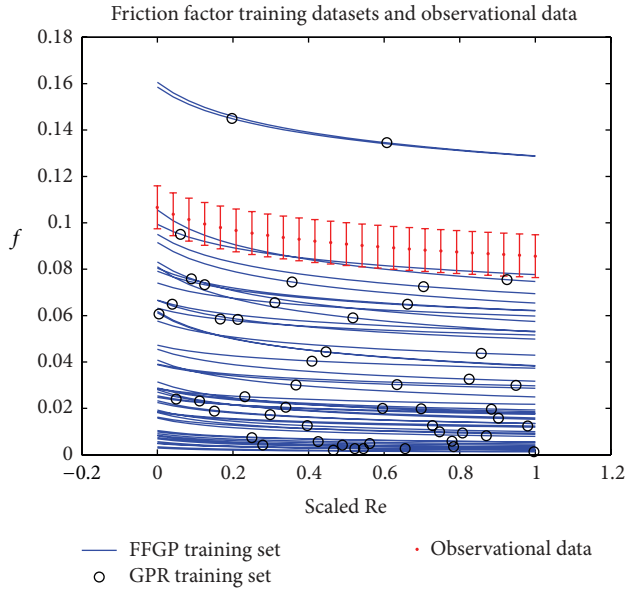

Figure 6: Two uncertain input demonstration training sets.

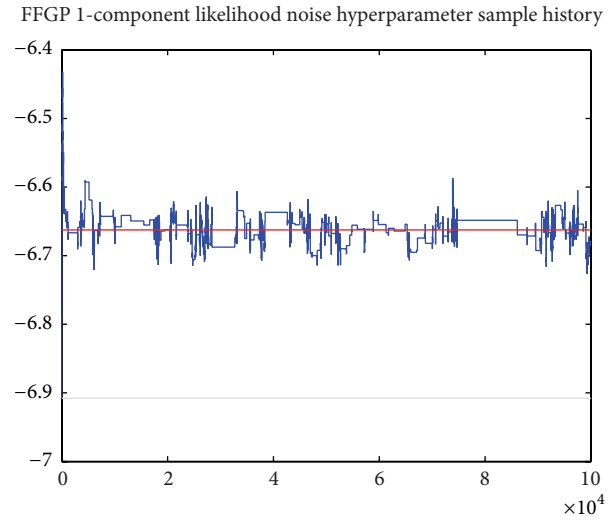

Figure 7: FFGP 2-factor 1-component likelihood noise hyperparameter samples.

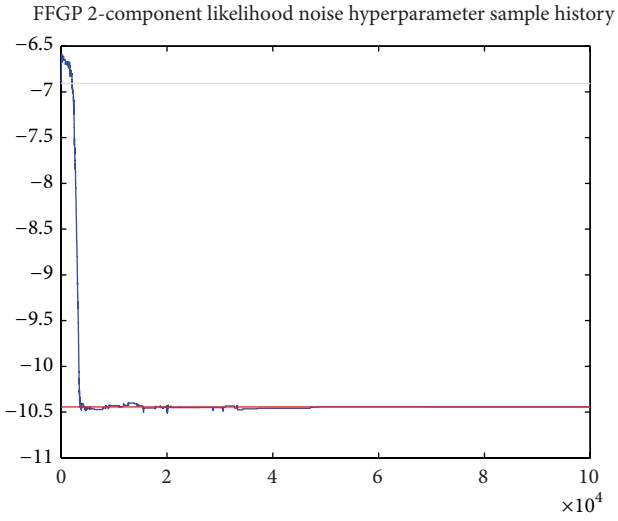

Figure 8: FFGP 2-factor 2-component likelihood noise hyperparameter samples.

guarantee the training dataset would "surround" or cover the observational data. For comparison purposes a standard GP emulator was built for this dataset. The GP emulator training points are shown as circles in Figure 6 and correspond to one point taken per case. The Reynolds numbers selected for the GP emulator training set were chosen as part of the LHS process. The FFGP emulator uses a total of $NM = 1250$ training points but the two factor covariance matrices are sizes $(M \times M) = (25 \times 25)$ for factor 1 and $(N \times N) = (50 \times 50)$ for factor 2. The GP emulator covariance matrix is size $(N \times N) = (50 \times 50)$ because only 50 points were used by assumption. If all of the training points were used, the GP emulator covariance matrix would be size $(NM \times NM) = (1250 \times 1250)$. The FFGP emulator setup can therefore drastically reduce the computational burden and facilitate using as many training points as possible.

Examining Figure 6 also shows that both the FFGP and GP emulator training sets are quite poor compared to the training set used in the one uncertain input demonstration. Only a few case runs lie within the error bars of the data and there are very few GP emulator training points near the observational data. It would be relatively easy to keep adding new training points manually for this demonstration to yield a training set that is closer to the observational data. However, in a real problem with many uncertain inputs it may be very difficult to do that manually. This demonstration problem was set up this way to show that the FFGP emulator would outperform the GP emulator due to the pattern recognition capabilities.

A total of 3 emulators were constructed, the standard GP and a 2-factor 1-component and 2-factor 2-component FFGP emulators. Due to different output scaling it is difficult to compare the training results between the GP and FFGP emulators, but a simple approach to compare FFGP performance at the end of the training algorithm is to compare the likelihood noise hyperparameter. The σ_n^2 hyperparameter was reparameterized during the RWM sampling as $\sigma_n^2 = \exp(2\phi_n)$. The more negative ϕ_n is, the smaller the likelihood noise will be for that particular emulator. Figures 7 and 8 show the sample histories for ϕ_n for the 1-component and 2-component FFGP models, respectively. In each figure, the gray line is the initial guess, the blue line shows the samples, and the red line shows the point estimate. It is very clear that the 2-component FFGP emulator is far more accurate relative to the training set. The ϕ_n point estimate for the 1-component model gives a likelihood standard deviation (σ_n) that is over 45x that of the 2-component emulator. This illustrates the point that the 1-component FFGP emulator is no longer an exact representation of the computer code. The additional component within the 2-component FFGP emulator provides

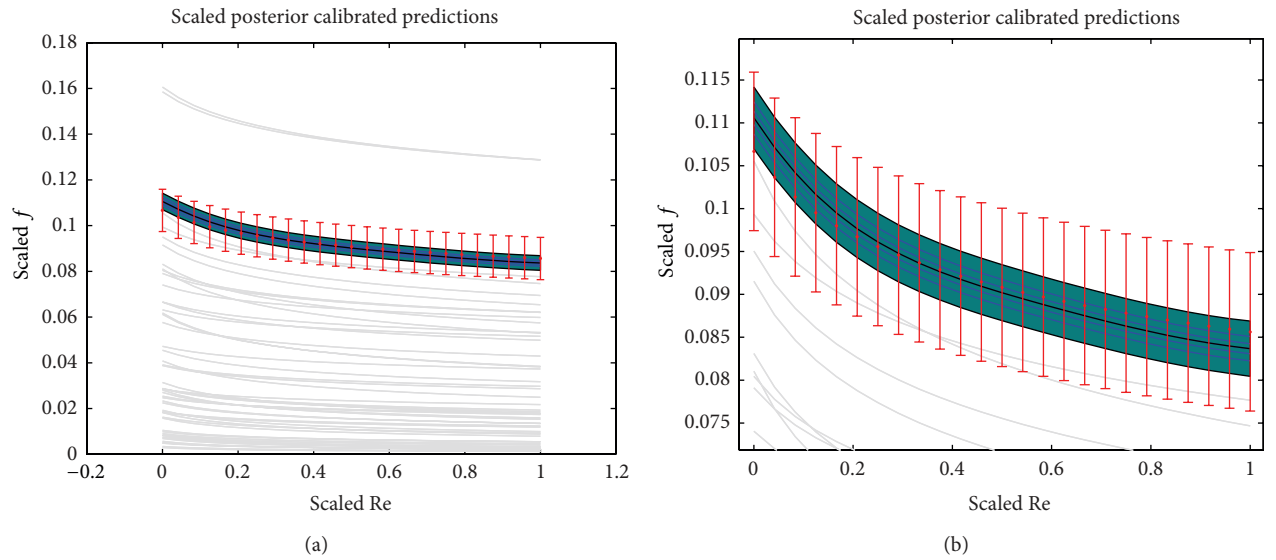

FIGURE 9: FFGP 2-factor 1-component calibrated posterior predictions ((a) covers entire training set; (b) zoom in on the observational data).

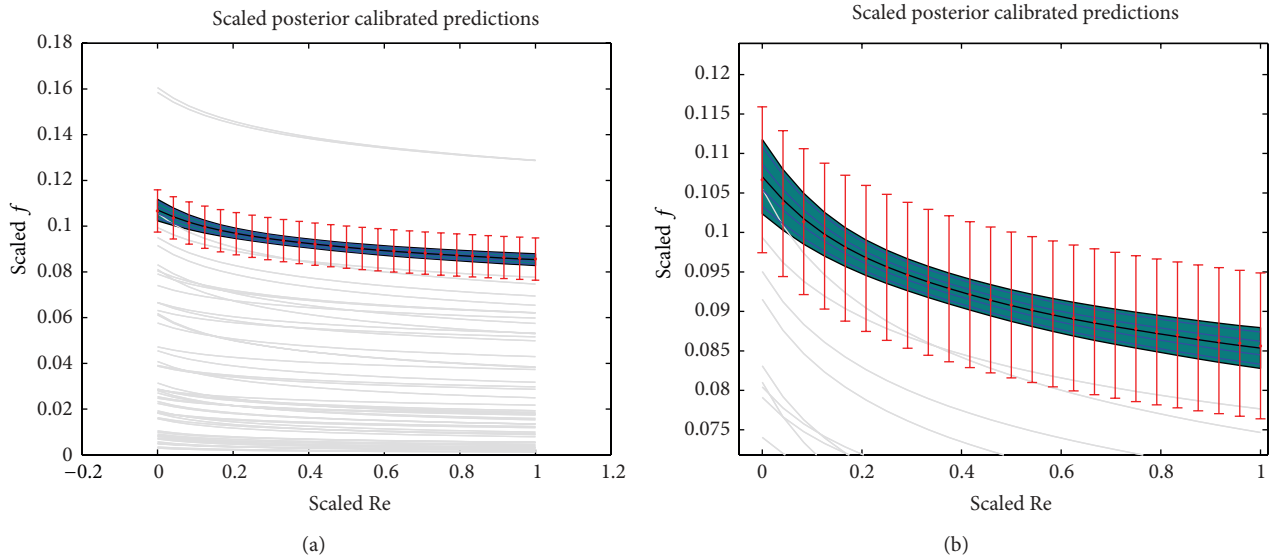

FIGURE 10: FFGP 2-factor 2-component calibrated posterior predictions ((a) covers entire training set; (b) zoom in on the observational data).

the extra flexibility needed to match the training set more accurately.

With the FFGP emulators built, they were used to calibrate the uncertain b and c inputs using the FFGP modified likelihood function within the AM-MCMC routine. A total of 10^5 samples were drawn with the first half discarded as burn-in. The calibrated posterior predictions for the 1- and 2-component FFGP emulators are shown in Figures 9 and 10, respectively. In both figures, the plot on the left shows the posterior calibrated predictions along with all of the training data. The plot on the right zooms in on the posterior calibrated predictions and the observational data. The gray lines are the training data. In both figures, the blue lines are the posterior quantiles (the 5th, 25th, 50th, 75th, and 95th quantiles) of the predictive means and although difficult to see, the black line is the mean of the predictive means. The blue lines therefore represent what the emulator thinks the computer code's posterior predictive quantiles would be if the computer code had been used. The green band is the total predictive uncertainty band of the emulator, spanning 95% of the emulator prediction probability, and is $\pm 2\sigma$ around the mean of the predictive means. Thus, the green band represents the emulator's confidence. If the edge of the green band falls directly on top of the outer blue lines, the emulator is essentially perfect and contributes no additional uncertainty to the posterior predictions. A gap between the outer blue lines and the edge of the green band, however, illustrates that the emulator has some associated uncertainty when it makes predictions. The emulator is not perfect, as described in the previous sections, and therefore

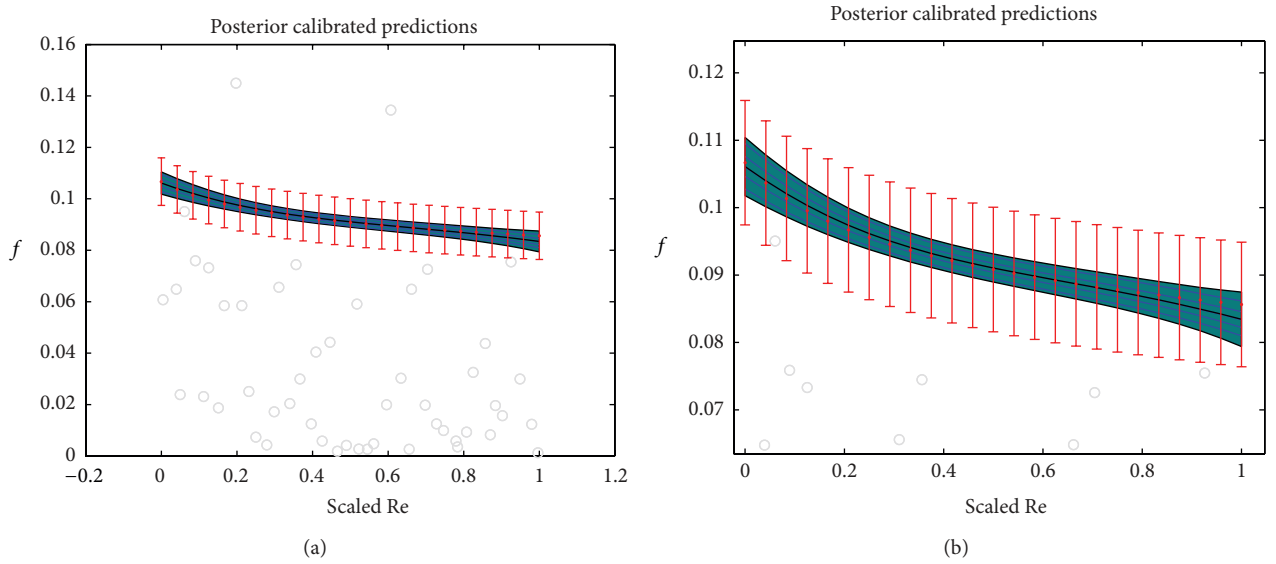

Figure 11: GP calibrated posterior predictions ((a) covers entire training set; (b) zoom in on the observational data).

some spacing between the green band's edge and the outer blue lines is expected. However, if the gap width is large, the emulator's own predictive uncertainty starts to dominate the total predictive uncertainty. Considering these conventions, the 1- and 2-component FFGP emulators can be visually compared quite easily. As shown in Figure 10 the green band is very close to the spread in the blue lines; thus the 2-component FFGP emulator adds very little additional uncertainty in the predictions. The 2-component FFGP emulator's higher posterior predictive precision relative to the 1-component FFGP emulator is in line with the training results shown in Figures 7 and 8. The 1-component FFGP emulator required more noise to match the training data, which was always propagated through onto the predictions, yielding more uncertain predictions.

Reducing the emulator predictive uncertainty allowed the 2-component FFGP emulator to be more accurate relative to the observational data. As shown in Figure 9, the 1-component FFGP emulator predictions seem to regress the observational data, within the total predictive uncertainty. The 2-component FFGP emulator's reduced total predictive uncertainty allows the data trend to be captured more accurately.

The b and c inputs were also calibrated using the GP emulator. Once constructed the GP modified likelihood function was used within the AM-MCMC scheme. The same number of samples was drawn as was done for the FFGP case, to provide a direct comparison between the GP-modified and FFGP-modified likelihood functions. The GP-based calibrated posterior predictions are shown in Figure 11 using the same format as the FFGP predictions. The GP emulator adds less uncertainty to the predictions than the 1-component FFGP emulator but is more uncertain and less accurate relative to the data than the 2-component FFGP emulator. The predictions over the first half of the (scaled) Reynolds numbers are very accurate and are similar to the 2-component FFGP emulator predictions. The latter half

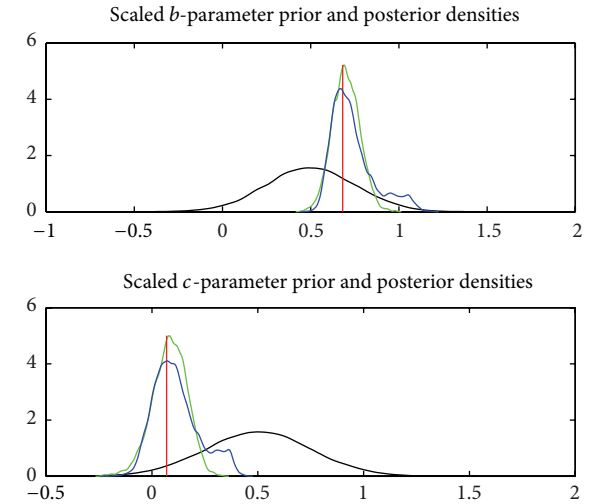

Figure 12: GP-based uncertain input posterior distributions (black: prior, blue: emulator-based posterior, green: computer code-based posterior, and red: true value).

of the Reynolds number predictions, however, are worse relative to the 2-component FFGP emulator predictions. The reasons for the difference are best explained by examining the posterior distributions on the b and c parameters.

The posterior distributions from each of the three emulator-based calibration processes are shown in Figures 12, 13, and 14. In all three figures, the black line is the estimated prior, blue is the emulator-based estimated posterior, red is the true value, and green is the estimated posterior when the computer code (the friction factor expression) is used in the AM-MCMC scheme instead of the emulators. Each of the figures is shown over the scaled input ranges, so 0.5 is the scaled prior mean. The computer code-based calibration results find the true values very well, with the posterior

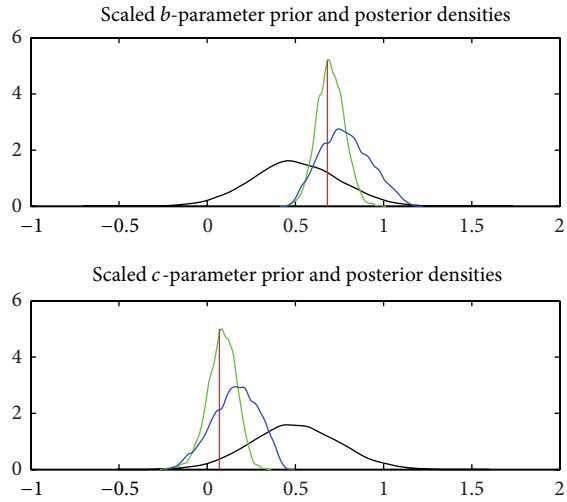

Figure 13: 1-component FFGP-based uncertain input posterior distributions (black: prior, blue: emulator-based posterior, green: computer code-based posterior, and red: true value).

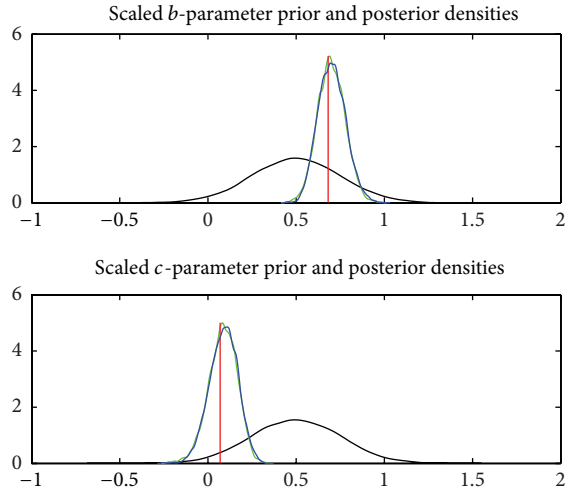

Figure 14: 2-component FFGP-based uncertain input posterior distributions (black: prior, blue: emulator-based posterior, green: computer code-based posterior, and red: true value).

mode lining up nearly exactly with the true values. The posterior variance is limited by the assumed measurement error. Although not shown, the posterior variance decreases as the assumed measurement error is decreased.

Although the GP emulator is capable of finding the correct posterior modes, the (marginal) posterior distributions do not match the computer code-based posterior distributions. Smaller second modes are present in both input parameters. As described in detail in [12], the relatively sparse GP training set is impacting the posterior results. The GP is only able to resolve the overall trend, as illustrated by the GP-based posterior mode roughly corresponding to the computer code-based posterior mode. The posterior tails however cannot be resolved since the emulator's own predictive variance starts to impact predictions far from the overall trend. The variation in the output data can therefore be explained by the additional noise from the emulator, rather than variation in either of the inputs. The inputs can therefore take on values they would not normally have, since from the emulator's point of view the prediction overlaps the data's own error. The 1-component FFGP emulator-based results also support this concept, since the posterior distributions in Figure 13 are still quite broad. The emulator is capable of shifting the (marginal) posterior distributions in the correct directions, but the additional emulator uncertainty prevents the MCMC sampling from resolving any additional information about the input values. The 2-component FFGP emulator, however, is so accurate relative to the actual friction factor "computer code," that its uncertain input (marginal) posterior distributions, as shown in Figure 14, are almost identical to the computer code-based results.

In more complex problems, it is not expected that the FFGP-based results will always be as accurate as in this simple demonstration. However, the FFGP emulator-based calibration is capable of matching the computer code-based calibration results as shown here. In more complex, and realistic situations, the computer code-based results will not be available for comparison, so it was important to verify through this method of manufactured solution problem that the emulator-based process works as expected.

5. Conclusions

The emulator-based calibration approach with the FFGP model was shown above to be capable of reproducing the calibration results obtained when the actual computer code is used in the MCMC sampling. As explored in [11, 17], the efficacy of the FFGP in this application can depend on how the model is structured, but, in cases explored, the additional FFGP emulator was shown to outperform the standard GP emulator, on the given friction factor demonstration problem, because it is capable of efficiently using more training data. This is an important feature because safety analysis problems produce time series predictions which could prove to be computationally expensive for standard GP emulators. Reducing the computational burden would require choosing a limited subset of all of the training runs, which might negatively impact the GP emulator-based results as described in [12]. The friction factor calibration GP-based results presented in this work confirmed those issues. The FFGP emulator however uses pattern recognition techniques to efficiently decompose the training data. The latent or hidden patterns allow more training data to be used which can drastically improve the predictive accuracy of the emulator.

This paper specifically covered the theory and formulation of the FFGP-based calibration approach. Work to presented in a subsequent paper, applies the FFGP-based calibration approach to a more realistic safety analysis scenario, an EBR-II loss of flow transient modeled with RELAP5. As will be shown in that paper, the FFGP-based calibration approach is over 600 times faster than if the RELAP5 model was used directly. Moreover, the FFGP approach is needed, because the standard GP emulator does not provide the

necessary flexibility to emulate the RELAP5 time series predictions.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] D. L. Kelly and C. L. Smith, "Bayesian inference in probabilistic risk assessment: the current state of the art," *Reliability Engineering and System Safety*, vol. 94, no. 2, pp. 628–643, 2009.

[2] D. Higdon, M. Kennedy, J. C. Cavendish, J. A. Cafeo, and R. D. Ryne, "Combining field data and computer simulations for calibration and prediction," *SIAM Journal on Scientific Computing*, vol. 26, no. 2, pp. 448–466, 2004.

[3] D. Higdon, J. Gattiker, B. Williams, and M. Rightley, "Computer model calibration using high-dimensional output," *Journal of the American Statistical Association*, vol. 103, no. 482, pp. 570–583, 2008.

[4] A. Keane and P. Nair, *Computational Approaches for Aerospace Design: The Pursuit of Excellence*, John Wiley & Sons, 2005.

[5] MUCM, Managing Uncertainty in Complex Models Project, http://www.mucm.ac.uk.

[6] C. E. Rasmussen and C. Williams, *Gaussian Processes in Machine Learning*, Springer, New York, NY, USA, 2004.

[7] K. Murphy, *Machine Learning: A Probabilistic Perspective*, MIT Press, Cambridge, Mass, USA, 2012.

[8] C. Linkletter, D. Bingham, N. Hengartner, D. Higdon, and K. Q. Ye, "Variable selection for Gaussian process models in computer experiments," *Technometrics*, vol. 48, no. 4, pp. 478–490, 2006.

[9] N. Zuber, G. E. Wilson, B. E. Boyack (LANL) et al., "Quantifying reactor safety margins Part 5: evaluation of scale-up capabilities of best estimate codes," *Nuclear Engineering and Design*, vol. 119, no. 1, pp. 97–107, 1990.

[10] N. Zuber, U. S. Rohatgi, W. Wulff, and I. Catton, "Application of fractional scaling analysis (FSA) to loss of coolant accidents (LOCA): methodology development," *Nuclear Engineering and Design*, vol. 237, no. 15–17, pp. 1593–1607, 2007.

[11] J. Yurko, *Uncertainty quantification in safety codes using a Bayesian approach with data from separate and integral effect tests [Ph.D. thesis]*, MIT, Cambridge, Mass, USA, 2014.

[12] F. M. Hemez and S. Atamturktur, "The dangers of sparse sampling for the quantification of margin and uncertainty," *Reliability Engineering and System Safety*, vol. 96, no. 9, pp. 1220–1231, 2011.

[13] J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn, "Design and analysis of computer experiments," *Statistical Science*, vol. 4, no. 4, pp. 409–435, 1989.

[14] W. J. Welch, R. J. Buck, J. Sacks, H. P. Wynn, T. J. Mitchell, and M. D. Morris, "Screening, predicting, and computer experiments," *Technometrics*, vol. 34, no. 1, pp. 15–25, 1992.

[15] N. A. Cressie, *Statistics for Spatial Data*, John Wiley & Sons, New York, NY, USA, 1993.

[16] M. C. Kennedy and A. O'Hagan, "Bayesian calibration of computer models," *Journal of the Royal Statistical Society. Series B. Statistical Methodology*, vol. 63, no. 3, pp. 425–464, 2001.

[17] M. N. Schmidt, "Function factorization using warped Gaussian processes," in *Proceedings of the 26th International Conference on Machine Learning*, pp. 921–928, New York, NY, USA, June 2009.

[18] J. Luttinen and A. Ihler, "Variation Gaussian process factor analysis for modeling Spatio-temporal data," in *Advances in Neural Information Processing Systems*, vol. 22, pp. 1177–1185, 2009.

[19] H. Haario, E. Saksman, and J. Tamminen, "An adaptive metropolis algorithm," *Bernoulli*, vol. 7, no. 2, pp. 223–242, 2001.

[20] R. Neal, "MCMC using Hamiltonian dynamics," in *Handbook of Markov Chain Monte Carlo*, pp. 113–162, Chapman & Hall, CRC Press, 2011.

[21] J. Yurko, J. Buongiorno, and R. Youngblood, "Bayesian calibration of safety codes using data from separate and integral effects tests," in *Proceedings of the International Topical Meeting on Probabilistic Safety Assessment and Analysis*, Sun Valley, Idaho, USA, 2015.