

Calibration under Uncertainty

Laura P. Swiler
Timothy G. Trucano
Optimization and Uncertainty Estimation Department
Sandia National Laboratories
PO Box 5800
Albuquerque, NM 87185-0370

Abstract

This report is a white paper summarizing the literature and different approaches to the problem of calibrating computer model parameters in the face of model uncertainty. Model calibration is often formulated as finding the parameters that minimize the squared difference between the model-computed data (the predicted data) and the actual experimental data. This approach does not allow for explicit treatment of uncertainty or error in the model itself: the model is considered the “true” deterministic representation of reality. While this approach does have utility, it is far from an accurate mathematical treatment of the true model calibration problem in which both the computed data and experimental data have error bars. This year, we examined methods to perform calibration accounting for the error in both the computer model and the data, as well as improving our understanding of its meaning for model predictability. We call this approach Calibration under Uncertainty (CUU). This talk presents our current thinking on CUU. We outline some current approaches in the literature, and discuss the Bayesian approach to CUU in detail.

Acknowledgments

We thank the Department of Energy's MICS program (Mathematics, Information, and Computational Sciences) for initial funding of this research. We also thank Bruce Hendrickson, Scott Mitchell, and Marty Pilch for their ongoing support of this research area.

Contents

Figures	iv
Introduction	5
Definitions	6
CALIBRATION	7
CALIBRATION UNDER UNCERTAINTY	7
Classical Parameter Estimation Methods	9
LINEAR LEAST SQUARES REGRESSION	10
NONLINEAR LEAST SQUARES REGRESSION	12
<i>Optimization</i>	13
Background	14
BAYESIAN ANALYSIS	14
<i>Discrete case</i>	14
<i>Continuous case</i>	15
<i>Hypothesis Testing</i>	15
<i>Controversy with Bayesian Inference</i>	16
<i>Examples</i>	17
<i>Conjugate pairs</i>	18
Sampling Distribution	19
Conjugate Prior Distribution	19
CALCULATIONS FROM MARKOV CHAIN MONTE CARLO	19
<i>Metropolis-Hasting algorithm</i>	20
<i>MCMC Convergence</i>	21
<i>Gibbs Sampling</i>	21
<i>Gibbs Sampling Algorithm</i>	22
SOFTWARE	22
EXAMPLE 1: BINOMIAL MODEL	24
GAUSSIAN PROCESSES	32
<i>Rosenbrock Example of Gaussian Processes</i>	34
Conclusions	44
List of References	46
BACKGROUND PAPERS ON BAYESIAN CALIBRATION IDEAS	46
WISH LIST	46
FROM THE SISC – UNCERTAINTY QUANTIFICATION PAPERS (MANUSCRIPT ONLY)	48
V&V EFFORTS	48
GAUSSIAN PROCESSES	49
BAYESIAN ANALYSIS/MCMC	49
MISCELLANEOUS PAPERS	50
BOOKS	50
WEB SITES	50
APPENDIX: Annotated Bibliography	52
K. CAMPBELL (2002), “A BRIEF SURVEY OF STATISTICAL MODEL CALIBRATION IDEAS.”	52
M. C. KENNEDY AND A. O’HAGAN “BAYESIAN CALIBRATION OF COMPUTER MODELS.”	54
D. D. COX, J-S. PARK, AND C. E. SINGER. “A STATISTICAL METHOD FOR TUNING A COMPUTER CODE TO A DATA BASE.”	59
THE DESIGN AND ANALYSIS OF COMPUTER EXPERIMENTS, BY SANTNER, WILLIAMS, NOTZ.	63
<i>Non Bayesian Approach</i>	63
<i>Bayesian Approach to Design of Experiments</i>	64

Figures and Tables

Figure 1. Calibration based on experimental data	7
Figure 2. Calibration with experimental data and uncertain model data	8
Figure 3. Confidence versus prediction intervals in linear regression.	11
Figure 4. Confidence versus prediction intervals in nonlinear regression	13
Table 1. Conjugate priors associated with various sampling distributions	19
Figure 5. Rosenbrock's Function	23
Figure 6. Beta Prior for Binomial Failure Example	25
Figure 7. Bayesian Updating of the Beta Distribution for p , the Failure Probability	25
Figure 8. Posterior Distribution Summary	26
Figure 9. Binomial Failure Example with $Be(1,9)$ prior distribution on p	27
Figure 10. Comparison of YADAS and BUGS output	32
Figure 11. Gaussian Process for Rosenbrock Function based on 11 input points	37
Figure 12. Gaussian Process for Rosenbrock Function based on 110 input points	37
Figure 13. Delta for Configuration 1, Experiment 1	41
Figure 14. Gaussian process (with a zero mean prior) on delta	42
Table 2. Parameters sampled for model runs	45
Figure 16. Delta values as a function of model configuration and experiment	46
Figure 17. Mean delta vs. time (in 1000s seconds on X-axis), with GP prediction of delta	47
Figure 18. Samples of the Model Discrepancy term for Configuration 5, with mean discrepancy shown in red	48

Introduction

A fundamental premise of many large DOE programs is that computer models can be calibrated using experimental data, and then used to predict phenomena for which experimental data are unavailable due to factors such as test facility restrictions, cost limitations, or treaty obligations. For example, Sandia uses various computer models calibrated with limited experimental data in the Stockpile Stewardship Program. Predictions from these models serve as a basis for decisions such as weapon refurbishment schedules, experimental facility planning, and national policy recommendations.

The classical approach to computer model calibration assumes that the computer model is exact and the experimental data have error bars (i.e., confidence intervals). Calibration then occurs when the computer model is tuned so that the computed data lie as close as possible, as in a least-squares sense, to the experimental data. While this approach does have utility, it is far from an accurate mathematical treatment of the true model calibration problem in which both the computed data and experimental data have error bars. In such a scenario, calibration occurs when, in layman's terms, the computer model is tuned to yield as much of an overlap as possible between the experimental and computational error bars. We refer to this type of calibration as "Calibration under Uncertainty" or CUU.

This year, we have focused on methods to perform this more accurate computer model calibration, accounting for the error in the computer model as well as in the data when making predictions, as well as better understanding its meaning for model predictability. This is an exploratory research topic. In this paper, we define calibration vs. calibration under uncertainty, review classical parameter estimation methods, present background material used in the current methods of CUU, and give a technical review of these methods, with an emphasis on Bayesian analysis techniques. We present a case study of CUU, and we conclude with some research questions and areas for further research.

The main CUU approach we examine is that of Kennedy and O'Hagan (2001), hereafter referred to as KOH. They formulate a model for calibration data that includes an experimental error term (similar to standard regression) and a model discrepancy term, with a Gaussian process chosen to model the discrepancy. They then use a Bayesian approach to update the statistical parameters associated with the discrepancy term and with the model parameters. The purpose of updating is generally to reduce uncertainty in the parameters through the application of additional information. Reduced uncertainty increases the predictive content of the calibration, or that is the expectation. We discuss several issues relating to the Bayesian approach: First, is the Gaussian prior the correct one, or the most effective choice for complex computational science and engineering (CSE) models? Second, the mathematics behind this approach involves covariance matrices of joint input variable distributions. Estimating the full joint posterior distribution therefore requires complicated integration and so techniques like Markov Chain Monte Carlo (MCMC) sampling are used to approximate the posterior distributions on the hyperparameters which

govern the prior distribution. We discuss MCMC methods and their suitability to the CUU problem. In our conclusion section, we discuss model prediction and the limitations of inferences under various frameworks.

Definitions

First, we define some general terms such as model, prediction, verification, and validation. For more discussion on these terms, the reader is referred to [Calibration contra Validation SAMO paper]. Then, we define the terms *calibration* and *calibration under uncertainty* in more detail.

We define a **model** to be a particular choice of inputs (e.g., initial, boundary and numerical parameters) for a particular computer code (e.g., a finite element code) that produce a specific calculation. The results of the model are the resulting numerical data. A computational **prediction** is simply a calculation that predicts a number or quantity or a collection of these quantities prior to their physical measurement. We are interested in **accurate prediction** using computation. Intuitively, accurate prediction means having confidence or belief in the prediction, and being willing to use the prediction in some meaningful way, especially in a decision process. By introducing the concept of accuracy into this discussion we have introduced the requirement for one or more measurement principles that we can use to quantify this accuracy. We call these measurement principles **benchmarks**. A **benchmark** is a choice of information that is believed to be accurate or true, one or more methods of comparing this information with computational results, and logical procedures for drawing conclusions from these comparisons. In particular, we are interested in benchmarks that are to be used for calibration, verification and validation of codes or computational models.

Verification is the process of determining that requirements for the intended application are correctly implemented in the code. Roache has stated that this effectively means that the equations implemented in the code are correctly solved for the intended application. The problem of verification in computational science and engineering (CS&E) codes boils down to the following. Given a set of equations, (1) are the chosen solution algorithms correct?; (2) is the software implementing these algorithms correct?; (3) are particular choices of input parameters and meshing providing accurate solutions? **Validation** deals with the question of whether the implemented equations in a code are correct for the intended applications. We define **validation** to be the process of quantifying the physical accuracy of a code for particular predictive applications through the comparison with defined sets of physical benchmarks. These benchmarks define what we will call the **validation domain**. We assume in this paper that validation benchmarks are always experimental. Finally, **calibration** is the process of improving the agreement of a code calculation or set of code calculations with respect to a chosen set of benchmarks through the adjustment of parameters implemented in the code. There may be a different set of benchmarks chosen for calibration vs. validation. We assume that the calibration benchmarks are also experimental.

Calibration

Model calibration refers to adjusting the parameters of a computer model (such as a simulation model or a finite element model). The adjustment of parameters is usually done with some objective in mind, such as to generate “better” predictions with the model or to update the parameters according to additional data or greater physical understanding of the phenomena of interest. There are many methods for adjusting the parameters. The most common approach to model calibration is to find the parameters which minimize the squared difference between the model computed data (the predicted data) and the actual experimental data. This approach does not allow for explicit treatment of uncertainty or error in the model itself: the model is considered the “true” deterministic representation of reality. Figure 1 shows a graph of a least squares fit where the coefficients of the model (in this case depicted by the pink line) are obtained by a standard regression technique. In this case, the calibrated parameters are the regression coefficients.

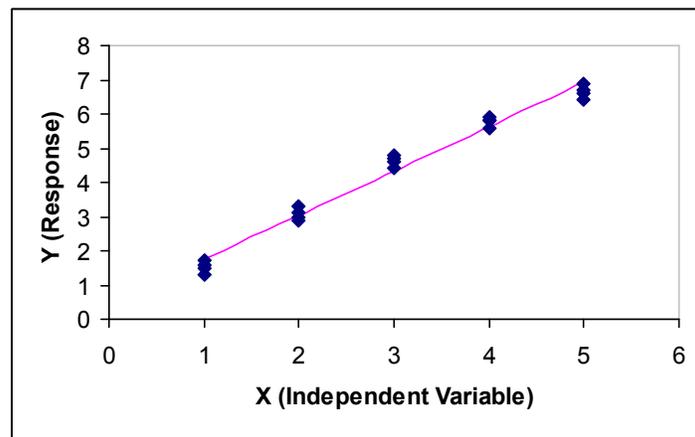


Figure 1. Calibration based on experimental data

Calibration under Uncertainty

CUU refers to adjusting parameters to reflect uncertainties both in the experimental data and in the computer model. Figure 2 depicts the situation of interest. The blue bars in Figure 2 represent the bounds on experimental data, and the yellow bars represent the bounds on results obtained by various model uncertainties. We want to determine the “optimal” parameter settings which will maximize the overlap between the blue and yellow bars.

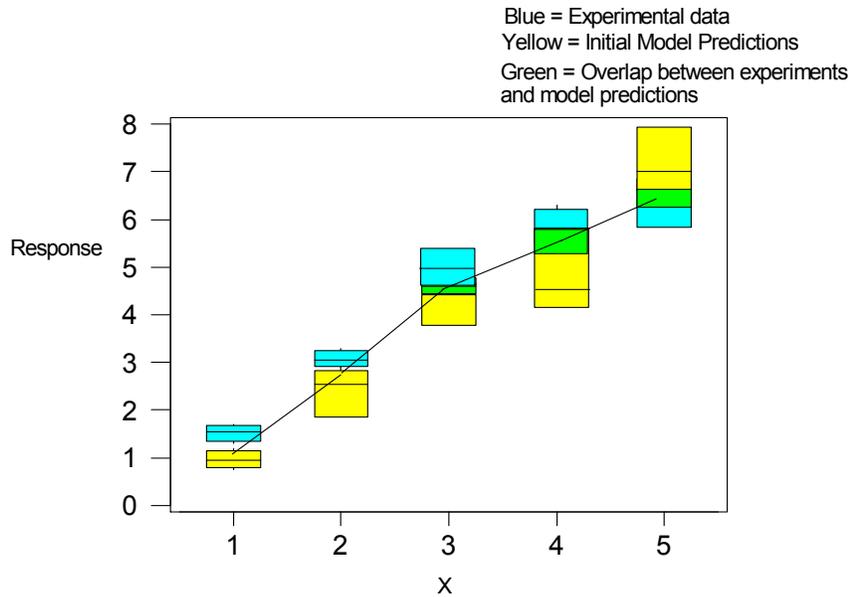


Figure 2. Calibration with experimental data and uncertain model data

The problem is made harder by the fact that models have in fact several different kinds of uncertainty. For our purposes here, we distinguish three different categories, all of which enter into the mathematics of calibration and the inference that can be achieved regarding the associated predictability.

- **Numerical:** This can be thought of as numerical errors associated with discretizations (we are primarily focused on models that solve systems of differential equations in our work). It can also include the difficult issue of undiscovered software “bugs” and other shortcomings associated with the numerical solution implementation of the model. Attacking this problem is classically associated with *verification*. Hence, success in verification reduces numerical uncertainty in our perspective, and potentially in quantified ways that can be used in CUU.
- **Parametric:** This is uncertainty associated with parameterizations in the model, which could include numerical “tunings” as well as specification of the initial and boundary conditions. Worrying about parametric uncertainty and how to reduce it as a calibration problem is part of this project.
- **Structural:** Structural uncertainty has to do with whether the model is “correct” in some sense, typically “correct for the application.” This is classically a problem of *validation*. Hence, success in validation reduces structural uncertainty in our perspective, and potentially in quantified ways that can be used in CUU.

Classical calibration addresses parametric uncertainty in a restricted sense, as in regression, but ignores numerical uncertainty and doesn’t even philosophically address the presence of structural uncertainty. Ideally, we would like to incorporate the three

kinds of uncertainty into model calibration. In later sections of this report, we discuss what types of uncertainty are treated in various approaches.

The classical parameter estimation methods, linear and nonlinear regression, are discussed below. There are two reasons for this:

1. These methods have been used successfully for many years to calibrate models.
2. Preliminary examination of the Bayesian framework has shown us that the “model discrepancy” or model error term often varies in a systematic way with respect to the model inputs or parameters. Thus, if we are trying to characterize the model discrepancy, we might want to use a regression model as a basis function in a Gaussian process describing the model error.

Classical Parameter Estimation Methods

In classical parameter estimation, the approach taken is to use a statistical process model to model the underlying process. In such an approach, the “optimal” calibrated parameters are determined by a regression technique. Note that the optimal parameters refer to parameters of a statistical model (e.g., a regression equation), and not the parameters of a simulation model. The statistical process model approach is based on the idea that experimental data about the process are available. While linear regression may not often capture the complex phenomena in CS&E models, we list the assumptions underlying linear regression here as a starting point for understanding the predictive capability one can obtain using linear regression. The underlying assumptions used in statistical process modeling are [Neter et al.; NIST]:

1. **The underlying process has random variation and is not deterministic.**
2. **The mean of the random errors are zero.**
3. **The random errors have constant standard deviation/variance.**
4. **The random errors follow a normal distribution.**
5. **The data must be sampled randomly from the underlying process.**
6. **The explanatory variables are observed without error.**

One of the most important assumptions in prediction is assumption #4, that the random errors follow a normal distribution. The mathematical theory for inferences using the normal distribution assumption of error terms is well developed. In practice, the normal distribution often describes the actual distribution of random errors reasonably well. There are a variety of statistical tests to check for normality of errors. If this assumption is violated, then the inferences made about the process may be incorrect.

Linear Least Squares Regression

The most widely-used method to estimate parameters in a model is to use a linear least squares regression. In a regression model with one dependent variable y and multiple independent variables $x_j, j = 1 \dots k$, the linear model is formulated as:

$$y = \beta_0 + \beta_1 x_1 + \dots + \beta_k x_k + \varepsilon$$

where ε represents random error associated with the model (note: some textbooks emphasize that the error is observational error; others state that all unexplained variation in y caused by important but unexplained variables or by unexplainable random phenomena is included in the random error term). Usually $\varepsilon \sim N(0, \sigma^2)$.

A particular observation is indexed by i : $y_i = \beta_0 + \beta_1 x_{1i} + \dots + \beta_k x_{ki} + \varepsilon_i$, and the expected value of the output at a particular input value is: $E(y_i) = \beta_0 + \beta_1 x_{1i} + \dots + \beta_k x_{ki}$

Least squares regression minimizes the sum of squares of the deviations of the y -values about their predicted values for all the data points. Thus, for n data points, the Sum of Squares of the Errors (SSE) is:

$$SSE = \sum_{i=1}^n [y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_{1i} + \dots + \hat{\beta}_k x_{ki})]^2$$

The quantities $\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_k$ that minimize the SSE are called the least squares estimates of the parameters $\beta_0, \beta_1, \dots, \beta_k$ and the prediction equation is: $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \dots + \hat{\beta}_k x_k$, where the “hat” notation can be read as “estimator of.” Thus, \hat{y} is the least squares estimator of $E(y)$, and $\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_k$ are estimators of the parameters $\beta_0, \beta_1, \dots, \beta_k$.

Note that the term “linear regression” means linear in the parameters. It is used for any function in which each explanatory or independent variable in the function is multiplied by an unknown parameter, there is at most one unknown parameter with no corresponding explanatory variable (the intercept term), and all of the individual terms are summed to produce the final function value. In statistical terms, any function that meets these criteria would be called a “linear function,” even though the function may not be a straight line. For example, an explanatory variable could be x^2 .

The values of the model parameters $\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_k$ which minimize the SSE are obtained by setting the partial derivatives $\partial SSE / \partial \beta_j = 0$. The solution is a system (usually over-determined) of linear equations that is solved for the values of the unknown parameters. The least squares estimates of the coefficients are: $\hat{\beta} = (X^T X)^{-1} X^T Y$

Regression models are used for prediction in a variety of ways. Some of the most common prediction tasks are to:

1. Predict a response at a particular set of input variables. The input variables are also called predictor, independent, or explanatory variables. This is done by substituting a particular value of x , say x' , into the regression formula with the fitted coefficients: $\hat{y}' = \hat{\beta}_0 + \hat{\beta}_1 x_1' + \dots + \hat{\beta}_k x_k'$. This is interpreted as: the mean or expected response at input x' is given by \hat{y}' .
2. Describe the confidence interval (CI) for the mean response at a particular set of input values (given in step 1). The confidence interval range (e.g., a 95% CI) is the range in which the estimated mean response for a set of predictor values is

expected to fall. This interval is defined by lower and upper limits, calculated from the confidence level and the standard error of the fits. The CI is around the \hat{y}' , and is interpreted as: the mean or expected response at input x' falls within the CI at a given confidence level.

- Describe the prediction interval (PI) at a particular set of input values. The prediction interval is the range in which the predicted response for a new observation is expected to fall. The PI differs from the CI in that the prediction interval is an interval in which a particular response for input x' is expected to fall, and the CI is the interval in which the mean response for input x' is expected to fall. The PI is defined by lower and upper limits, which are calculated from the confidence level and the standard error of the prediction. The prediction interval is always wider than the confidence interval because of the added uncertainty involved in predicting a single response versus the mean response. Note that the prediction interval formula has an extra term in it, namely 1. So regardless of sample size, the added term will always cause the prediction confidence interval to be a little bigger than the confidence interval.

The formulas used to calculate the CI and the PI are found in most statistical text [e.g., Neter et. al].

An example of confidence vs. prediction intervals for a linear regression model is shown in Figure 3. Note that these intervals are for individual input points. If you want a CI or PI for the entire function, it is necessary to use a simultaneous method such as Scheffé or Bonferroni confidence intervals.

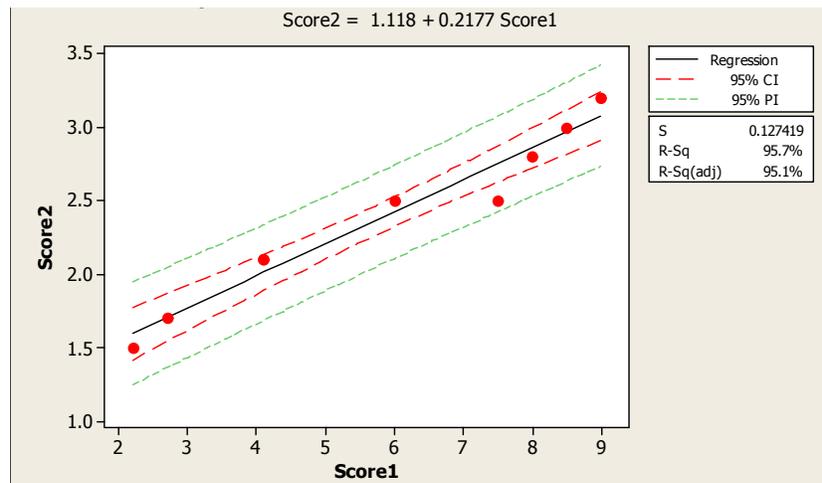


Figure 3. Confidence versus prediction intervals in linear regression.

Linear regression is used because so extensively because many processes are well-described by linear models, or at least well-approximated by a linear model over a short range. The theory associated with linear regression is well-understood and allows for construction of different types of easily-interpretable statistical intervals for predictions,

calibrations, and optimizations. These statistical intervals can then be used to give clear answers to scientific and engineering questions.

The main disadvantages of linear least squares are limitations in the shapes that linear models can assume over long ranges, and sensitivity to outliers. Linear models with nonlinear terms in the independent variables curve relatively slowly, so for inherently nonlinear processes it becomes increasingly difficult to find a linear model that fits the data well as the range of the data increases. Additionally, the method of least squares is very sensitive to the presence of unusual data points in the data used to fit a model. One or two outliers can sometimes seriously skew the results of a least squares analysis.

Nonlinear Least Squares Regression

Nonlinear least squares regression extends linear least squares regression for use with a much larger and more general class of functions. Almost any function that can be written in closed form can be incorporated in a nonlinear regression model. Unlike linear regression, there are very few limitations on the way parameters can be used in the functional part of a nonlinear regression model. The way in which the unknown parameters in the function are estimated, however, is conceptually the same as it is in linear least squares regression.

In nonlinear regression, the nonlinear model between the response y and the predictor x is given as: $y = f(x, \theta) + e$, where e is the random error term and θ may be a vector. As an example, one could have $y_i = \theta_1[1 - \exp(-x_i/\theta_2)] + e_i$. The goal of nonlinear regression is to find the optimal values of θ to minimize the function: $\sum_{i=1}^n (y_i - f(x_i, \theta))^2$

The value of θ that minimizes the sum of squares is $\hat{\theta}$, and has an estimated covariance matrix given by: $Var(\hat{\theta}) = s^2(W'W)^{-1}$, where W is an $n \times p$ matrix of first derivatives evaluated at $\hat{\theta}$, and $s^2 = RSS/(n-p) = \text{estimator of } \sigma^2$. In terms of prediction, equations similar to linear regression are used. If one is predicting the mean response at a particular value of x_0 , the confidence interval is given by $\hat{y}_0 \pm t_{n-p} s \sqrt{\omega_0'(W'W)^{-1}\omega_0}$, where ω_0 is the vector of first derivatives of f evaluated at the parameter estimates and x_0 . The prediction interval adds the factor of 1: $\hat{y}_0 \pm t_{n-p} s \sqrt{1 + \omega_0'(W'W)^{-1}\omega_0}$. An example of nonlinear regression is shown in Figure 4.

POLYNOMIAL MODEL FOR DECAY DATA

$$\begin{aligned} \text{Count}(y) = & 566.815 - 31.5638 \text{ Time}(t) \\ & + 0.782551 \text{ Time}(t)^2 - 0.0065666 \text{ Time}(t)^3 \\ S = & 47.1669 \quad R\text{-Sq} = 92.8 \% \quad R\text{-Sq}(\text{adj}) = 91.0 \% \end{aligned}$$

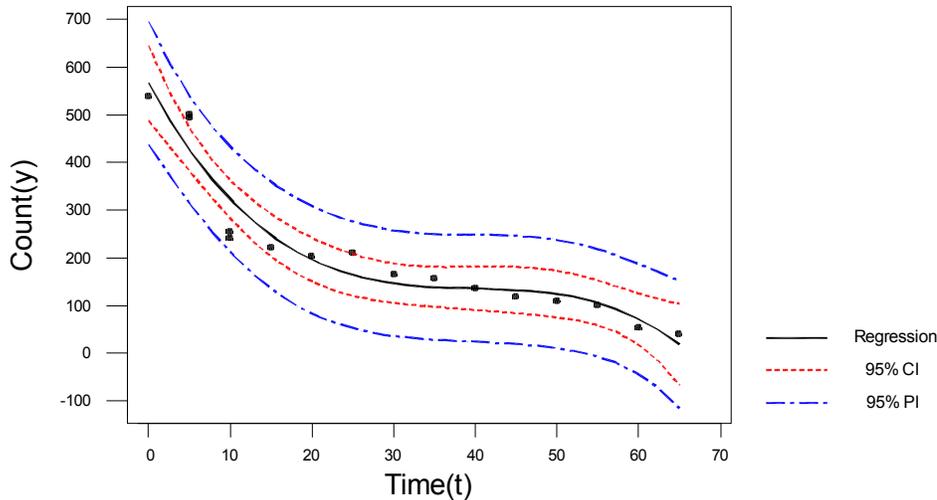


Figure 4. Confidence versus prediction intervals in nonlinear regression.

Optimization

Nonlinear regression requires an optimization algorithm to find the value of \hat{p} that minimizes the sum of squares. This is often difficult. Nonlinear least squares optimization algorithms have been designed to exploit the special structure of a sum of the squares objective function. If the objective function is formulated as:

$$\text{Minimize } f(x) = \sum_{i=1}^n [T_i(x)]^2, \text{ where } T_i(x) \text{ is the } i^{\text{th}} \text{ least squares term (residual).}$$

If $f(x)$ is differentiated twice, terms of $T_i(x)$, $T_i''(x)$ and $[T_i'(x)]^2$ result. By assuming that the former term tends to zero near the solution (assuming that we can get the residuals $T_i(x)$ close to zero), the Hessian matrix of second derivatives of $f(x)$ can be approximated using only first derivatives of $T_i(x)$.

An algorithm that is particularly well-suited to the small-residual case is the Gauss-Newton algorithm. Gauss-Newton algorithms exhibit quadratic convergence rates near the solution. By exploiting the structure of the problem, the second order convergence characteristics of a full Newton algorithm can be obtained using only first order information from the least squares terms. Other optimization methods used in nonlinear regression are based on sequential quadratic programming. Most statistical packages now offer at least one optimization method, and sometimes several, to solve for the optimal parameter values. However, these solvers may experience difficulty when the residuals at the optimal solution are significant.

Nonlinear regression provides a much more general framework than linear regression, and can be used to calibrate numerical parameters of a model. In nonlinear regression, the estimated coefficients are usually more closely tied to a physical quantity than in linear regression. However, it is more expensive and requires the use of optimization methods as well as numerical or analytical calculation of derivatives when finding the prediction intervals.

Background

The Kennedy and O'Hagan approach to CUU involves Bayesian analysis and Gaussian processes. We provide some background for the subsequent discussion here.

Bayesian Analysis

Bayesian data analysis is a method for making inferences from data using probability models for quantities we observe and for quantities about which we wish to learn [Gelman et. al]. In a Bayesian formulation, one models a relationship between a quantity of interest, such as number of successes of a system in N trials, and other parameters, such as the failure probability of an individual component. The parameters in the probability model are characterized by distributions themselves: these are called *prior* distributions. Data is observed, and the resulting values of the parameters resulting from incorporating the data is are called the *posterior* distributions. In the notation below, θ represents the unobserved quantity of ultimate interest, and \mathbf{x} represents the data.

Discrete case

In the discrete case, the Bayesian formulation is:

$$h(\theta | \mathbf{x}_1, \dots, \mathbf{x}_N) = \frac{f(\mathbf{x}_1 | \theta) \dots f(\mathbf{x}_N | \theta) g(\theta)}{\sum_{\theta} f(\mathbf{x}_1 | \theta) \dots f(\mathbf{x}_N | \theta) g(\theta)}$$

where $\mathbf{x}_1, \dots, \mathbf{x}_N$ are independent and identically distributed observable random vector variables with probability mass function $f(\mathbf{x}|\theta)$. [Press, 1989] Note that $f(\mathbf{x}|\theta)$ denotes the mass function of random vector \mathbf{x} conditional upon another variable $\Theta = \theta$. Θ is assumed to be unobservable, and θ denotes the numerical value at which Θ is conditioned. In this case, we are assuming that Θ is discrete, and $g(\theta)$ is the probability mass function. The posterior probability density function of θ for a given set of observed data is $h(\theta|\mathbf{x})$.

Note that the denominator of (1) only depends on the \mathbf{x}_i 's and not on θ . Bayes formula is often written as:

$$h(\theta | \mathbf{x}_1, \dots, \mathbf{x}_N) \propto L(\mathbf{x}_1, \dots, \mathbf{x}_N | \theta) g(\theta)$$

where $L(\mathbf{x}_1, \dots, \mathbf{x}_N | \theta) = f(\mathbf{x}_1 | \theta) * \dots * f(\mathbf{x}_N | \theta)$ = the likelihood function of the data given the parameter θ for independent data. The expression (2) is a statement that the posterior distribution is proportional to the likelihood times the prior distribution.

Continuous case

The formulation is identical to (1), only the parameter θ is now a continuous parameter with prior density $g(\theta)$. An alternative approach to expressing equation (1) is to use the likelihood function $L(\mathbf{x}_1, \dots, \mathbf{x}_N | \theta)$ instead of the conditional probability density functions $f(\mathbf{x}_i | \theta)$. So, one way of expressing Bayes' Theorem in the continuous case is:

$$h(\theta | \mathbf{x}_1, \dots, \mathbf{x}_N) = \frac{L(\mathbf{x}_1, \dots, \mathbf{x}_N | \theta)g(\theta)}{\int L(\mathbf{x}_1, \dots, \mathbf{x}_N | \theta)g(\theta)d\theta}$$

Hypothesis Testing

Bayesian analysis can be useful for hypothesis testing, specifically in comparing hypothesis H (often called the null hypothesis) against an alternative hypothesis A. For example, in the discrete case, we may have H: $\theta = \theta_0$ and A: $\theta = \theta_1$. Let T be a test statistic based on a sample of N observations, $T \equiv T(\mathbf{x}_1, \dots, \mathbf{x}_N)$. Then Bayes' Theorem states that

$$p(H | T) = \frac{p(T | H)p(H)}{p(T | H)p(H) + p(T | A)p(A)}$$

where $p(H)$ and $p(A)$ denote the prior probabilities of H and A (these probabilities sum to one).

Likewise, for hypothesis A, we have:

$$p(A | T) = \frac{p(T | A)p(A)}{p(T | H)p(H) + p(T | A)p(A)}$$

Taking the ratio of 4 and 5, we have:

$$\frac{p(H | T)}{p(A | T)} = \left[\frac{p(H)}{p(A)} \right] \left[\frac{p(T | H)}{p(T | A)} \right]$$

This is interpreted as the posterior odds ratio in favor of H is equal to the product of the prior odds ratio and the likelihood ratio. If the posterior odds ratio exceeds one, we accept H, otherwise we reject H and accept A. Also, the ratio of the posterior odds to the prior odds is sometimes called "Bayes factor" and only depends on the sample data T. If we assume equal probability on H and A, the posterior odds ratio is just equal to the likelihood ratio.

This approach to hypothesis testing does differ somewhat with classical hypothesis testing. In classical hypothesis testing, we are usually interested in testing $H: \theta = \theta_0$ vs. $A: \theta \neq \theta_0$. If θ is a very small amount away from θ_0 , say by distance $\epsilon > 0$, then for sufficiently large N , we will always reject the null hypothesis. In contrast, the Bayesian hypothesis testing is based on the relative likelihood of the two hypotheses given the data and the prior odds. So, in the case of θ being very close but not exactly θ_0 , the Bayesian method would choose H over A .

The approach outlined above for comparing two hypotheses can be extended. One common extension is to have the alternative hypothesis be a “non-informative” distribution such as a uniform distribution. Then, testing H vs. A gives some indication of the “correctness” of H relative to knowing very little about the distribution of the prior. Dr. Mahadevan has applied this to a reliability model and has a paper outlining this form of testing. [Zhang and Mahadevan, 2003]

Controversy with Bayesian Inference

The Bayesian framework allows one to integrate observed data and prior knowledge. In this case where one has no data or very little data, the posterior distribution is equal to or very close to the prior distribution. In the case where there is a lot of data, and especially in the case where the likelihood function differs from the prior distribution, the posterior distribution is dominated by the likelihood function. In the context of many of the science and engineering problems encountered at Sandia, we need to seriously question the usefulness of the Bayesian approach. While the approach is very intuitive and reasonable conceptually, implementation may be difficult depending on the choice of parameters (more about this later). In addition, there is the important question of how the Bayesian updating will be performed in a situation characterized by computationally expensive models and expensive testing. If we only have a few observed data points, then our posterior distribution is likely to be very similar to the prior and we haven't learned that much. If we have lots of data, then we should probably use a maximum likelihood approach which may be simpler and easier to defend than formulating a prior distribution.

One criticism of Bayesian statistics that we need to be aware of is the formulation of the prior distribution. Ideally, the prior distribution is supposed to be obtained from subjective judgment and previous experience. In practice, the prior is often chosen from a family of distributions that makes the calculation of the posterior distribution tractable. These families are called “conjugate prior” distributions and will be discussed below in more detail. The biggest problem is that one often ends up estimating the “hyperparameters” of the prior distribution from the current data set, using maximum likelihood techniques or sample moments. Calculating the parameters of the prior distribution from the current data set AND using this data to calculate the likelihood terms in Bayes' equation violates the theorem: the prior distribution is supposed to only depend on its parameters and not on the data. This situation results in incoherent estimators.

Examples

Examples are helpful to see the implications of using Bayesian inference. To start with, consider the binomial distribution. This is often used to model the number of successes, x , in n independent trials. If θ = the probability of success on a single trial, then the probability mass function for x is:

$$f(x | \theta) = \binom{n}{x} \theta^x (1 - \theta)^{n-x}$$

Let us assume that θ can have two possible values, 0.3 and 0.6, with the following prior mass function: $P\{\theta=0.3\}=g(0.3)=0.1$ and $P\{\theta=0.6\}=g(0.6)=0.9$. According to Bayes' Theorem, the posterior probability mass function from Equation (1) is:

$$h(\theta | x) = \frac{\theta^x (1 - \theta)^{n-x} g(\theta)}{(0.3)^x (0.7)^{n-x} g(0.3) + (0.6)^x (0.4)^{n-x} g(0.4)} \text{ for } \theta=0.3 \text{ and } 0.6.$$

Suppose $n=5$ and $x = 2$. Then $h(0.3|x)=0.13$ and $h(0.6)=0.87$. Thus, $h(\theta|x)$ does not differ that much from $g(\theta)$, since the update was only based on five points. The posterior distribution does reflect the fact that in this set of data, θ is closer to 0.3 than 0.6 and so the probability of $\theta=0.3$ has risen from the prior value of 0.1 to the posterior value of 0.13.

A related example shows how the update differs if we assume that θ is a continuous parameter between zero and one. In this case, the posterior density is:

$$h(\theta | x) = \frac{\theta^x (1 - \theta)^{n-x} g(\theta)}{\int \theta^x (1 - \theta)^{n-x} g(\theta) d\theta}$$

If we assume a uniform prior, then $g(\theta) = 1$ for $0 < \theta < 1$, and $g(\theta) = 0$ elsewhere. In this case, the posterior distribution is given by:

$$h(\theta | x) = \frac{\theta^x (1 - \theta)^{n-x}}{B(x+1, n-x+1)}$$

where B is the beta distribution. The posterior distribution is a beta distribution with a mode at the value $\theta=x/n$. The mean of θ given x is $E(\theta|x)=(x+1)/(n+2)$.

Often a beta distribution is assumed for the prior density function $g(\theta)$, where θ is the parameter of the binomial distribution. In this case, $g(\theta)$ is given by:

$$g(\theta) = \frac{\theta^{\alpha-1} (1 - \theta)^{\beta-1}}{B(\alpha, \beta)}$$

where $0 < \theta < 1$, $0 < \alpha$, and $0 < \beta$. Note that in this case, we are postulating that the prior distribution for one parameter is characterized by a two-parameter distribution. Thus, to specify the prior distribution, we need to determine α , and β . The mean of this beta distribution is given as $\alpha/(\alpha+\beta)$, and the mode is given by $\alpha-1/(\alpha+\beta-2)$. If someone specifies the mean and the mode, or the mean and the variance, it is possible to solve the equations to obtain α and β . The posterior distribution of θ given x is also given by a beta distribution:

$$h(\theta | x) = \frac{\theta^{x+\alpha-1} (1-\theta)^{n-x+\beta-1}}{B(x+\alpha, n-x+\beta)}$$

This means that if one is updating the parameter θ that characterizes a binomial likelihood function, and if the parameter θ has a prior distribution that is beta, the posterior distribution is also beta and the parameters of that beta distribution can be obtained very easily from the data. This situation, where the prior and posterior distribution come from the same family of distributions, is called a “conjugate prior” or a conjugate pair. More examples of conjugate priors are listed in the section below.

To demonstrate the continuous case, assume that $g(\theta)$ is given by a beta distribution with $\alpha = 3$ and $\beta = 12$. In this case, $E(\theta) = 0.2$ and the mode of θ is 0.15. If we have $x=2$ and $n=5$ as in the discrete example, we find that the posterior distribution is a beta distribution with parameters $(x+\alpha, n-x+\beta)$, or $B(5,15)$. The mean of the posterior beta distribution is 0.25 and the mode is 0.22. The posterior distribution has changed based on the data.

Conjugate pairs

As mentioned above, there are distribution families which are often used as prior distributions because they have convenient mathematical properties. These families are called natural conjugate families, and the prior distribution is called a conjugate prior. In such cases, performing Bayesian updating usually does not involve complex integration: the posterior distribution is from the same family as the prior, with parameters that can be obtained from the prior parameters and the data.

Table 1 shows some conjugate prior distributions.

Sampling Distribution	Conjugate Prior Distribution
Binomial	Success probability is beta
Negative binomial	Success probability is beta
Poisson	Mean is gamma
Exponential with mean (1/λ)	λ is gamma
Normal with known variance but unknown mean	Mean is normal
Normal with unknown variance but known mean	Variance is an inverted gamma

Table 1. Conjugate priors associated with various sampling distributions

Calculations from Markov Chain Monte Carlo

It is not always possible to formulate a Bayesian analysis with one of the conjugate priors as outlined in the section above. Many times the calculation of the posterior density function involves complex integration. There have been specific methods to approximate Bayesian integrals developed for low-dimensional cases (e.g., Tierney-Kadane, Lindley approximations). To calculate the posterior distribution for higher dimensions, some type of Monte Carlo method is often used to generate samples over which the integrand is calculated. A popular method for doing this is called Markov Chain Monte Carlo (MCMC), where one wants to generate a sampling density that is approximately equal to the posterior density.

The idea behind Monte Carlo Markov Chain is to construct a Markov Chain such that its stationary distribution is exactly the same as the distribution of interest. [Gamerman, 1997] A stationary distribution of a Markov chain with transition probability matrix $P(x,y)$ is f if:

$$f_Y(y) = \sum_x f_X(x)P(x,y)$$

for a discrete state chain. The continuous state equation relates the state of the system after n steps to the state of the system at $n-1$ steps:

$$f_Y^n(y) = \int_{-\infty}^{\infty} p(x,y)f_X^{n-1}(x)dx$$

Another representation that is often used is that we want to obtain $E[f(x)]$

$$E[f(x)] = \int_{-\infty}^{\infty} p(x)f(x)dx$$

in situations where drawing samples from the density function $p(x)$ is not feasible and the inverse transform is not available (note: by inverse transform we mean draw a sample from $U(0,1)$, equate this random number to a cumulative probability from distribution p , then solve for x given this cumulative probability).

The point of using MCMC methods is to generate a Markov Chain $\{X_0, X_1, X_2, \dots\}$ where X_{k+1} only depends on X_k . The distribution of X_k will approach a stationary form as k gets large, but in practice, one has to ignore the first M iterations. That is:

$$\frac{1}{n-M} \sum_{k=M+1}^n f(x) \rightarrow E[f(x)]$$

There are several methods for generating the Markov chain that has a stationary distribution with the properties of interest. Three of the best known are the Metropolis-Hastings algorithm, the Metropolis algorithm, and Gibbs sampling. Here is a brief outline of the Metropolis-Hastings algorithm. In this approach, one needs to define a “proposed density function” for generating the next point, conditional on the previous point generated in the chain. This density function is given by $q(Y|X)$. The density of interest (e.g., the posterior density) is given by $f(X)$.

Metropolis-Hasting algorithm

Set $i=0$.

Repeat until converged:

1. Sample a candidate Y from the proposal density function $q_Y(Y|X_i)$
2. Calculate the acceptance ratio $\alpha(X, Y) = \min(1, \frac{f_X(Y)q_Y(Y|X_i)}{f_X(X)q_X(X_i|Y)})$
3. Sample a uniform (0,1) random variable U
4. If $\alpha(X_i, Y) \geq U$, set $X_{i+1}=Y$, else set $X_{i+1}=X_i$
5. Increment i .

Although the algorithm is simple, there are many issues: how does one choose q , does q have to be a symmetric distribution so that $q(Y|X) = q(X|Y)$, how does one deal with multiple variables, etc.? In the case of multiple variables, there is a stepwise procedure where one has to specify all the full conditional distributions (distributions of one variable conditional on all of the others). Transforming back and forth between conditionals and marginals is not trivial in high-dimension problems. In the case of using a MCMC method to generate a posterior probability at a system level (for example, system reliability or performance), one then has to take the system posterior distribution and work back to the posterior distributions on individual parameters using analytic methods. Finally, the issue of convergence is very important in MCMC: when is the set of generated points a close enough approximation to the posterior that one can stop sampling?

These questions are addressed in more detail:

First, the issue of selecting the proposal density q : In theory, it doesn't matter what density function one chooses for q . In practice, it matters a lot because some densities will converge more quickly than others. There are many options for q , but here are some of the most common ones:

1. Symmetric chains. In this case, $q(X|Y)=q(Y|X)$. For this situation, the acceptance ratio reduces to $\alpha(X, Y) = \min(1, \frac{f_X(Y)}{f_X(X)})$.
2. Random walk chains. A random walk is a Markov Chain defined as $\theta_j = \theta_{j-1} + \omega_j$, where ω_j is a random variable, usually with a multivariate normal distribution f_ω . In this case, $q(Y|X) = f_\omega(X-Y)$, where Y_j is drawn according to the process $Y_{j-1} + \omega_j$. The random walk chain results in proposed values equal to the current value plus noise.
3. Independence chains. In this case, $q(Y|X) = q(Y)$ and the proposed transition is formulated independently of the previous position of the chain.

For the Metropolis-Hasting algorithm, the acceptance rate is critical and the parameters governing the q distribution must be tuned appropriately. If the moves are very small and the acceptance probability is very high, most moves will be accepted but the chain will take many more iterations to converge. If the moves are large, they are likely to fall in the tails of the posterior distribution and result in a low value of the acceptance ratio. One wants to cover the parameter space in a computationally efficient fashion. Many studies have been done on optimal acceptance rates, and the results seem to indicate that 0.45-0.5 is the optimal acceptance rate for 1-dimensional problems, whereas 0.23—0.25 is the optimal acceptance rate for high-dimensional problems. It is often difficult to tune the proposal density parameters to obtain these acceptance rates.

MCMC Convergence

There have been two approaches to analyze convergence of a MCMC: one is more theoretical and examines the structure of the chain itself, and the other is more empirical and analyzes the properties of the observed output from the chain.[Gamerman, 1997] The empirical methods have had more success as applied to real-world problems. One method is to take n parallel chains, and run each of them for m iterations, and build a histogram of the m th iterates. This can be repeated after a further k iterates are obtained. Convergence is accepted when the histograms cannot be distinguished. Raftery and Lewis established a method for determining the length of a MCMC run given estimates of the quantiles of the chain, however it is highly parameterized and based on many assumptions. From a practical standpoint, the examples we have seen indicates that we would need to generate thousands of samples from a MCMC, and throw away hundreds of them in the “burn-in” period.

Gibbs Sampling

Gibbs sampling is an attractive MCMC method because it doesn't require a proposed density: the proposal distribution is built directly from the conditional density functions of the posterior. Because of this, Gibbs sampling is very appropriate for Bayesian analysis and high-dimensional problems. Note that the new points are always accepted in Gibbs sampling.

Gibbs Sampling Algorithm

Set $i=0$, and initialize the chain as $X^0 = (X_1^0, X_2^0, \dots, X_d^0)$ for a d -dimensional random vector X .

Repeat until converged:

Obtain new values of $X^i = (X_1^i, X_2^i, \dots, X_d^i)$ through success generation of values:

- a. $X_1^i \sim \pi(X_1 | X_2^{i-1}, \dots, X_d^{i-1})$
- b. $X_2^i \sim \pi(X_2 | X_1^i, X_3^{i-1}, \dots, X_d^{i-1})$
- .
- .
- .
- c. $X_d^i \sim \pi(X_d | X_1^i, X_2^i, \dots, X_{d-1}^i)$

Increment i .

Thus, at each stage when one is calculating a particular value for an individual variable, it is done based on the “full conditional” distribution of that variable with respect to the other variables. The latest information for the other variables is used in the conditioning.

Note that with Gibbs sampling, one ends up getting a joint distribution by only sampling the conditionals. The BUGS software is highly recommended for Gibbs sampling. This software is worthy of further examination for us. It is public domain software that has been around for several years and is probably the most commonly used for Gibbs sampling. There is more detail about BUGS in the software section below.

Software

At this point, we have evaluated three software packages for Bayesian analysis: FirstBayes, BUGS, and Yadas. FirstBayes is a program written by Anthony O’Hagan for teaching and for people wanting to work through some examples and learn Bayesian statistics. It is fairly easy to use, and quite useful for showing how a prior distribution or likelihood function will affect the posterior distribution in various situations. The allowable distributions are conjugate priors, as outlined above. FirstBayes runs on Windows and requires some system configuration to start. It is not a long-term solution for us, but it may be useful to provide some ideas for test problems or prototype examples this year. It does have a GUI. One of the nice features is that it overlays the prior, posterior, and likelihood function on top of each other in a graph so that one can compare them.

BUGS is based on Gibbs sampling. BUGS is the MCMC software with the longest history. It was developed by MRC Biostatistics group and Imperial College School of Medicine, St. Mary’s, London. There is a UNIX version that is publicly available, and we downloaded it. The Windows version would require some type of license. BUGS has very good documentation and detailed examples. It allows a fair amount of flexibility in the problem setup, but the user cannot alter the parameters of the MCMC chain itself.

YADAS stands for Yet Another Data Analysis System. YADAS was developed by Todd Graves in the Statistical Sciences Division at LANL. It is open source software, written in Java. YADAS has similar capabilities to BUGS. It is more flexible than BUGS, but somewhat harder to use in that the user has to write Java classes to run a problem vs. use the BUGS command interface.

To demonstrate the software on a real problem, we tested FirstBayes, BUGS, and YADAS on the Rosenbrock function, given by: $f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$. A contour plot of this function is shown in Figure 5, for variable bounds $-2 \leq x_1, x_2 \leq 2$.

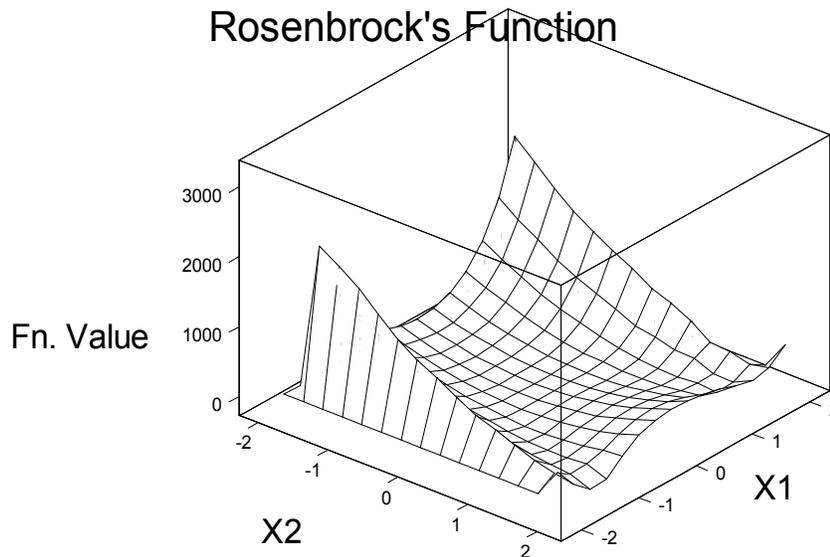


Figure 5. Rosenbrock's Function

The unique solution to the optimization problem: $\min f(x_1, x_2)$ over this domain is given by the point $(x_1, x_2) = (1, 1)$ where the function value is zero.

A test function such as the Rosenbrock function is nice to use for a Bayesian analysis because we have the following:

1. A set of samples of the function over the input space
2. The "true" function
3. An approximation of the function (given by the surrogate)

We used the surrogate-based optimization capability within DAKOTA to generate 110 sample values.

Example 1: Binomial Model

The first example we show is one where we are interested in the probability of failure over the input space. Arbitrarily, we defined the probability of failure as the probability that the response is greater than 1000. For the 110 LHS samples performed during the surrogate run, 13 samples had objective function values > 1000 . This corresponds to a probability of failure estimate for the sample of 0.118.

To show how we might use Bayesian analysis to obtain a posterior distribution on p , the probability of failure, say we use a binomial distribution to model failures. That is, the probability that one will have k failures in n trials is given by:

$$\Pr(k | p) = \text{Bin}(k | n, p) = \binom{n}{k} p^k (1-p)^{n-k}$$

We are interested in obtaining the probability distribution of p , given the data. That is, we want $\Pr(p|k,n)$. To do this, we first need to specify a prior distribution for p . The conjugate prior distribution for p is a beta distribution: $\Pr(p) \propto p^{\alpha-1}(1-p)^{\beta-1}$, so $p \sim \text{Beta}(\alpha, \beta)$. The posterior distribution for p given k failures out of n trials is:

$\Pr(p | k) \propto p^{k+\alpha-1}(1-p)^{n-k+\beta-1}$, so the posterior distribution is a $\text{Beta}(\alpha+k, \beta+n-k)$.

Note that the mean of the posterior distribution, which can be interpreted as the posterior probability estimate of failure for a future sample from the population, is:

$E[p | k] = \frac{\alpha + k}{\alpha + \beta + n}$. This value lies between the sample value k/n and the prior mean, $\alpha/(\alpha+\beta)$.

In this example, we arbitrarily created a prior beta distribution for p , assuming that my prior knowledge was that the mean probability of failure was 0.10. The beta distribution we chose was $\text{Beta}(10,90)$. We used the FirstBayes software to generate the posterior distribution. The prior information is shown in Figure 6.

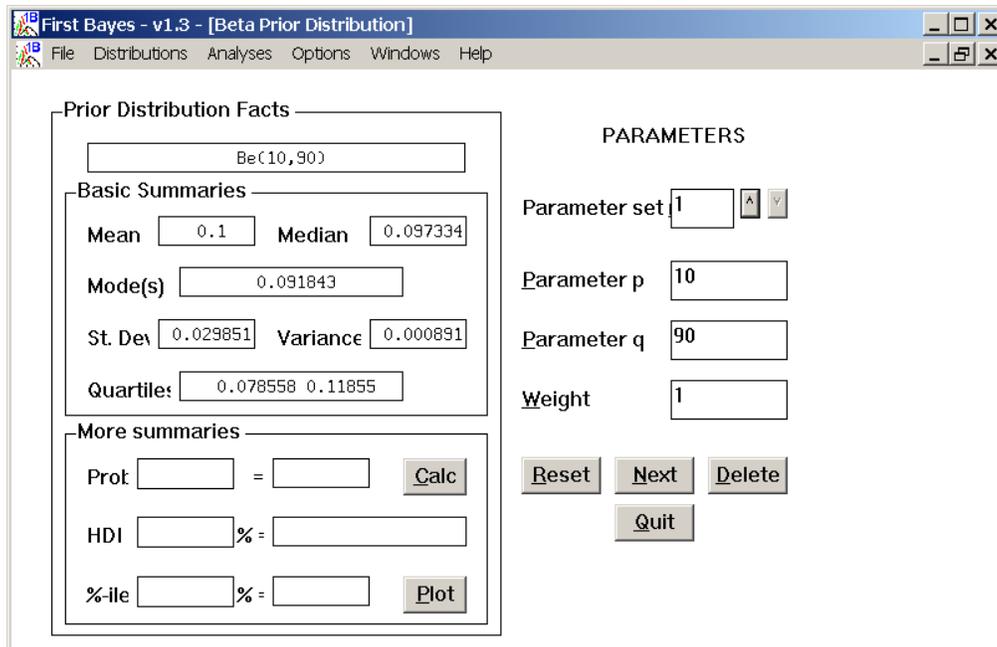


Figure 6. Beta Prior for Binomial Failure Example

The plot showing the prior, the posterior, and the likelihood function is shown in Figure 7:

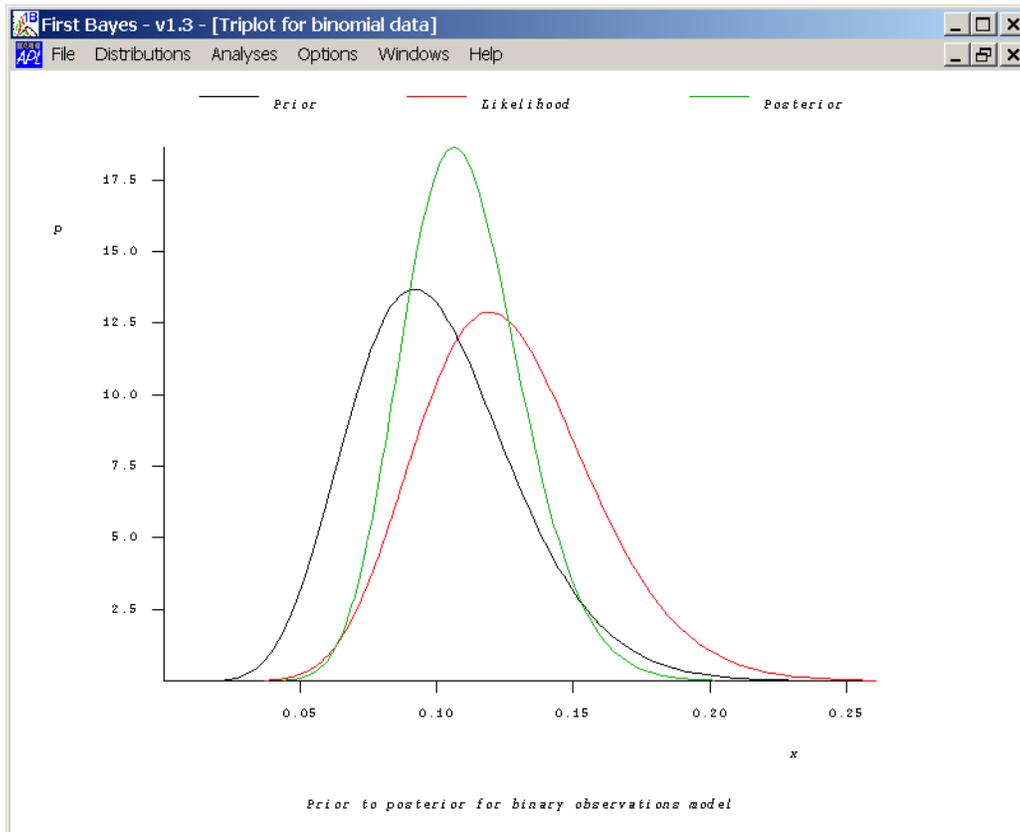


Figure 7. Bayesian Updating of the Beta Distribution for p , the Failure Probability

This shows that the posterior distribution is between the prior and the likelihood function. In fact, the posterior distribution shown in Figure 8 is a Beta(23,187) which is the formula $\text{Beta}(\alpha+k, \beta+n-k)$ since we had 13 failures and 97 successes in the original 110 points.

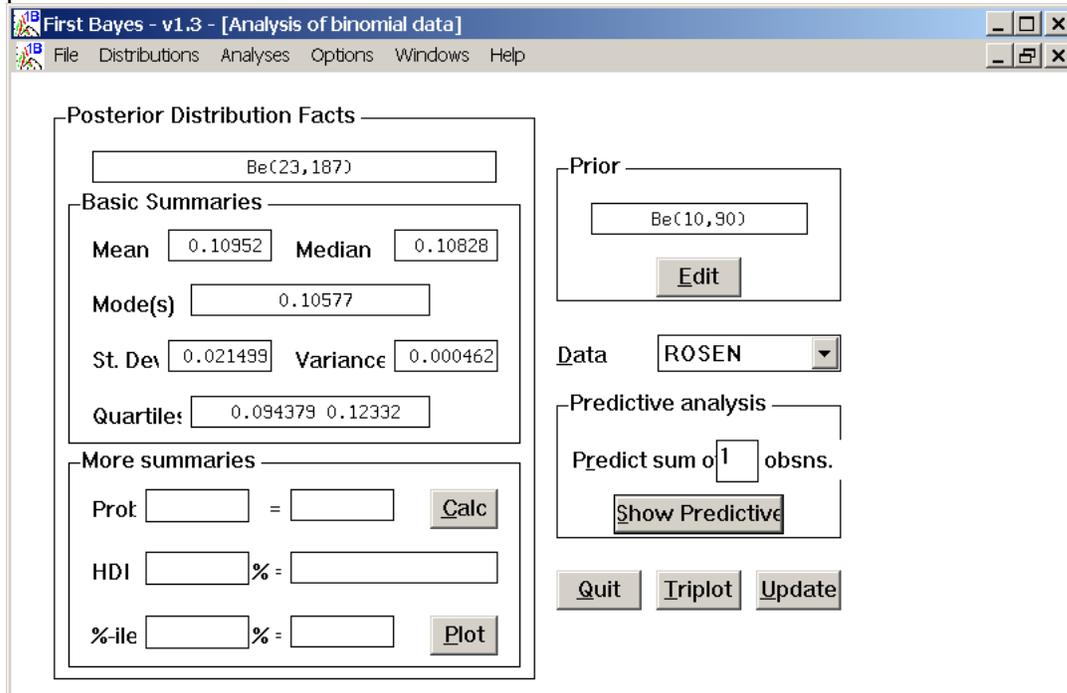


Figure 8. Posterior Distribution Summary

The posterior distribution would be different if we assumed a different prior. For example, with a prior given by a Beta distribution(1,9), the mean is still 0.1 but the variance is much higher, and the posterior distribution is now much closer to the likelihood (less “weight” is given to the prior). This is shown in Figure 9.

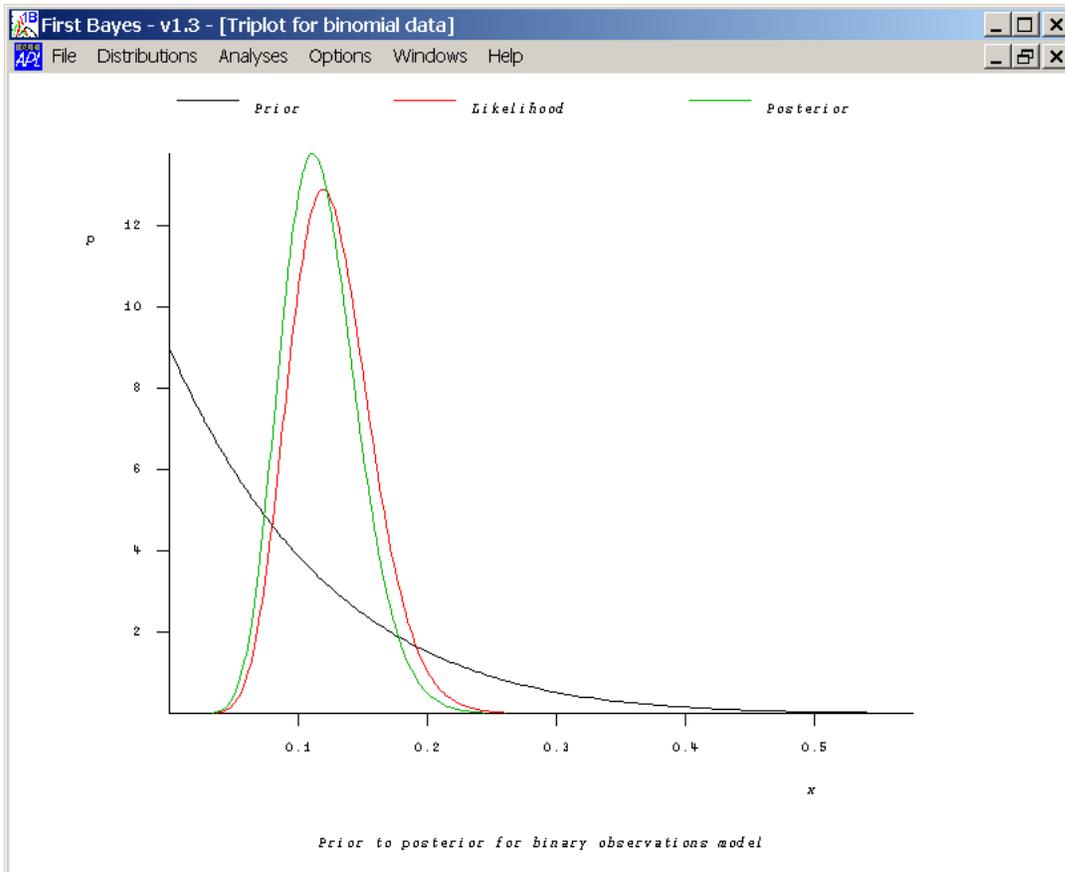


Figure 9. Binomial Failure Example with Be(1,9) prior distribution on p

In BUGS, the input file looks like:

```

model bino;
const
  N = 110; # number of observations
var
  K[N],Y[N],p;
data K,Y in "vol1/bino/bino.dat";
inits in "vol1/bino/bino.in";
{
  p ~ dbeta(10,90);
  for (i in 1:N) {
    K[i] ~ dbin(p ,Y[i]);
  }
}

```

The data in read by this file is in the form $K[i], Y[i]$, where $K[i]$ = number of failures and $Y[i]$ = number of trials thus far. $K[i]$ is distributed as a binomial distribution with failure parameter p on number of trials $Y[i]$, and the parameter p is distributed as a beta

distribution with parameters (10,90) as in the FirstBayes example shown above. The data file for the BUGS example looks like:

```

0 1
0 2
0 3
0 4
1 5
1 6
1 7
1 8
1 9
1 10
1 11
1 12
2 13
2 14
3 15
4 16
4 17
4 18
etc.

```

The output results from running this BUGS example are shown below. There are two outputs: summary statistics relating to the run (statistics on the output distribution of p), and a file with the results of sampling p according to the Gibbs MCMC procedure. The summary output is as follows. First, we ran 500 samples to account for initialization effects. Then, we ran 1000 samples. This generated a p with a mean value of 0.1219 and standard deviation of 4.174E-3. We then ran another 10000 samples to see if the samples generated by this Markov chain would change. They did not change significantly:

```

Bugs>update(1000)
time for 1000 updates was 00:00:00
Bugs>stats(p)
      mean      sd    2.5% : 97.5% CI  median  sample
1.219E-1 4.174E-3 1.136E-1 1.298E-1 1.220E-1 1000
Bugs>update(10000)
time for 10000 updates was 00:00:00
Bugs>stats(p)
      mean      sd    2.5% : 97.5% CI  median  sample
1.220E-1 4.162E-3 1.140E-1 1.303E-1 1.220E-1 11000

```

The sample output is simply a list of values for p , starting at sample 501 because that is when we started recording data from the chain:

```

501 1.21288E-1
502 1.24131E-1

```

```

503      1.24431E-1
504      1.26233E-1
505      1.27563E-1
506      1.14140E-1
507      1.19549E-1
508      1.24497E-1
509      1.20018E-1
510      1.19420E-1
511      1.26276E-1
512      1.18542E-1
513      1.27141E-1
514      1.20705E-1
etc.

```

The binomial example in YADAS is similar, though more complicated to specify. There is an input java class file. We will not copy the entire file, but shown below is the heart of the update procedure using the YADAS java classes:

```

MCMCParameter[] paramarray = new MCMCParameter[]
{
    x = new MCMCParameter ( d.r("x"), d.r(1.0), direc + "x"),
    y = new LogitMCMCParameter ( d.r("y"), d.r("ymss"), direc + "y"),
};

MCMCBond betabond, binomialbond;

ArrayList bondlist = new ArrayList ();

// y ~ Beta(a,b) and x ~ Binomial (n, y).
bondlist.add ( betabond = new BasicMCMCBond
( new MCMCParameter[] { y },
  new ArgumentMaker[] {
    new IdentityArgument (0),
    new ConstantArgument (d.r("alpha")),
    new ConstantArgument (d.r("beta")) },
  new Beta () ));

bondlist.add ( binomialbond = new BasicMCMCBond
( new MCMCParameter[] { x, y },
  new ArgumentMaker[] {
    new IdentityArgument (0),
    new ConstantArgument (d.r("n")),
    new IdentityArgument (1) },
  new Binomial () ));

```

There are several ways to specify the inputs for this example in YADAS. We consulted Dr. Todd Graves, the author of YADAS at Los Alamos, about this. He suggested that my original formulation, which is the most concise, is best:

```
1
x|y|ymss|alpha|beta|n|ni
r|r|r|r|r|r|i
13|0.1|0.5|10|90|110|111
```

In this input file, the first line represents the number of data samples. Here, we have taken the 110 data points and assumed that we essentially have one piece of information from it: that there were 13 failures in 110 samples. The second line represents all of the variables in this problem: x(number of failures), y(probability of failure), ymss(standard deviation of the Gaussian distribution used to generate proposals using a random walk Markov chain Monte Carlo method), alpha and beta (parameters of the beta distribution governing the failure probability), and n and ni, the number of trials and the number of trials+1. The third line specifies the variable types: x is a real (r), y is a real, ni is an integer, etc. The fourth line provides the actual data for these variables: x = 13, y = 0.1 (initial estimate), ymss = 0.5, alpha = 10, beta = 90, n = 110.

YADAS does not produce output statistics on the sampling distribution generated, but it does output the number of samples accepted from the proposal distribution in the Markov chain generation. Since we want to keep the acceptance probability around 50% for a one-dimensional problem, we had to play with ymss quite a bit to get this to work out correctly. Also, we had to change y from a regular MCMC parameter to a Logit MCMC parameter (defined between 0 and 1) to get the sampling to work better. The advantage of YADAS is that you can do this (in BUGS you have no control over parameters governing the performance of the MCMC) but the disadvantage is that the formulations are more complex and require greater understanding to use.

As an example output, the acceptance statistics when we ran a 1000 sample Markov chain in YADAS were:

```
java BBEx 1000
0
Update 0: 0:474
```

This means that 474 out of the 1000 samples were accepted. The output samples from YADAS look like:

```
0.1
0.1
0.1
0.1
0.1
0.1
0.09
```

0.084
0.091
0.091
0.091
0.093
0.093
0.093
0.093
0.13
0.13
etc.

As a final analysis of this failure probability estimation problem, we compared the sample distribution of p generated by the BUGS software (which is based on Gibbs MCMC) and that generated by YADAS. To make a fair comparison, we looked at 10000 samples from each, generated after a 500-sample initialization phase. The results are shown in Figure 10. Note that in Figure 10, only 2000 of the 10000 samples are plotted to make the graph readable, but the pattern holds over the full 10000 samples.

The posterior distribution of p generated by YADAS clearly has a much larger variance than the posterior distribution generated by BUGS: $4.5E-4$ vs $1.7E-5$. Also, the means are quite different: 0.109 vs. 0.122. At this point, my suspicion is that the reason the variance is larger may be due to the fact that we aggregated the data in YADAS into “one” piece of failure information: 13 failures in 110 trials. We tried to get YADAS to parse the input in blocks of 10 trials (so there are 11 overall data points, each one the number of failures in 10 samples), but it treated each block of 10 samples as an individual process and created 11 Markov Chains. We did run YADAS with a larger step size (in which case the acceptance probability dropped to 28%) and a smaller step size (in which case the acceptance probability rose to 90%) but in both cases, the variance was nearly the same as the case in Figure 10, and the posterior data had significantly more spread than the BUGS output.

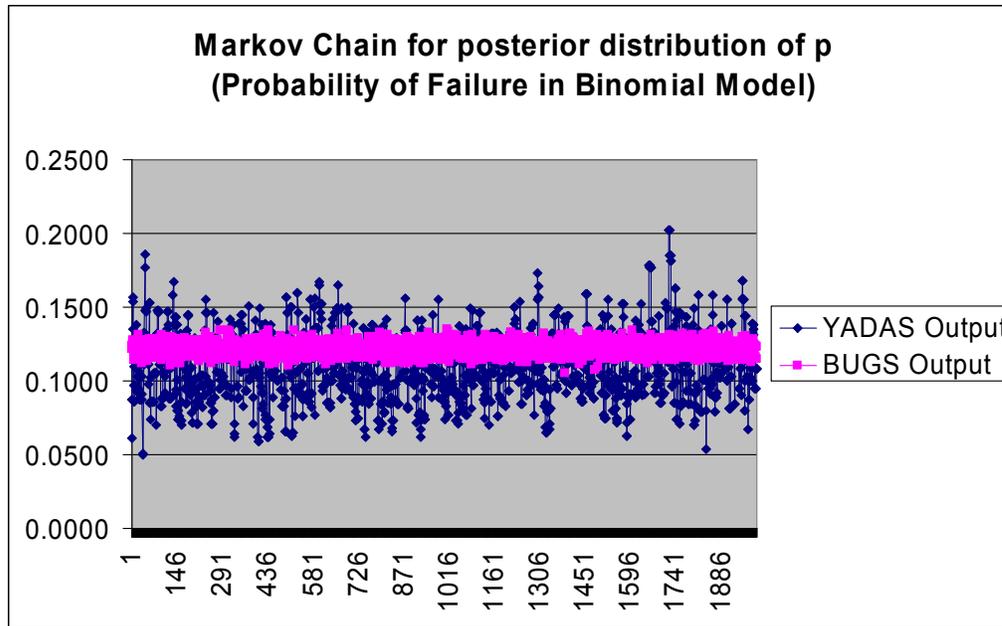


Figure 10. Comparison of YADAS and BUGS output

Gaussian Processes

For those unfamiliar with Gaussian processes (GP), this section provides a very brief introduction. Gaussian Process models are used in response surface modeling, especially response surfaces which “emulate” complex computer codes. Gaussian processes have also been used in conjunction with Bayesian analysis for regression problems and for calibration. Gaussian processes have also been widely used for estimation and prediction in geostatistics and similar spatial statistics applications [Cressie].

Much of this material has been drawn from the work of three experts: Radford Neal and Carl Rasmussen at the University of Toronto, and Chris Williams at Edinburgh University. Carl has a Gaussian processes web site: <http://www.cs.toronto.edu/~carl/gp.html>. He did a dissertation on GP in ‘96 that has some discussion of GPs as surrogates or emulators. Chris Williams’ website is http://www.dai.ed.ac.uk/homes/ckiw/online_pubs.html. Radford Neal has developed some software that we have looked at for demonstration. Neal’s software has the capability to model GP and do Bayesian updating, but it requires significant user knowledge of/development of the code. The software is at: <http://www.cs.toronto.edu/~radford/fbm.software.html>. Neal also has a good overview paper, Technical report 9702 “Monte Carlo implementation of Gaussian process models for Bayesian regression and classification”, which is found on his home page: <http://www.cs.toronto.edu/~radford/papers-online.html>.

A Gaussian process is defined as follows [Williams]: A stochastic process is a collection of random variables $\{Y(\mathbf{x}) \mid \mathbf{x} \in X\}$ indexed by a set X (in most cases, X is \mathcal{R}^d , where d is the number of inputs). The stochastic process is defined by giving the joint probability

distribution for every finite subset of variables $Y(\mathbf{x}_1), \dots, Y(\mathbf{x}_k)$. A Gaussian process is a stochastic process for which any finite set of Y -variables has a joint multivariate Gaussian distribution. A GP is fully specified by its mean function $\mu(\mathbf{x}) = E[Y(\mathbf{x})]$ and its covariance function $C(\mathbf{x}, \mathbf{x}')$. The basic steps in defining/using a GP are:

1. Define the mean function. The mean function can be any type of function. Often the mean is taken to be zero, but this is not necessary. A common representation, for example in a regression model, is that $y(\mathbf{x}) = \sum_j w_j \phi_j(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x})$, where $\{\phi_j\}$ is a set of fixed basis functions and \mathbf{w} is a vector of weights. Combining Gaussian process and a Bayesian approach, one places a prior probability distribution over possible functions and lets the observed data transform the prior into a posterior.
2. Define the covariance. There are many different types of covariance functions that can be used. At this stage, we shall focus on stationary covariance functions where $C(\mathbf{x}, \mathbf{x}')$ is a function of $\mathbf{x} - \mathbf{x}'$ and is invariant to shifts of the origin in the input space. A commonly-used covariance function (KOH) is:

$$C(\mathbf{x}, \mathbf{x}') = v_o \exp\left\{-\sum_{u=1}^d \rho_u^2 (\mathbf{x}_u - \mathbf{x}'_u)^2\right\}$$

This covariance function involves the product of d squared-exponential covariance functions with different lengthscales on each dimension. The form of this covariance function captures the idea that nearby inputs have highly correlated outputs.

3. Perform the “prediction” calculations. Given a set of n input data points $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ and a set of associated observed responses or “targets” $\{z_1, z_2, \dots, z_n\}$, we use the GP to predict the target z_{n+1} at a new set of inputs \mathbf{x}_{n+1} . The target is usually represented as the sum of the “true” response, y , plus an error term: $z_i = y_i + \varepsilon_i$, where ε_i is a zero mean Gaussian random variable with constant variance σ_ε^2 . We assume that the prior distribution on the y_i ’s is given by a GP defined as $Y \sim N(\mathbf{0}, \mathbf{K})$, where \mathbf{K} is the $n \times n$ covariance matrix with entries $K_{ij} = C(\mathbf{x}_i, \mathbf{x}_j)$. Then the prior distribution on the targets z_i is $N(\mathbf{0}, \mathbf{K} + \sigma_\varepsilon^2 \mathbf{I}_n)$. The distribution of the predicted term z_{n+1} is conditional on the data $\{z_1, z_2, \dots, z_n\}$. It is Gaussian with the following mean and variance:

$$\begin{aligned} E[z_{n+1} | z_1, z_2, \dots, z_n] &= \mathbf{k}^T \mathbf{C}^{-1} \mathbf{z} \\ \text{Var}[z_{n+1} | z_1, \dots, z_n] &= C(\mathbf{x}_{n+1}, \mathbf{x}_{n+1}) - \mathbf{k}^T \mathbf{C}^{-1} \mathbf{k} \end{aligned}$$

where \mathbf{k} is the vector of covariance between the n known targets and the new $n+1$ data point: $\mathbf{k} = (C(\mathbf{x}_1, \mathbf{x}_{n+1}), \dots, C(\mathbf{x}_n, \mathbf{x}_{n+1}))^T$, \mathbf{C} is the $n \times n$ covariance matrix of the original data, and \mathbf{z} is the $n \times 1$ vector of target values.

The equations for the mean and variance of the predictive distribution for z_{n+1} both require the inversion of \mathbf{C} , an $n \times n$ matrix. In general, this is a $O(n^3)$ operation. Neal (1997) and Williams (2002) claim that this is feasible on modern computers when n is the order of a few hundred, but that it becomes computationally expensive when n is larger than 1000.

4. Use Monte Carlo Markov Chain (MCMC) sampling to generate posterior distributions on the hyperparameters which govern the covariance function (and the mean function). A common approach in GP is to assume all GPs are zero mean, so the Bayesian updating only involves hyperparameters governing the

covariance function. Since these may be quite complex, one usually still needs a MCMC sampling method to generate the posterior. For example, Neal assumes the ρ^2 terms in the covariance function are distributed as gamma distributions (which themselves are governed by three parameters), so one needs to calculate/update these three parameters for every ρ^2 term.

We examined two existing, public domain codes which have capabilities for Gaussian process models, predictions from GPs, and MCMC to generate the posteriors on the hyperparameters.

The first code is Radford Neal's FBM, Flexible Bayesian Modeling. This code is written in C and command line driven. It has a lot of capabilities: Bayesian regression and classification models based on neural nets or Gaussian processes, Monte Carlo Markov Chain sampling, and clustering methods using mixture models. The documentation is reasonable but somewhat cryptic. Specifically, the formulation of the hyperparameters and the specification of the MCMC are not intuitive at all. It is very difficult to understand what is driving the output results.

The second code is called Netlab, which is a collection of Matlab M-files. This code also has a lot of capabilities, more focused on pattern recognition/classification: it has functions for Principal Component Analysis, K-means clustering, self-organizing maps, multi-layer perception networks, radial basis function networks, some optimization algorithms, MCMC methods, and GPs.

Rosenbrock Example of Gaussian Processes

We ran the FBM and Netlab codes with the Rosenbrock function to see how difficult it was to formulate a GP in these codes, what the outputs look like, etc. From my earlier work looking at Bayesian applications on the Rosenbrock function, we have a data set of 110 output values based on sample values over the input space $-2 \leq x_1, x_2 \leq 2$.

Our data set, then, looked initially like this:

X1	X2	Rosenbrock fn. value
-0.9275	-0.8922	310.8349
1.6726	-0.3028	961.7705
0.1565	0.9491	86.2010
-1.3489	1.6003	10.3279
-1.8911	-1.4831	2567.8868
-0.2727	-0.4198	26.0397
-0.7271	0.7687	8.7475
0.5548	0.3538	0.4098
1.2691	1.2266	14.8165
1.1261	-1.6565	855.2737
0	0	1.00000

.....

However, we found that when trying to formulate a Gaussian process with the “target” data equal to the third column in the dataset above, the inverse of the covariance matrix was extremely ill-conditioned and could not be calculated. So, we tried some transformations. Subtracting the mean from the output data values to create a zero-mean data set was not sufficient: we needed to divide by the standard deviation to create (loosely speaking) a normal (0,1) distribution. Our final data set thus looked like:

X1	X2	Normalized Rosenbrock fn. value
-0.9275	-0.8922	-0.17507
1.6726	-0.3028	0.983807
0.1565	0.9491	-0.57499
-1.3489	1.6003	-0.71007
-1.8911	-1.4831	3.84321
-0.2727	-0.4198	-0.68209
-0.7271	0.7687	-0.71288
0.5548	0.3538	-0.72772
1.2691	1.2266	-0.70208
1.1261	-1.6565	0.794208
0	0	-0.72667

Note that we have not seen any restriction in theory on the form of the output in the GP literature (and normalizing the output should not change the raw correlations between points). However, performing this transformation did allow for the covariance matrix inversion. Note that with the full dataset of 110 points, the covariance matrix with this transformed output data is very ill-conditioned: the ratio of the largest to smallest eigenvalue is 10^{16} . This brings into question the “goodness” of the predictions.

We found that the covariance matrix inversion performed much better on smaller data sets, with only 10 or 20 input points. Tony Giunta confirmed that he has seen this phenomenon in kriging as well. Intuitively, it seems wrong: more data should always be better in terms of creating a response model or performing prediction. But if points are close together in the input space, the resulting covariance matrix can have rows that are nearly dependent, and the inversion falls apart. Neal explains the problem as: “Roughly speaking, the covariances between neighboring training cases are so high that knowing all but one function value is enough to determine the remaining function value to a precision comparable to the level of round-off error.”

There are a couple of ways to rectify this problem. One way is to perform a singular value decomposition on the covariance matrix and remove eigenvalues less than a certain threshold. Andrew Booker of Boeing has proposed an alternative approach to the problem of ill-conditioning. He takes a small set of data points and uses it to estimate a “primary” Gaussian process. He then fixes the parameters of this first GP, and calculates a second GP for a “finer” correlation structure on the remainder of the data points. Booker uses a Gaussian correlation function for the primary GP, but he uses a cubic

spline correlation function for the second GP. Booker claims that the resulting response model given by the sum of these two GPs is much “better” than a standard GP, at least in the context of optimization: the two GP approach resulted in many fewer function evaluations in a surrogate-based optimization comparison.

At this point, all we have done is experiment with the Gaussian process and its output. Below are two graphs showing the Gaussian process output vs. one input, X1, for the Rosenbrock function. Figure 11 is based on 11 input points, while Figure 12 is based on 110 points. These plots were generated using the Netlab software. A few comments: the prediction intervals (based on the covariance matrix) are able to be calculated in the case of 11 input points, but not in the case of 110 input points because the computed inverse of the covariance matrix has negative values.

In terms of the errors, one can calculate the prediction error at a particular point using the difference in the actual value and the GP estimate given by $E[t_{n+1} | t_1, t_2, \dots, t_n] = \mathbf{k}^T \mathbf{C}^{-1} \mathbf{t}$. The FBM software is set up so that you specify which of the full data set you want for “training” and which you want for testing. For example, using the first 60 as training and the remaining 50 as test data, we get an output of predictions at the 50 test points:

Case	Means	Error^2
1	-0.67	0.0008
2	0.19	0.0038
3	-0.63	0.0001
4	-0.41	0.0000
5	-0.40	0.0001
6	0.47	0.0023
7	-0.48	0.0001
8	-0.42	0.0003
9	1.04	0.0036
10	-0.17	0.0006
11	-0.71	0.0003
12	-0.55	0.0011

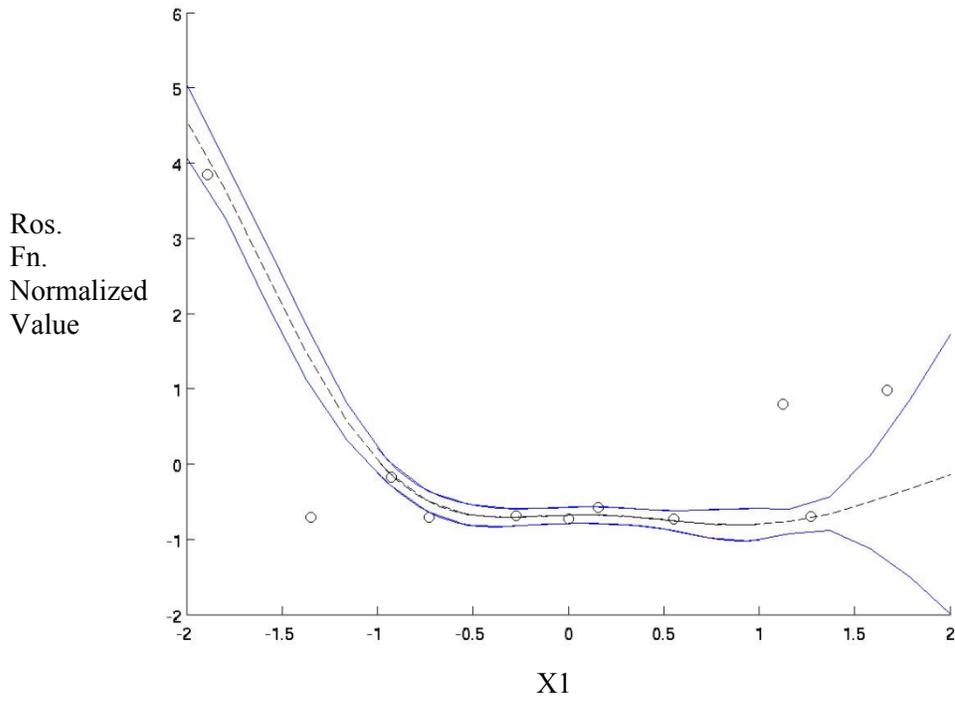


Figure 11. Gaussian Process for Rosenbrock Function based on 11 input points

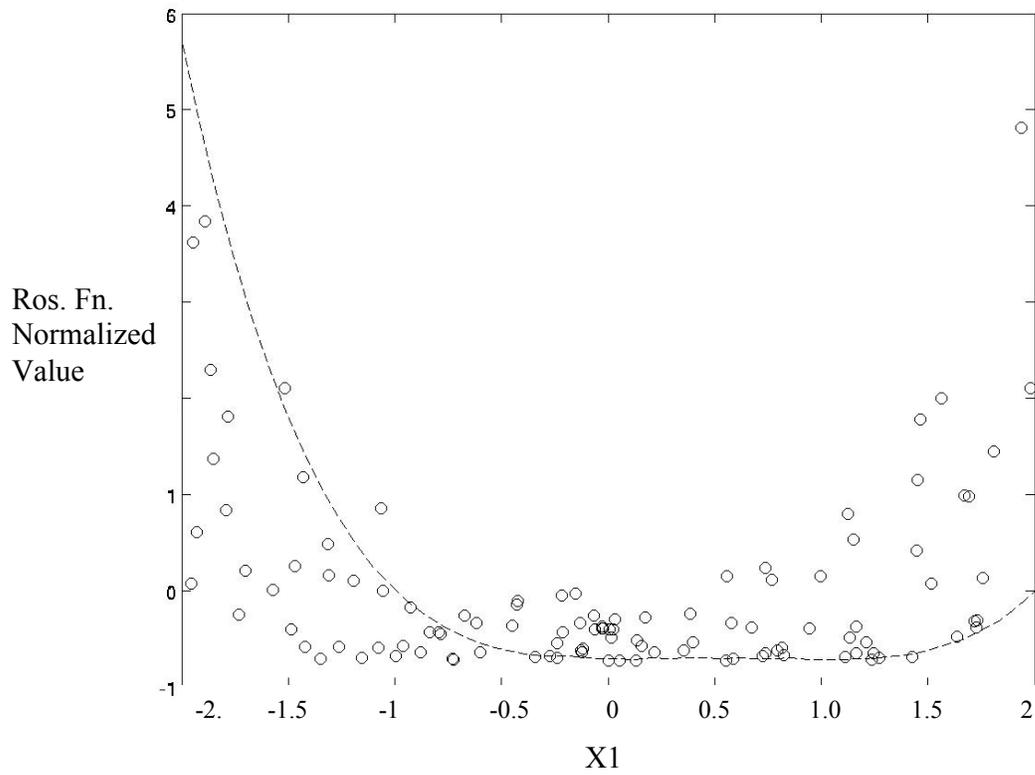


Figure 12. Gaussian Process for Rosenbrock Function based on 110 input points

Both software packages require a bit of manipulation to obtain the actual parameters governing the GP. In Netlab, the output looks like:

```
net =  
  
    type: 'gp'  
    nin: 2  
    nout: 1  
    bias: -1.5269  
min_noise: 1.4901e-08  
    noise: -5.7542  
inweights: [-0.1884 -2.8436]  
    covar_fn: 'sqexp'  
    fpar: 2.0028  
    nwts: 5  
    tr_in: [11x2 double]  
tr_targets: [11x1 double]
```

The fields in governing a Gaussian Process NET are:

```
type = 'gp'  
nin = number of inputs  
nout = number of outputs: always 1  
nwts = total number of weights and covariance function  
parameters  
bias = logarithm of constant offset in covariance function  
noise = logarithm of output noise variance  
inweights = logarithm of inverse length scale for each input  
covarfn = string describing the covariance function:  
    'sqexp'  
    'ratquad'  
fpar = covariance function specific parameters (1 for squared  
    exponential, 2 for rational quadratic)  
trin = training input data (initially empty)  
trtargets = training target data (initially empty)
```

Note that for this example, there are five parameters (nwts): the bias, the noise, the inverse length scale for X1 and X2 in the covariance parameter, and a covariance parameter fpar that gets updated. The updating of the posterior distributions in Netlab is not done with a Bayesian approach, rather it is done with a conjugate gradient method which maximizes the likelihood of the data given the hyperparameters.

The basic steps to generating a Gaussian process, updating the parameters, then using it for prediction in Netlab are: define the GP by defining a “NET” with parameters listed above, initialize the priors, optimize the net (get posterior estimates of the parameters), calculate the covariance/inverse covariance matrix, define the set of Xtest values for which you want predictions, do a “forward” propagation given the GP structure and hyperparameters to calculate an estimated Ytest vector for the Xtest inputs (along with prediction intervals), graph the original data and the predictions.

One feature that is very nice in Netlab is that one can see the matrix manipulations and covariance calculations at each stage. Hybrid MCMC can be used for a Bayesian

updating of the parameters vs. a max likelihood optimization, though we haven't done that yet.

The FBM, Flexible Bayes Modeling software, has many of the same capabilities as Netlab. The output defining the GP is much more cryptic:

```
GAUSSIAN PROCESS IN FILE "lin2.gp" WITH INDEX 100
```

```
HYPERPARAMETERS
```

```
Constant part:
```

```
10.000
```

```
Exponential parts:
```

```
8.826  
0.314 :      0.314      0.314
```

```
Noise levels:
```

```
0.015 :      0.015
```

In this output, the constant part of the covariance is listed followed by the exponential part and the noise levels. All of the parameters (with the exception of the constant term) are given with gamma functions as priors, according to Neal's explanation: "if θ is a hyperparameter, then $\phi = \theta^{-2}$ can be given a gamma prior with density:

$$p(\phi) = \frac{(\alpha/2\omega)^{\alpha/2}}{\Gamma(\alpha/2)} \phi^{\alpha/2-1} e^{-\phi\alpha/2\omega} ."$$
 However, this gamma density has two

hyperparameters associated with it (α is a positive shape parameter and ω is the mean of ϕ). It is not clear what the software is reporting, for example, when it reports 8.826 as the parameter governing the covariance distribution (is it ϕ , θ , α , or ω)? Also, the three parameters below 8.826 (three values all equal to 0.314) are "relevance parameters." Originally we had thought these were lengthscale parameters, but Neal specifies that they "control the amount by which the input has to change to produce a change in the non-linear component of the function that is comparable to the overall scale over which this component varies."

Neal strongly advocates the use of hybrid MCMC methods to generate the posterior distribution. Neal claims that a standard MCMC approach will lead to inefficient random walks over the posterior distribution space. The hybrid approach suppresses part of the "random walk" aspect of MCMC by introducing "momentum" variables that are associated with "position variables" that are the focus of interest (for example, the hyperparameters governing the covariance function). The momentum variables cause the particle to continue in a consistent direction until such time as a region of high energy (low probability) is encountered. At that point, the position "leapfrogs" to another state. This sounds somewhat like simulated annealing within a Markov chain. One problem

with this is that it introduces yet another set of parameters the user must specify – momentum parameters, stepsizes, windowsizes, etc.

Overall, one can specify a GP model, perform the updating, and make predictions with a few command lines of input. However, the input specification is very cryptic, and it is difficult to tell what algorithms or approach is being used without stepping through the code line by line. FBM does produce output in the form of predicted values for our test cases. Also, the FBM software has a variety of functions which let the user see the covariance matrix, the eigenvalues of the covariance matrix, etc.

Prototype SNL Gaussian Process code

To overcome some of the problems with FBM and Netlab, we decided to implement our own version of a Gaussian process model so that we could fully control the form of the basis and the covariance functions, the parameters governing those functions, and the methods to obtain the parameter estimates (Bayesian vs. maximum likelihood, etc.)

The SNL code allows the user to follow the basic steps in generating a Gaussian process: define the GP, initialize the priors, determine posterior estimates of the parameters, calculate the covariance/inverse covariance matrix, define the set of X values for which you want predictions, do a “forward” propagation given the GP structure and hyperparameters to calculate an estimated Y vector for the X inputs along with prediction intervals. The SNL code is discussed in more detail in the example section below.

Thermal Validation Challenge Problem

This section discusses an approach to calibration under uncertainty using data provided as part of the Thermal Validation Challenge problem developed by Dowding and Hills [Dowding and Hills, 2005]. The purpose of this problem is to test some of the validation approaches in a formal way. The problem was designed to “incorporate features that represent the practical realities that are often imposed on modelers and experimentalists in the development and execution of validation experiments”, including experimental error, unit-to-unit variability, and model approximation/model form uncertainty.

The problem is one of heat conduction through a cylinder. There is an analytic solution to this, given by a PDE. Experiments were run at four experimental configurations (involving fixed values of applied heat flux and cylinder length), and one final experiment was run as the “test” for an extrapolated model.

We started by looking at the first experiment. For this experiment, Dowding and Hills provided experimental data: measured temperature histories from time = 0 to time = 2000 seconds. Because the PDE model was inexpensive to run, we were able to generate code predictions exactly at the experimental times, so there was not an issue “matching” the experimental data configuration to the code configuration, although often there is.

The graph of delta, the difference between the experimental data and model prediction is shown in Figure 13 for Configuration 1, Experiment 1. As you can see, there is a difference between the model predictions and the experimental data. This error grows as a function of temperature.

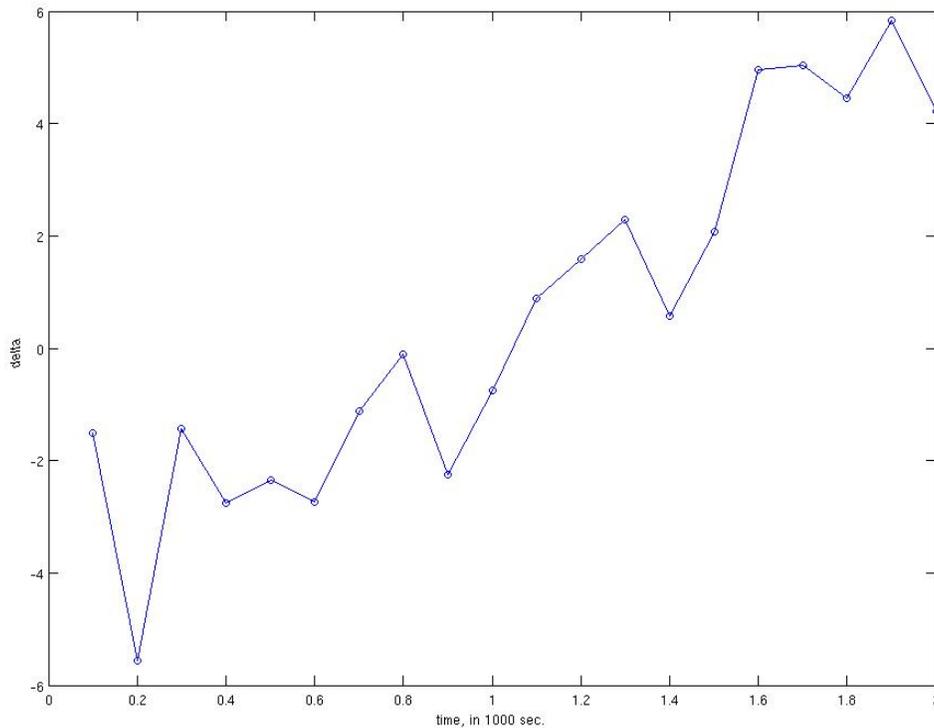


Figure 13. Delta for Configuration 1, Experiment 1

Figure 13 tells us that model discrepancy term is not a zero mean process. We did initially try to model it as a zero mean Gaussian process, to see if the correlation structure could account for the trend, but it did not. The discrepancy vs. temperature plot in Figure 14 (note temperature is scaled to 0-2 from 0-2000 in Figure 14) shows that the GP prediction of the discrepancy term reverts back to the zero mean, constant variance process by 5000 seconds. The Gaussian process mean is given by the center blue line, and the ± 2 standard deviation confidence interval is given by the upper and lower blue dotted lines. If we took this approach, we would not be capturing the trend of the data (which would indicate the mean of the GP on delta should be around 20 degrees by 4000 seconds, indicating that the mean model prediction would be lower than experimental data by 20 degrees at that time).

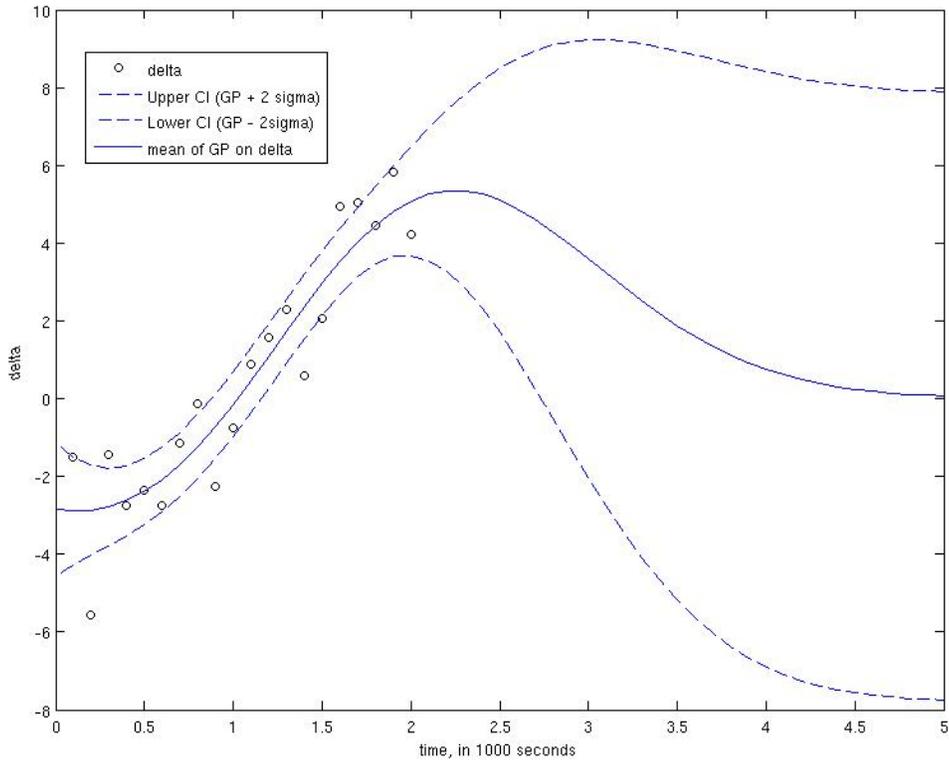


Figure 14. Gaussian process (with a zero mean prior) on delta

Our approach to modeling a discrepancy term with a Gaussian Process

We start with an approach similar to KOH, but with some differences. We are going to assume that the experimental data is equal to some “true” process plus some error, but we assume the true process is equal to code calculation plus a discrepancy term. Therefore, we only have one GP in our approach and do not use a GP as a code emulator:

$$\text{Experimental data} = z_i = \zeta(\mathbf{x}_i) + e_i = \text{Code Output} + \delta(\mathbf{x}_i) + e_i$$

Based on the type of discrepancy we see in Figure 13, which has a clear linear trend, we need to use a Gaussian process with a non-zero mean. Our approach is straightforward:

1. Calculate the model discrepancy as (experimental data – code prediction) for a set of points which “match” in terms of experimental configuration and computer code configuration. Thus, $\delta(\mathbf{x}_i) = z_i - \text{Code Output}$.
2. Examine the model discrepancy term. Fit a polynomial regression model to the data. This is a regression where the dependent variable is the model discrepancy and the independent variables are the independent variables \mathbf{x}_i . Thus, the mean of the GP is now the regression function: $\delta(\mathbf{x}_i)$ is distributed normally with a mean =

- $\mathbf{h}(\mathbf{x})^T \boldsymbol{\beta}$, where $\boldsymbol{\beta}$ are the coefficients of the regression terms $\mathbf{h}(\mathbf{x})$. In this example, x is one dimensional, corresponding to time, and so $\mathbf{h}(\mathbf{x})^T = [1 \ x]$ and $\boldsymbol{\beta} = [\beta_0, \beta_1]$.
3. Calculate the mean and variance of the resulting model discrepancy term $\delta(\mathbf{x}_i)'$, where $\delta(\mathbf{x}_i)' = \delta(\mathbf{x}_i) - \mathbf{h}(\mathbf{x}_i)^T \boldsymbol{\beta}$. The mean of $\delta(\mathbf{x}_i)'$ should be very close to zero. The estimated variance of $\delta(\mathbf{x}_i)'$, σ^2 , is what we will use as a prior estimate of the variance of the Gaussian process. Thus, the total model discrepancy is: $\delta(\mathbf{x}_i) = \delta(\mathbf{x}_i)' + \mathbf{h}(\mathbf{x}_i)^T \boldsymbol{\beta}$, where $\delta(\mathbf{x}_i)' \sim N(0, K + \sigma^2 \mathbf{I})$. The covariance matrix K has entries $K(x, x')$:

$$K(\mathbf{x}, \mathbf{x}') = \exp \left\{ - \sum_{u=1}^d w_u (\mathbf{x}_u - \mathbf{x}'_u)^2 \right\}$$

4. Define the Gaussian process and estimate its parameters. Note that once we have defined a GP and estimated the hyperparameters (either via a Bayesian, maximum likelihood, or other approach), it is possible to calculate the covariance matrix and its inverse, define the set of X values for which you want predictions, do a “forward” propagation given the GP structure and hyperparameters to calculate an estimated Y vector for the X inputs (along with prediction intervals based on the variance estimate at Y). This is the way GP models are used for prediction.

The GP $\delta(\mathbf{x}_i)'$ is defined to have mean zero, variance σ^2 , and covariance matrix given by K . To determine the optimal values of the hyperparameters w_u , we used the constrained minimization algorithm given by `fmincon` in Matlab to find the parameters which maximizes the log likelihood. The log likelihood is the log of the likelihood of the data, given the hyperparameters and this Gaussian process model. There is an analytic form of the log likelihood. For n data points, with the data in vector \mathbf{z} , and C as the covariance matrix $= K + \sigma^2 \mathbf{I}$, the log likelihood is:

$$L = -\frac{1}{2} \log \det C - \frac{1}{2} \mathbf{z}^T C^{-1} \mathbf{z} - \frac{n}{2} \log 2\pi$$

5. Use the Gaussian process model for prediction. After the optimal values of the parameters w_u are obtained, one can then use the GP model for prediction. Given a set of n input data points $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ and a set of associated observed responses or “targets” $\{z_1, z_2, \dots, z_n\}$, we use the GP to predict the target z_{n+1} at a new set of inputs \mathbf{x}_{n+1} . We assume that the prior distribution on the targets z_i is $N(\mathbf{0}, K + \sigma^2 \mathbf{I}_n)$, where K is the $n \times n$ covariance matrix with entries $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$. The distribution of the predicted term z_{n+1} is conditional on the data $\{z_1, z_2, \dots, z_n\}$. It is Gaussian with the following mean and variance:

$$\begin{aligned} E[z_{n+1} | z_1, z_2, \dots, z_n] &= \mathbf{k}^T C^{-1} \mathbf{z} \\ \text{Var}[z_{n+1} | z_1, \dots, z_n] &= C(\mathbf{x}_{n+1}, \mathbf{x}_{n+1}) - \mathbf{k}^T C^{-1} \mathbf{k} \end{aligned}$$

where \mathbf{k} is the vector of covariance between the n known targets and the new $n+1$ data point: $\mathbf{k} = (C(\mathbf{x}_1, \mathbf{x}_{n+1}), \dots, C(\mathbf{x}_n, \mathbf{x}_{n+1}))^T$, C is the $n \times n$ covariance matrix of the original data, and \mathbf{z} is the $n \times 1$ vector of target values.

Note that the equations for the mean and variance of the predictive distribution for z_{n+1} both require the inversion of C , an $n \times n$ matrix. This becomes computationally expensive when n is larger than 1000.

The equations for the mean and variance can be re-run at different potential X values to obtain a graph of the GP. Note that for each possible X you want to examine, you have to re-run the calculation of the covariance matrix and its inverse.

We implemented this in Matlab, vectorizing as many of the operations as we could. Figure 15 shows an example of what results are calculated in the process:

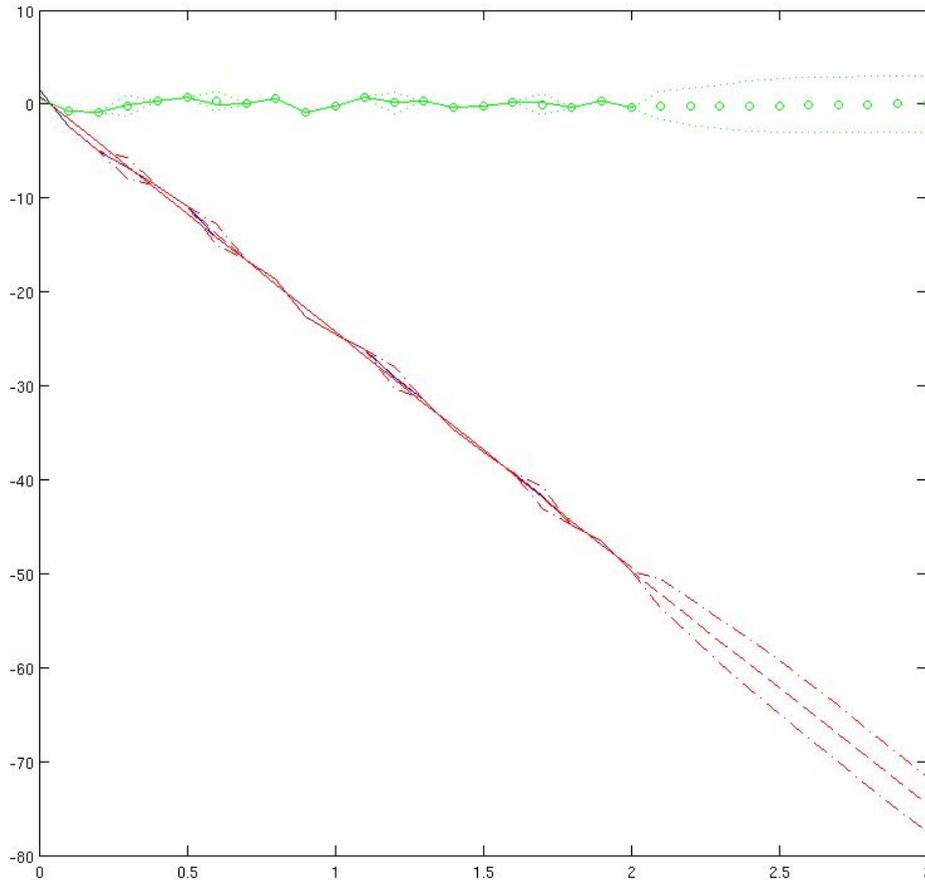


Figure 15. Mean delta vs. time (in 1000s seconds on X-axis), with GP prediction of delta

There is an additional aspect that we need to address in this CUU problem: Determine how to extend this GP approach to model a general case. For example, how would the approach predict the results from Configuration 5, when the GP parameters seemed to be very specific to the particular experiment and set of configuration conditions for that experiment?

The graph shown in Figure 15 is based on one experimental configuration and one code run result. It did not sample from the uncertain parameters when running the thermal

model of heat conduction through a cylinder. Subsequently, we took 20 Latin Hypercube samples from distributions associated with k , ρC_p , L , and q . For each configuration, we generated the appropriate LHS samples according to Table 2:

Parameter/ Configuration	k (W/mK)	ρC_p (W/mK)	L (m)	q (W/m ²)
Configuration 1	Triangular (l.b. = .0378, Mode = 0.5, u. b. = 0.662)	Uniform (l.b. = .331E+6, u. b. = .449E+6)	Normal (μ = .0127, σ = 2.54E-4)	Normal (μ = 1000, σ = 15)
Configuration 2	“	“	Normal (μ = .0254, σ = 2.54E-4)	Normal (μ = 1000, σ = 15)
Configuration 3	“	“	Normal (μ = .0127, σ = 2.54E-4)	Normal (μ = 2000, σ = 30)
Configuration 4	“	“	Normal (μ = .0254, σ = 2.54E-4)	Normal (μ = 2000, σ = 30)
Configuration 5	“	“	Normal (μ = .019, σ = 2.54E-4)	Normal (μ = 3000, σ = 45)

Table 2. Parameters sampled for model runs

We then ran the heat conduction model with the sampled parameters, resulting in sets of output temperature profiles over time. The model output was subtracted from the experimental data to obtain delta values: $\delta(\mathbf{x}_i) = z_i - \text{Code Output}_i$. These samples of the delta temperature profiles are shown by the blue lines in the panels in Figure 16. The red line in each graph shows the mean of the model discrepancy terms. We were surprised that there was so much variation in the deltas: variations in the model inputs caused the model to overpredict or underpredict the experiment by as much as 300 degrees (especially Configuration 3). The difference in the model errors out at 2000 seconds was large, in many cases 150 degrees or more.

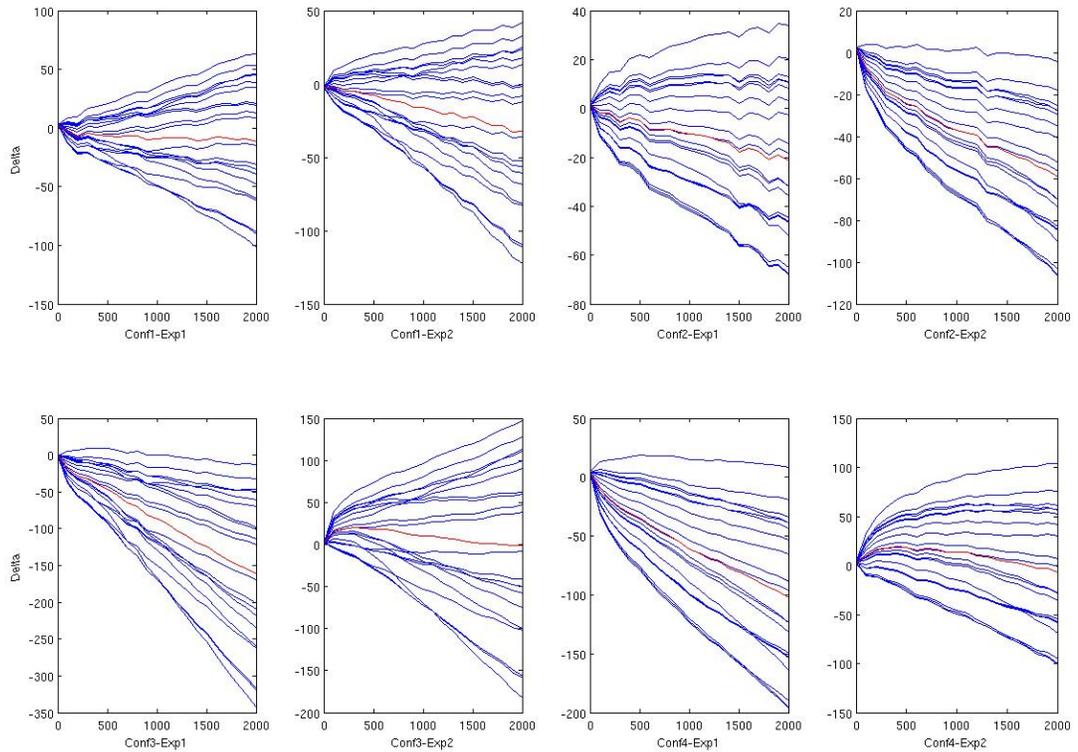


Figure 16. Delta values as a function of model configuration and experiment

Configuration 5 prediction

After having obtained the mean model discrepancy terms for each configuration as shown in Figure 16, we averaged these mean error terms over all configurations. This step is probably the most questionable, in that the assumption is that model discrepancy for configuration 5 is represented by an equal weighting of the model discrepancies from all of the previous configurations and experiments. This “mean of means” model discrepancy term is shown in Figure 17 in red: it is about 50 degrees at 2000 seconds, meaning that the model overpredicts by 50 degrees at 2000 seconds, based on the average of the previous configuration results. Projecting this out to 3000 seconds, we have that the model discrepancy to be about 75 degrees.

The mean model discrepancy in Figure 17 is EXTREMELY linear, so a linear regression was used to fit a basis function for the mean of the Gaussian process. The remaining “noise” is zero mean, as seen by the solid green line in Figure 17. The green circles show the Gaussian process predictions both at points between 0 and 2000 seconds, and from 2000 to 3000 seconds. The predictions follow the actual points exactly in most cases. In some cases there are minor variations between the Gaussian process and the actual prediction, and then the variance of the GP is shown as a dotted green line. The variance goes to a constant by 3000 seconds. The mean of the GP and the regression function are added to obtain the mean delta prediction values, shown by the center dashed red lines. The dashed red lines around the center line are the 2σ upper and lower bounds on the

prediction. Note that the prediction interval on the mean delta value is fairly tight. This is to be expected, as the regression fit is very good and the GP governing the residual error has a small variance.

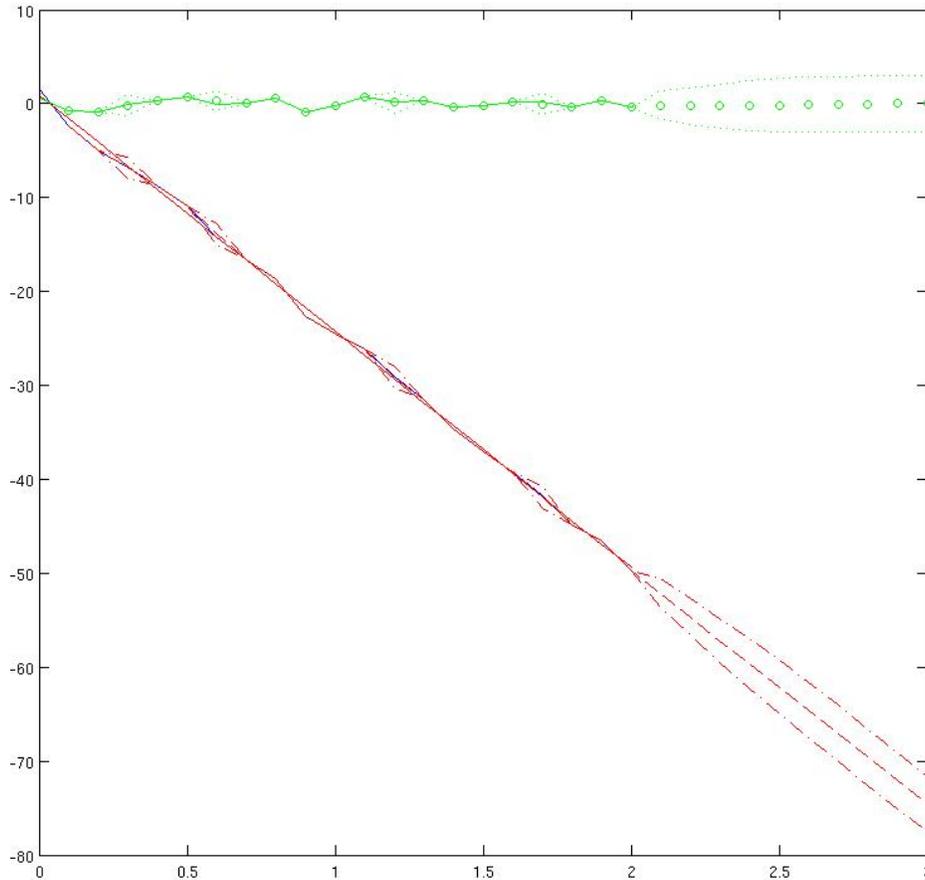


Figure 17. Mean delta vs. time (in 1000s seconds on X-axis), with GP prediction of delta

We then ran the heat conduction model with the configuration 5 parameter values. The results of this run are shown in Figure 18.

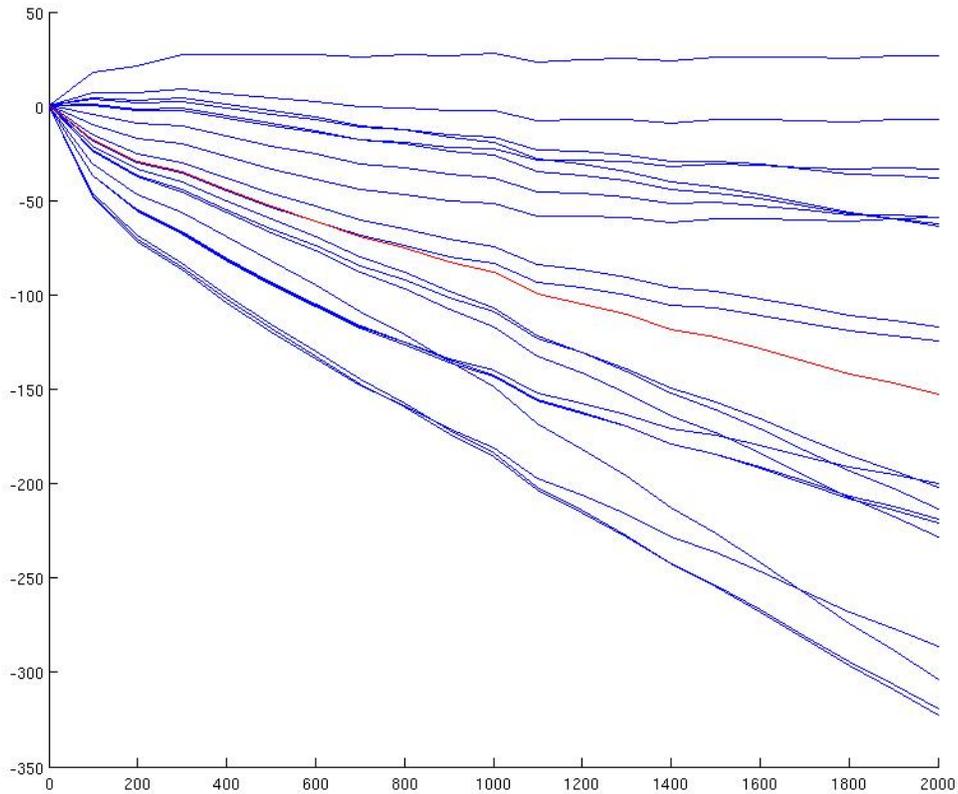


Figure 18. Samples of the Model Discrepancy term for Configuration 5, with mean discrepancy shown in red

As you can see, the model tends to overpredict in Configuration 5, by up to 300 degrees at 2000 seconds. The mean discrepancy term for Configuration 5 has a value of -152.7 at 2000 seconds. This is in contrast to Figure 17, where the mean discrepancy term at 2000 seconds is -49.7. Thus, if we use this approach to modeling discrepancy, we must look some other way of translating the results of model discrepancies from previous runs to the current run. Mean discrepancy is not sufficient. However, this does not imply that the Gaussian process approach is not useful. We need to reconsider exactly what we are modeling with the GP. Looking at the residuals left from fitting a linear regression to the deltas and fitting a GP to that may not be particularly useful, especially if the “structure” of the discrepancy term is mostly captured in the basis function (the regression), as it is in these cases. Our next step is to make the delta term multi-variate, in this case a function of other parameters in the problem such as cylinder length, applied heat flux, etc. We believe that a multi-variate delta term GP will have better predictive modeling capability.

Conclusions

Overall, our conclusions to date from this work in progress are the following: Gaussian process models are powerful emulators. Implementing them requires some knowledge about the data set used and what data will have to be discarded to make the covariance matrix well-conditioned; or additional formulations are needed that are robust to poor covariance conditioning. KOH's formulation of "observation = model + discrepancy + error" is very important because it explicitly separates the model discrepancy term from the model itself. The Gaussian process assumption of the model discrepancy needs further examination, but in general, GP models are extremely flexible at representing a wide variety of functional relationships. The additional assumption that these GP models are governed by parameters that can be updated using Bayesian methods adds a great deal of computational complexity to the picture. The formulation of the joint posterior is difficult. Even if one does not try for analytic solutions but uses MCMC methods, there are many issues to resolve around numerical performance, such as convergence of the MCMC to the correct underlying posterior and determination of tuning parameters, etc.

At this point, we see some interesting paths for further investigation. One is using KOH's formulation but calculating the parameters by Maximum Likelihood Estimation (MLE) methods instead of Bayesian updating. Dennis Cox has pursued this approach [Cox et al.] and it removes the difficulties associated with posterior generation (such as via MCMC). Another is to look at the model emulation term, and replace it with another type of surrogate, perhaps a lower fidelity or reduced order model. This has the advantage of "simplifying" the estimation in that one is solely focused on calculating the parameters for the GP delta model, but it may introduce limitations in terms of the capability to predict.

We wish to emphasize the difference between calibration and validation. Calibration of a computational model is adjusting a set of model parameters associated so that we maximize the model agreement with a set of experimental data (or, in certain cases, a set of numerical benchmarks). Validation of a computational model is quantifying our belief in the predictive capability of a computational model through comparison with a set of experimental data. Uncertainty in both the data and the model is critical and must be mathematically understood to do both calibration and validation correctly.

CUU is therefore a progression of thought that leads to an overlap of the concepts of calibration and validation. For example, the formalism discussed above of incorporating model uncertainty in Bayesian calibration procedures through the model discrepancy term $\delta(\mathbf{x})$ is directly relevant to validation. In validation, we seek to quantify the discrepancy term by comparisons with experiments. From the validation perspective, it is natural to expect that $\delta(\mathbf{x})$ is a random process of some type [Trucano et al., 2001]. The Gaussian process characterization of the model discrepancy discussed above seems to us to be useful in this context as well as in the CUU task. The task of validation should

provide information that helps define specific parameterizations of the model discrepancy and should facilitate the process of calibrating this term.

We believe that predictability of a computational model centers on a specification of intrinsic limitations of the model as well as on our ability to predict model accuracy for specific applications. Directly attacking the problem of characterization of the model discrepancy formalized above in $\delta(\mathbf{x})$ really strikes at the need to quantify model uncertainty in a foundational way that is to some extent independent of the calibration problem of attempting to reduce this uncertainty. A rigorous validation process should achieve the goal of characterizing $\delta(\mathbf{x})$. CUU provides the proper formalism, at least in principle, for using this characterization to improve model accuracy. Thus, we see CUU as an important formalism for linking calibration and validation for quantitative improvement of the predictive content of computational models. Our future work on CUU will elaborate this view more systematically.

List of References

Background papers on Bayesian Calibration Ideas

Beck, M. (1987), "Water Quality Modeling: A Review of the Analysis of Uncertainty," *Water Resources Research*, Vol. 23, No. 8, pp. 1393-1442.

Campbell, K. "A Brief Survey of Statistical Model Calibration Ideas", Los Alamos Technical Report LA-UR-02-3157. 2002.

Campbell, K. Exploring Bayesian Model Calibration: A Guide to Intuition. Los Alamos Technical Report LA-UR-02-7175, 2002.

Cox, D.D., J. S. Park, and C. E. Singer (1996), "A Statistical Method for Tuning a Computer Code to a Data Base," Rice University Report, Tech Report 96-3.

Craig, P. S., Goldstein, M., Rougier, J. C., and A. H. Seheult. "Bayesian Forecasting for Complex Systems using Computer Simulators." *Journal of the American Statistical Association*, **96**(454). 2001

Hoeting, J. A., D. Madigan, A. E. Raftery and C. T. Volinsky, "Bayesian Model Averaging: A Tutorial (with discussion)," *Statistical Science*, **14**(382-401).1999

Kennedy, M. C. and A. O'Hagan. "Bayesian Calibration of Computer Models." *Journal of the Royal Statistical Society*, **63**, pp. 425-464. 2001.

Kennedy, M. C. and A. O'Hagan, "Supplementary Details on Bayesian Calibration of Computer Codes," University of Sheffield, (<http://www.shef.ac.uk/~st1ao/ps/calsup.ps>) (Ref: Kennedy and O'Hagan, 2001)

Lim, B. Y., J. Sacks, W. J. Studden and W. J. Welch. "Design and Analysis of Computer Experiments When the Output is Highly Correlated Over the Input Space." *Canadian Journal of Statistics*, **30**, No. 1 (109-126). 2002. [TGT has copy]

Poole, D. and A. E. Raftery. "Inference for Deterministic Simulation Models: The Bayesian Melding Approach." *Journal of the American Statistical Association*, **95**(452). 2000.

Swiler, L. P., Trucano, T.G. "Treatment of model uncertainty under calibration." *Proceedings of the 9th ASCE Specialty Conference on Probabilistic Mechanics and Structural Reliability*, 2004.

From the SISC – Uncertainty Quantification papers

Higdon, Kennedy, Cavendish, Cafeo, Ryne, and Qiang. "Combining Field Data and Computer Simulations for Calibration and Prediction."

Chinellato, Achermann, and Broker. "Including Covariances in Calibration to Obtain Better Measurement Uncertainty Estimates."

Goldstein and Rougier. “Probabilistic Models for Transferring Inferences from Mathematical Models to Physical Systems.”

V&V Efforts

Dowding, K. J. and R. G. Hills. “*Thermal Validation Challenge Problem*” in draft form (to be a SAND report, 2005).

Easterling, R.G. SAND2005-149814980287. “Statistical Foundations for Model Validation: Two Papers.”

Hanson, K. M. “A Framework for Assessing Uncertainties in Simulation Predictions.” *Physica D*, **133** (1999), pp. 179-188.

Hills, R. G. and I. Leslie. SAND2005-149814980706. “Statistical Validation of Engineering and Scientific Models: Validation Experiments to Application.”

Hills, R. G. and K. J. Dowding. “Statistical Validation of Engineering and Scientific Models: Bounds, Calibration, and Extrapolation. SAND report undergoing review (2003).

All of Dr. Mahadevan’s progress reports

Oberkampf and Trucano. “Verification and Validation in Computational Fluid Dynamics.” *Progress in Aerospace Sciences* **38**(2002), pp. 209-272.

Rutherford, B. M. and K. J. Dowding. SAND2005-149814982336. “An Approach to Model Validation and Model-based Prediction: Polyurethane Foam Case Study.”

Trucano, T. G., M. Pilch, W. B. Oberkampf. SAND2005-149814980341. “General Concepts for Experimental Validation of ASCII Code Applications.”

Trucano, T. G., M. Pilch, W. B. Oberkampf. SAND2005-149814982752. “On the Role of Code Comparisons in Verification and Validation.”

Trucano, T. G., R. G. Easterling, K. J. Dowding, T. L. Paez, A. Urbina, V. J. Romero, B. M. Rutherford, and R. G. Hills (2001), “Description of the Sandia Validation Metrics Project,” Sandia National Laboratories, SAND2005-149814981339, Albuquerque, NM.

Wigley, T. M. L. and B. D. Santer (1990), “Statistical Comparison of Spatial Fields in Model Validation, Perturbation, and Predictability Experiments,” *Journal of Geophysical Research*, Vol. 95, No. 1, pp. 851-865.

R. Zhang and S. Mahadevan (2003), “Bayesian Methodology for Reliability Model Acceptance,” *Reliability Engineering and System Safety*, Vol. 80, 95-103.

Gaussian Processes

Booker, Andrew. “Well-conditioned Kriging Models for Optimization of Computer Simulations.” [Tony – I am not sure where this appeared. Do you know?]

Gibbs, M. and D. J. C. MacKay. *Efficient Implementation of Gaussian Processes*. On the Gaussian process web site: <http://www.cs.toronto.edu/~carl/gp.html>

Mackay, D. J. C. “*Gaussian Processes: A replacement for supervised neural networks?*” Also on the Gaussian process web site.

Neal, Radford. “Monte Carlo Implementation of Gaussian Process Models for Bayesian Regression and Classification.” Technical Report 9702, Dept. of Statistics, University of Toronto (1997).

Neal, Radford. Flexible Bayesian Software documentation: <http://www.cs.toronto.edu/~radford/fbm.software.html>

Nabney, Ian. Netlab software. Documentation and software at: www.ncrg.aston.ac.uk/netlab/

Rasmussen, Carl. “Evaluation of Gaussian Processes and Other Methods for Nonlinear Regression.” Ph.D. Thesis, University of Toronto, 1996.

Williams, Chris (2002). “Gaussian Processes” chapter in *The Handbook of Brain Theory and Neural Networks*, M. Arbib, ed. Cambridge, MA: MIT Press.

Bayesian Analysis/MCMC

Berger, J.O. *Statistical Decision Theory and Bayesian Analysis*. Springer-Verlag, 1985.

Chen, M-H., Shao, Q-M., and J. G. Ibrahim (2000). *Monte Carlo Methods in Bayesian Computation*. Springer-Verlag, New York.

Gamerman, D. (1997), *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference*, Chapman and Hall/CRC, Boca Raton.
Good survey of Bayesian ideas and algorithms, w/ chapters on Gibbs sampling and Metropolis-Hastings.

Gelman, A. J. B. Carlin, H. S. Stern and D. B. Rubin (1995), *Bayesian Data Analysis*, Chapman and Hall/CRC, Boca Raton.

Gilks, W.R., S. Richardson, and D.J. Spiegelhalter (1996). *Markov Chain Monte Carlo in Practice*. Chapman and Hall/CRC, Boca Raton.

Miro-Quesado, G., Del Castillo, E., and J. Peterson. “A Bayesian Approach for Multiple Response Surface Optimization in the Presence of Noise Variables.” Penn State Technical Report – Engineering Statistics Laboratory, 2002. *Journal of Applied Statistics*, 31 (3), pp. 251-270, (2004).

O’Hagan, A. (1994). *Kendall’s Advanced Theory of Statistics*. Vol. 2B: Bayesian Inference. Oxford University Press, New York.

Peterson, J. "A Probability-based desirability function for multiresponse optimization." *Proceedings of the Section on Quality and Productivity*, Annual Meeting of the American Statistical Association, 2000.

Press, S. James. *Bayesian Statistics: Principles, Models, and Applications*. Wiley, 1989.

Press, S. J. (2003), *Subjective and Objective Bayesian Statistics: Principles, Methods and Applications*, 2nd edition, 2003, Wiley, New York.

Robert, C. P. (2001). *The Bayesian Choice*, 2nd ed. Springer-Verlag, New York.

Sivia, D. *Data Analysis: A Bayesian Tutorial*. 1996. [Publisher?]

Gamerman, D. (1997), *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference*, Chapman and Hall/CRC, Boca Raton.

Gibbs, M. and D. J. C. MacKay. *Efficient Implementation of Gaussian Processes*. On the Gaussian process web site: <http://www.cs.toronto.edu/~carl/gp.html>

Mackay, D. J. C. "Gaussian Processes: A replacement for supervised neural networks?" Also on the Gaussian process web site.

Books

Au, Siu-Kui. "On the Solution of First Excursion Problems by Simulation with Applications to Probabilistic Seismic Performance Assessment." CIT Dissertation, May 2001. Note: this thesis provides a lot of detail about importance sampling and Monte Carlo Markov Chain sampling, especially in large sample size situations.

Cressie, N. A. C. (1993), *Statistics for Spatial Data*, Wiley, New York.

Neter, J. W. Wasserman, and M. L. Kutner. *Applied Linear Regression: Regression, Analysis of Variance, and Experimental Designs*. Homewood IL: Irwin, 1985.

Roache, P.J. *Verification and validation in computational science and engineering*. Albuquerque, NM: Hermosa Publishers, 1998.

Ripley, B. D. *Stochastic Simulation*, Wiley. 1987.

Note: Older interesting survey of stochastic algorithms (non Markov-Chain-Monte-Carlo).

Saltelli, A., Chan, K., Scott, E.M. *Sensitivity Analysis*. New York: John Wiley & Sons, 2000.

Spall, J. C. *Introduction to Stochastic Search and Optimization*, Wiley. 2003

Note: Discusses Gibbs, Metropolis-Hastings and Metropolis in a larger Chapter (16) devoted to Markov-Chain-Monte-Carlo for Bayesian search.

Web sites

General link to Bayesian software sites:

http://www.mas.ncl.ac.uk/~ndjw1/bookmarks/Stats/Software-Statistical_computing/Bayesian_software/
<http://astrosun.tn.cornell.edu/staff/loredo/bayes/-software>
<http://www.math.wsu.edu/math/faculty/genz/homepage>
<http://www.cs.toronto.edu/~radford/fbm.software.html>
<http://www.mrc-bsu.cam.ac.uk/bugs/>

Nabney, Ian. Netlab software. Documentation and software at: www.ncrg.aston.ac.uk/netlab/

Neal, Radford. Flexible Bayesian Software documentation:
<http://www.cs.toronto.edu/~radford/fbm.software.html>

NIST Statistical site:

<http://www.itl.nist.gov/div898/handbook/pmd/section1/pmd141.htm>

Wish List

R. A. Bates, R. J. Buck, E. Riccomagno and H. P. Wynn, "Experimental Design and Observation for Large Systems," J. R. Statist. Soc. B, **58**(77-94).1996. (Ref: Kennedy and O'Hagan, 2001)

W. L. Chapman, W. J. Welch, K. P. Bowman and J. E. Walsh, "Arctic Sea Ice Variability: Model Sensitivities and a Multidecadal Simulation," Journal of Geophysical Research, **99**(919-935).1994. (Ref: Lim, Sacks, Studden & Welch, 2002)

D. Draper, A. Pereira, P. Prado, A. Saltelli, R. Cheal, S. Eguilior, B. Mendes and S. Tarantola, "Scenario and Parametric Uncertainty in GESAMAC: A Methodological Study in Nuclear Waste Disposal Risk Assessment," Comput. Phys. Commun., **117**(152-155).1999. (Ref: Kennedy and O'Hagan, 2001)

E. Novak, Deterministic and Stochastic Error Bounds in Numerical Analysis, Lecture Notes in Mathematics, 1349, Springer-Verlag, NY, 1988. (Ref: Kennedy and O'Hagan, 2001; out of print)

A. O'Hagan, "Curve Fitting and Optimal Design for Prediction (with discussion)," J. R. Statist. Soc B, **40**(1-42).1978. (Ref: Kennedy and O'Hagan, 2001)

A. O'Hagan, "A Markov Property for Covariance Structures," Tech. Report 98-13, Statistics Section, University of Nottingham (<http://www.shef.ac.uk/~st1ao/ps/kron.ps>) (Ref: Kennedy and O'Hagan, 2001)

J. S. Hodges, "Uncertainty, Policy Analysis and Statistics (with discussion)," Statistical Science, **2**(259-261).1987. (Ref: Poole and Raftery, 2000)

J. S. Hodges, "Six (or so) Things You Can Do With A Model," Operations Research, **39**(355-365). 1991. (Ref: Poole and Raftery, 2000)

R. E. Kass and A. E. Raftery, "Bayes Factors," Journal of the American Statistical Association, **90**(773-795). 1995. (Ref: Poole and Raftery, 2000)

D. B. Rubin, "Comment on 'The Calculation of Posterior Distributions by Data Augmentation,' by M. Tanner and W. H. Wang, Journal of the American Statistical Association, **82**(543-546). 1987. (Ref: Poole and Raftery, 2000)

D. B. Rubin, "Using the SIR Algorithm to Simulate Posterior Distributions," in Bayesian Statistics 3, eds. J. M. Bernardo, M. H. DeGroot, D. V. Lindley and A. F. M. Smith, Oxford, U. K., Clarendon Press, 395-402, 1988. (Ref: Poole and Raftery, 2000)

S. G. Walker, P. Damien, P. W. Laud and A. F. M. Smith, "Bayesian Nonparametric Inference for Random Distributions and Related Functions (with discussion)," *J. R. Statist. Soc. B*, **61**(485-527). 1999. (Ref: Kennedy and O'Hagan, 2001)

Miscellaneous Papers

D. Draper. "Assessment and Propagation of Model Uncertainty." *J. Royal Statistical Society B*, **57**, No. 1 (45-97). 1995. [TGT has copy]

R. A. Moeed and A. Papritz (2002), "An Empirical Comparison of Kriging Methods for Nonlinear Spatial Point Prediction," *Mathematical Geology*, Vol. 34, No. 4, 365-386. [TGT has copy. "This study compares linear and nonlinear kriging methods with respect to precision and their success in modeling prediction uncertainty."]

APPENDIX: Annotated Bibliography

K. Campbell (2002), “A Brief Survey of Statistical Model Calibration Ideas.”

Los Alamos Report, LA-UR-02-3157

Section 1. Context and Nomenclature

Concerned with computer models and their application to decision making. Computer models in this context are called **simulators** and produce **simulations**.

Simulator Features:

- Contain theoretical and observational information. Claimed that observational information in a simulator is substantial [but this is in the eye of the beholder, and clearly not true at all in some cases (such as certain agent-based models)]. Emphasized that the simulators of interest for this paper are physics based.
- Typically depend on large number of input parameters.
 - Parameters are often estimated for sub-models, independent of the simulator. [Example would be calibrating EOS models in a shock wave physics code without calibrating the code itself. This is not what the author is thinking of] It is assumed that the uncertainty in sub-model parameterization [my phrase] is understood and defensible.
 - These sub-model calibrations and associated uncertainty distributions are called **prior** or **uncalibrated**. [Again, she is using this notion in a very specific sense – they have not undergone the calibration that she is discussing in this paper.]
- Typically output a large number of [potential] observables. She emphasizes that it is unlikely that there is observational data for everything that the simulator outputs, either in general or in specific simulations. Otherwise, what would be the need for the simulator? [This is too narrow a view of why we simulate; but it may be exactly relevant to her notion of calibration. My emphasis is how reliable all the outputs are. In computational physics, it is hard to argue that the outputs we care about are correct while the ones we don't can be incorrect, because of couplings and so on. Her emphasis is really on understanding what outputs calibration should be centered on, which is certainly unlikely to be everything a simulator can output.]

Section 2. Non-Iterative Paradigm for Uncertainty Quantification

The following is the clean and philosophically clear paradigm for quantification of total uncertainty in simulations:

#1 – Prior uncertainty: assembling prior information (all scenarios, input values, constraints on outputs, etc). Uncertainty described by (prior) pdfs. It is emphasized that this information is model independent. This is blurred when the inputs include numerical specifications and constraints, like meshing, “but even in these cases the allowable ranges may sometimes be specified using expert judgment, etc. more or less independently of the simulator.” [Note the

potential difficulties in how clean this uncertainty paradigm might be in reality. Note that the problem of specifying pdfs is not discussed.]

#2 – Uncertainty propagation: Map the joint prior pdf to the joint output pdf. This will typically be computationally difficult. If the simulator itself is non-deterministic [Monte Carlo radiation transport, for example] then the resulting output pdf will have variability due to this. [Epistemic uncertainty in the prior is not discussed, but this is dealt with readily in the performance assessment paradigm of Helton and colleagues for NRC, WIPP, and Yucca Mountain assessments. References TBD.]

#3 – Uncertainty quantification: Quantify the joint output distribution, typically for some purpose. For example, in the case of validation, we are interested in

$$y_{computed}(x_{test}) - y_{nature}(x_{test}) \tag{1}$$

We are interested in accuracy (bias) and precision (variance) of this comparison.

#4 – Prediction: Extrapolate to characterize the distribution of equation (1) at one or more new points $x_{prediction}$. [Note this prediction can be accomplished using formal statistical procedures (time series extrapolation, for example); the question is how much do we believe the prediction.]

A diagram of this process looks like:

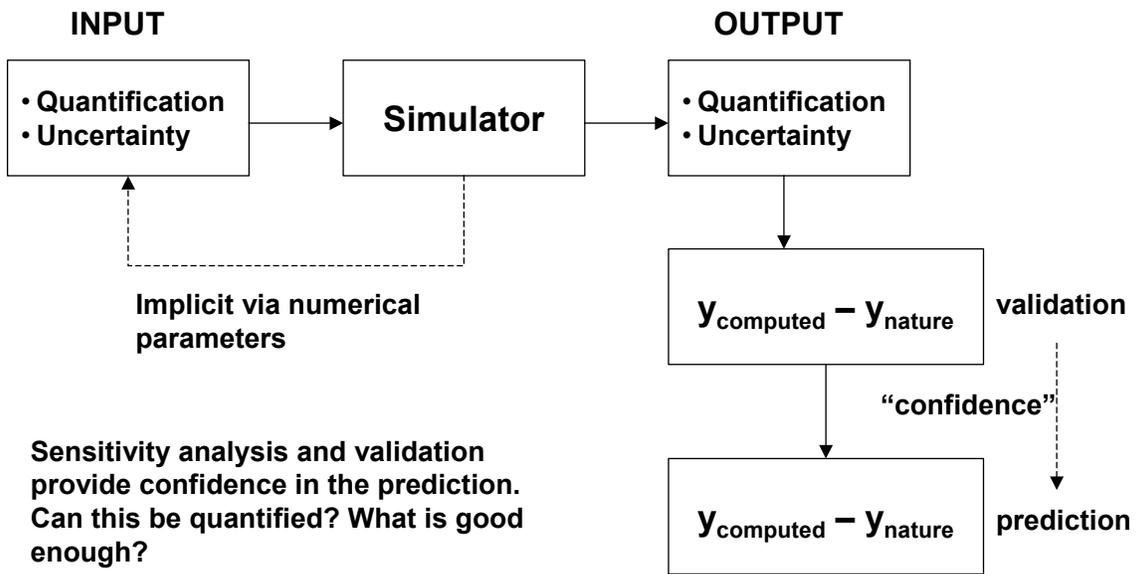


Figure 1. Uncertainty propagation and prediction.

M. C. Kennedy and A. O’Hagan “Bayesian Calibration of Computer Models.”

Journal of the Royal Statistical Society B, 2001. Volume 63, pp. 425-464.

- KOH outline the main types of uncertainty they wish to differentiate/identify:
- Parameter uncertainty
- Model Inadequacy
- Residual Variability
- Parametric Variability
- Observational Error
- Code Uncertainty

In practice, it will be difficult to isolate or separate parameter uncertainty (unknown inputs to a model) and parametric variability (when inputs require more detail than we are able to use, and so are left uncontrolled and unspecified in the model). Also, we foresee problems isolating residual variability (the variation in the process even when the conditions are fully specified – the stochastic nature of the process) and observation error. Finally, it may be difficult to isolate some of the aspects of model inadequacy (the difference between the “true” mean value of the real world process and the code output at the true value of the inputs) and code uncertainty (including cases where the program is wrong).

We need to understand the subtleties/differences in the approaches of Craig et al., Cox, Raftery and Poole, and Kennedy and O’Hagan.

The basic outline of the KOH approach is as follows:

Let $f(\mathbf{x})$ be a function mapping an input $\mathbf{x} \in X$ into an output $y=f(\mathbf{x})$ in \mathcal{R} . Usually we consider the input space a subset of \mathcal{R}^q : the vector $\mathbf{x} = (x_1, x_2, \dots, x_q)$. y is a scalar, $y \in \mathcal{R}$. A function $f(\mathbf{x})$ has a Gaussian process distribution if for every $n = 1, 2, 3, \dots$, the joint distribution of $f(\mathbf{x}_1) \dots f(\mathbf{x}_n)$ is multivariate normal for all $\mathbf{x}_1 \dots \mathbf{x}_n \in X$. In particular, $f(\mathbf{x})$ is normally distributed for all $\mathbf{x} \in X$.

LPS note: Why is it reasonable to think that a simulator output should be normally distributed? If we have an output that is clearly not normally distributed, we could take the mean of the simulator output as normally distributed according to the Central Limit Theorem. However, we need to think about how that affects the mean and covariance of the GP distribution. My concern is that if we are dividing the unnormalized mean and covariance by n or n^2 respectively to obtain a normal distribution, the prior (and posterior) distributions could be much narrower than are justified. Also, we need to understand the comment that a Gaussian process is nonparametric.

A standard representation of a GP is: $f(\mathbf{x}) \sim \mathcal{N}(m(\mathbf{x}), c(\mathbf{x}, \mathbf{x}'))$, where $c(\mathbf{x}, \mathbf{x}') = \text{cov}(f(\mathbf{x}), f(\mathbf{x}'))$. In the geostatistics method of kriging, the estimation of the covariance function (also called the semivariogram) is very important. KOH suggest a standard form for the

covariance matrix: $c(\mathbf{x}, \mathbf{x}') = \sigma^2 r(\mathbf{x} - \mathbf{x}')$, where r is a correlation function having the property $r(0) = 1$. This expression states that the covariance depends on a variance term σ^2 and a correlation term r that depends on the difference $\mathbf{x} - \mathbf{x}'$. The underlying assumption is that $f(\mathbf{x})$ and $f(\mathbf{x}')$ are similar if \mathbf{x} and \mathbf{x}' are sufficiently close in X . For nonlinear codes, this may not be true. Also, we need to understand what smoothness properties are required of f to have a covariance function like the one outlined in KOH.

KOH specify $r(\mathbf{x} - \mathbf{x}') = \exp\left\{\sum_{j=1}^q \omega_j (x_j - x'_j)^2\right\}$. This formulation requires that we specify

a prior on the variance term σ^2 and on the parameters ω_j .

The mean of the GP in the KOH formulation is given by $m(\mathbf{x}) = \mathbf{h}(\mathbf{x})^T \boldsymbol{\beta}$, where \mathbf{h} is a vector of p known functions over X , $\mathbf{h}(\mathbf{x}) = (h_1(\mathbf{x}), h_2(\mathbf{x}) \dots h_p(\mathbf{x}))^T$ and $\boldsymbol{\beta} = (\beta_1 \dots \beta_p)^T$ is a vector of p unknown coefficients which are given a prior distribution. KOH state that the multivariate normal is often used as a prior for $\boldsymbol{\beta}$, but then they state that “in practice information about hyperparameters such as $\boldsymbol{\beta}$ will typically be weak,” so they suggest using an improper uniform density $p(\boldsymbol{\beta}) \propto 1$. The $\mathbf{h}(\mathbf{x})$ vector describes a class of shapes and the belief that f may be approximated by a function. For example $\mathbf{h}(\mathbf{x}) = (1, x, \dots, x^{p-1})^T$ defines $m(x)$ to be a polynomial of degree $p-1$ when x is a scalar.

The important point to remember here is that we are trying to obtain a posterior estimate of the mean and covariance term of the GP: $f(\mathbf{x}) \sim N(m(\mathbf{x}), c(\mathbf{x}, \mathbf{x}'))$. These terms are scalar quantities (for example, the dot product of $\mathbf{h}(\mathbf{x})^T \boldsymbol{\beta}$ reduces to a single number). However, there are many assumptions and estimates of model form (of $\mathbf{h}(\mathbf{x})$ and $c(\mathbf{x}, \mathbf{x}')$) and of prior distributions that go into developing these estimates. We should be concerned about how sensitive our posterior distribution estimates are to various choices for these terms, but also we should be concerned for the amount that can get “lost in translation” with Bayesian models that are hierarchical and involve many terms. Is it better, for example, to generate many data points and get as close a functional form as possible for $\mathbf{h}(\mathbf{x})$? Or is it better to assume a simple model for $\mathbf{h}(\mathbf{x})$ such as $\mathbf{h}(\mathbf{x}) = 1$. This is what KOH actually do when they develop an example: they use $\mathbf{h}(\mathbf{x}) = 1$. This means that their prior on $\boldsymbol{\beta}$ represents the prior on the GP mean.

With the background of Gaussian processes outlines, KOH then develop their model for calibration. They assume that the calibration inputs are supposed to take fixed but unknown values $\boldsymbol{\theta} = (\theta_1 \dots \theta_{q_2})$. The output of the computer model when the variable inputs are given $\mathbf{x} = (x_1, x_2, \dots, x_{q_1})$ and when the calibration inputs are given values $\mathbf{t} = (t_1, t_2, \dots, t_{q_2})$ is denoted by $\eta(\mathbf{x}, \mathbf{t})$. KOH differentiate between the unknown value $\boldsymbol{\theta}$ of the calibration inputs which we wish to calibrate and a known particular set of their values, \mathbf{t} , which we set as inputs when running the model. They denote the “true” value of the real process when the variable inputs take value \mathbf{x} by $\zeta(\mathbf{x})$. The code outputs from N runs of the computer code are represented as $y_j = \eta(\mathbf{x}_j, \mathbf{t}_j)$. The observed data (consisting of n points, where $n < N$ usually) is denoted as $\mathbf{z} = (z_1, z_2, \dots, z_n)^T$. An observed data point z_i is an observation of the underlying process $\zeta(\mathbf{x}_i)$. In KOH’s formulation, they represent the relationship between the observations, the true process, and the computer model output by the equation:

$$z_i = \zeta(\mathbf{x}_i) + e_i = \rho \eta(\mathbf{x}_i, \mathbf{t}_i) + \delta(\mathbf{x}_i) + e_i$$

where e_i is the observation error for the i^{th} observation, ρ is an unknown regression parameter, and $\delta(\mathbf{x})$ is a model discrepancy or model inadequacy function that is independent of the code output $\eta(\mathbf{x}, \mathbf{t})$.

A few comments: this is a highly parameterized model, with both the code output $\eta(\mathbf{x}, \mathbf{t})$ and $\delta(\mathbf{x})$ represented as a Gaussian process. The error term e_i should include both residual variability as well as observation error, but KOH do not use replicated points and their model is deterministic, so they do not strictly address residual variability. They assume that e_i is normally distributed as $N(0, \lambda)$. The constant value of ρ implies that the underlying process $\zeta(\mathbf{x})$ is stationary.

The prior information about $\eta(\mathbf{x}, \mathbf{t})$ and $\delta(\mathbf{x})$ is given by Gaussian processes: $\eta(\mathbf{x}, \mathbf{t}) \sim N(m_1(\mathbf{x}, \mathbf{t}), c_1((\mathbf{x}, \mathbf{t}), (\mathbf{x}', \mathbf{t}'))))$ and $\delta(\mathbf{x}) \sim N(m_2(\mathbf{x}), c_2(\mathbf{x}, \mathbf{x}'))$. Using the hierarchical form on the means outlines above, we have $m_1(\mathbf{x}, \mathbf{t}) = \mathbf{h}_1(\mathbf{x}, \mathbf{t})^T \boldsymbol{\beta}_1$ and $m_2(\mathbf{x}) = \mathbf{h}_2(\mathbf{x})^T \boldsymbol{\beta}_2$. If a noninformative prior is assumed, $p(\boldsymbol{\beta}_1 \boldsymbol{\beta}_2) \propto 1$. KOH then formulate all of the hyperparameters relating to this problem. They denote (ρ, λ, ψ) by ϕ , where they state that ψ represents some “further hyperparameters” relating to the covariance functions. Then they assume that the prior distribution takes the form: $p(\boldsymbol{\theta}, \boldsymbol{\beta}, \phi) = p(\boldsymbol{\theta})p(\phi)$ because of the weak prior distribution on $\boldsymbol{\beta}$ and assumptions of independence.

In calculating the posterior, the important point is that the full data vector \mathbf{d} is normally distributed given $(\boldsymbol{\theta}, \boldsymbol{\beta}, \phi)$. The data vector \mathbf{d} is composed of two parts: the code output and the set of observations: $\mathbf{d}^T = (\mathbf{y}^T, \mathbf{z}^T)$. D_1 is the set of input points at which the code outputs y are available: $D_1 = \{(\mathbf{x}_1^*, \mathbf{t}_1), \dots, (\mathbf{x}_N^*, \mathbf{t}_N)\}$. D_2 is the set of inputs for which observations are available: $D_2 = (\mathbf{x}_1, \dots, \mathbf{x}_n)$. This next step seems pretty important to me, though it wasn't emphasized in the paper: the input points in D_2 are “augmented” with the true calibration parameters $\boldsymbol{\theta}$ to define $D_2(\boldsymbol{\theta}) = \{(\mathbf{x}_1, \boldsymbol{\theta}), \dots, (\mathbf{x}_n, \boldsymbol{\theta})\}$. This makes the observed set equivalent in dimension to a computer model set. If $\mathbf{H}_1(D_1)$ is the matrix with rows $\mathbf{h}_1(\mathbf{x}_1^*, \mathbf{t}_1)^T, \dots, \mathbf{h}_1(\mathbf{x}_N^*, \mathbf{t}_N)^T$, the expectation of \mathbf{y} is then $\mathbf{H}_1(D_1)\boldsymbol{\beta}_1$. The expectation of \mathbf{z} is: $\rho \mathbf{H}_1(D_2(\boldsymbol{\theta}))\boldsymbol{\beta}_1 + \mathbf{H}_2(D_2)\boldsymbol{\beta}_2$. The overall expectation for \mathbf{d} is:

$E(\mathbf{d} | \boldsymbol{\theta}, \boldsymbol{\beta}, \phi) = m_d(\boldsymbol{\theta}) = \mathbf{H}(\boldsymbol{\theta})\boldsymbol{\beta}$, where

$$\mathbf{H}(\boldsymbol{\theta}) = \begin{pmatrix} \mathbf{H}_1(D_1) & 0 \\ \rho \mathbf{H}_1(D_2(\boldsymbol{\theta})) & \mathbf{H}_2(D_2) \end{pmatrix}$$

The variance matrix of \mathbf{d} is defined as follows:

Define $\mathbf{V}_1(D_1)$ to be the matrix with the (j, j') element = $c_1((\mathbf{x}_j^*, \mathbf{t}_j), ((\mathbf{x}_{j'}^*, \mathbf{t}_{j'}))$. $\mathbf{V}_2(D_2)$ and $\mathbf{V}_1(D_2(\boldsymbol{\theta}))$ are defined similarly. Then let $\mathbf{C}_1\{D_1, D_2(\boldsymbol{\theta})\}$ be the matrix with (j, i) element $c_1((\mathbf{x}_j^*, \mathbf{t}_j), ((\mathbf{x}_i, \boldsymbol{\theta}))$. The overall variance matrix is given by:

$$\text{var}(\mathbf{d} | \boldsymbol{\theta}, \boldsymbol{\beta}, \phi) = \begin{pmatrix} \mathbf{V}_1(D_1) & \rho \mathbf{C}_1\{D_1, D_2(\boldsymbol{\theta})\}^T \\ \rho \mathbf{C}_1\{D_1, D_2(\boldsymbol{\theta})\} & \lambda \mathbf{I}_n + \rho^2 \mathbf{V}_1(D_2(\boldsymbol{\theta})) + \mathbf{V}_2(D_2) \end{pmatrix}$$

The full joint posterior distribution is then given by:

$p(\boldsymbol{\theta}, \boldsymbol{\beta}, \phi | \mathbf{d}) = p(\boldsymbol{\theta})p(\phi)f\{\mathbf{d}; \mathbf{m}_d(\boldsymbol{\theta}), \mathbf{V}_d(\boldsymbol{\theta})\}$, where $f\{\mathbf{d}; \mathbf{m}_d(\boldsymbol{\theta}), \mathbf{V}_d(\boldsymbol{\theta})\}$ is a normal $N(\mathbf{m}_d(\boldsymbol{\theta}), \mathbf{V}_d(\boldsymbol{\theta}))$ density function.

It is this expression for the posterior distribution that is impossible to calculate analytically. Even with simplification, it would require a high-dimensional quadrature to integrate $p(\boldsymbol{\theta}, \boldsymbol{\beta}, \phi | \mathbf{d})$ over $\boldsymbol{\beta}$ and ϕ to obtain the posterior estimate for the calibration parameters $p(\boldsymbol{\theta} | \mathbf{d})$. KOH basically say this is intractable, so they fix many of these parameters and use a two stage process, where they estimate the hyperparameters relating to the covariance matrix for the code term, c_1 , separately and before estimating the hyperparameters relating to the covariance matrix of the discrepancy term, c_2 .

The final theoretical section that KOH present relates to prediction. Usually, one would not just be interested in the updated calibration parameter estimates, but in using these to predict future realizations of the true process $\zeta(\mathbf{x})$. The posterior distribution of $\zeta(\mathbf{x})$ conditional on the estimated hyperparameters ϕ and the calibration parameters $\boldsymbol{\theta}$ is a Gaussian process, with an expectation given by:

$$E[\mathbf{z}(\mathbf{x}) | \boldsymbol{\theta}, \phi, \mathbf{d}] = \mathbf{h}(\mathbf{x}, \boldsymbol{\theta})^T \hat{\boldsymbol{\beta}}(\boldsymbol{\theta}) + \mathbf{t}(\mathbf{x}, \boldsymbol{\theta})^T \mathbf{V}_d(\boldsymbol{\theta})^{-1} \{\mathbf{d} - \mathbf{H}(\boldsymbol{\theta}) \hat{\boldsymbol{\beta}}(\boldsymbol{\theta})\}$$

KOH specify what the component terms in this equation are in their paper. Clearly, there is some component relating to the original $\mathbf{H}(\boldsymbol{\theta})\boldsymbol{\beta}$ in the first expression to the right of the equal sign, with a correction factor in the second expression.

KOH have a section on computational issues where they focus on two problems: the first is that the inverse of the variance matrix $\mathbf{V}_d(\boldsymbol{\theta})$ needs to be calculated to calculate the posterior distribution. If the size of this matrix, $(N+n)*(N+n)$, is very large, this can be a difficult problem. In practice with very complex codes, we do not foresee that we would be dealing with more than a few hundred observed points and a few hundred simulations, which is tractable with today's linear algebra packages. The bigger issue is how to do the integration necessary to obtain the posterior expectation, for example. KOH say that they use a Gauss-Hermite quadrature method, but they admit that this only works for low-dimensional problems, and so in practice they suggest that it may be necessary to use simulation methods of integration, such as MCMC. A useful exercise for us would be to outline what terms in the calculation of the posterior would involve actual data points (from the code runs or the observations) and what terms would need to be generated from a simulation.

The radionuclide deposition example that KOH present is useful, but it leaves out some details of the calculations. The authors do make many simplifications of their overall approach. The most important one is that they leave out the code uncertainty entirely, saying that “we have treated the plume code as a known function.” It would be helpful to know what this means in terms of the GP terms for $\eta(\mathbf{x}, \mathbf{t})$: they state that they do not have to specify a correlation matrix, but they don't say if they assume a constant mean. But since many of the difficulties of the calculations involve the $\mathbf{m}_1(\mathbf{x}, \mathbf{t}) = \mathbf{h}_1(\mathbf{x}, \mathbf{t})^T \boldsymbol{\beta}_1$ and the associated covariance terms, it would be helpful to see the resulting equations that they did use. Another confusing thing was how they aggregated the original 695

observed data points: they state that they formed subsets of 10, 15, 20, and 25 points. We are assuming that subset averages were taken on the subsets and used as the observed data. Finally, this example has the characteristic that the code runs were performed for the same conditions/locations as the observed input data. This may not always be feasible with real problems: we could have code runs that don't correspond exactly to observed data.

KOH do show that their calibration approach (strategy 2) is much better than a GP interpolation of the observed data alone (strategy 1) or a typical least squared regression model (strategy 3). We wish that they had shown the prior and posterior distribution of θ for each case, and not only the residuals of the prediction error.

D. D. Cox, J-S. Park, and C. E. Singer. “A statistical method for tuning a computer code to a data base.”

Computational Statistics and Data Analysis 37(2001), pp. 77-92.

Overall, there are many similarities between this paper and the Kennedy/O’Hagan paper. However, we found this one more readable. The examples provided were helpful, but we found them weak on implementation details.

We like the goal statement of Cox et al.: “to develop a statistical procedure for efficiently utilizing the computer model in conjunction with the data base to obtain good (model) parameter estimates.” The authors claim that their methodology allows one to “combine information from a limited number of noisy executions of the computer code with the information in the data base to make inferences about the unknown parameters.” Thus, they acknowledge that there may be few experimental runs AND few computational runs, and that the computer model involves a Monte Carlo procedure (which KOH do not, at least in the form they presented.).

The approach presented in this paper is similar to KOH in that they assume a Gaussian process distribution and have a covariance structure which leads to a complicated variance matrix for the full data set. However, the two main (and important) differences are:

- The calibration parameters are estimated via a Maximum likelihood method instead of Bayesian updating.
- There is no “model discrepancy” term. The results from the experimental model are basically treated the same as the results from the computational model.

With these differences in mind, we summarize the technical details of the paper:

The authors present the same statistical model for both the experimental and the computational data. The model is:

$y_{iE} = g(\mathbf{x}_{iE}, \mathbf{c}_E) + \varepsilon_{iE}$ where there are n_E observations of experimental data. Each observation is given by $(\mathbf{x}_{iE}, y_{iE})$ where \mathbf{x}_{iE} is a vector of values of the independent variables and y_{iE} is the response. $g(\mathbf{x}, \mathbf{c}_E)$ is a regression function where \mathbf{c}_E is the unknown vector of regression parameters. In classical statistics, the optimal approach to estimating \mathbf{c}_E is to find the values of \mathbf{c} which minimize the residual sum of squares (the sum squared error terms $[y_{iE} - g(\mathbf{x}_{iE}, \mathbf{c}_E)]$).

For the computational model, the response is modeled in a similar way:

$y_{iC} = g(\mathbf{x}_{iC}, \mathbf{c}_{iC}) + \varepsilon_{iC}$, where the subscript C denotes the computer model instead of the experimental data which is denoted by subscript E. The main different here is that $g(\mathbf{x}_{iC}, \mathbf{c}_{iC})$ involves running an expensive computer code. The other difference, which the authors don’t emphasize but which is important is that the calibration terms in this equation are indexed by i: there is not just “one” optimal setting for the calibration terms as there is with the experimental data, but rather an optimal setting for each input set. In

practice, we may also want to do something like this but for initial problems, we would probably want to determine just one set of optimal parameters.

Cox et al. explain the function evaluations necessary to perform a nonlinear least squares optimization to determine the optimal settings of the parameters: on the order of $n_E * M$, where M is the number of function evaluations required for a numerical optimization routine to find the optimal c -dimensional vector. In practice, $n_E * M$ could be on the order of 100,000, which is way too large for an expensive computer code. Thus, the authors propose their method as an “approximate nonlinear least squares” technique, or ANLS.

The approximation they use for $g(\mathbf{x}, \mathbf{c})$ is assuming that g is a realization of a Gaussian process, which they denote by $Y(\mathbf{t})$. The mean and variance of this GP are:

$$\begin{aligned} E[Y(\mathbf{t})] &= \mu(\mathbf{t}) \\ \text{Cov}(Y(\mathbf{s}), Y(\mathbf{t})) &= K(\mathbf{s}, \mathbf{t}) \end{aligned}$$

Note that \mathbf{s} and \mathbf{t} are vectors composed of both \mathbf{x} and \mathbf{c} (this is a way of combining inputs): $\mathbf{t} = (\mathbf{x}, \mathbf{c})$. \mathbf{t} can represent both types of data: experimental data or computer data: $\mathbf{t}_{iE}(\mathbf{c}) = (\mathbf{x}_{iE}, \mathbf{c})$ and $\mathbf{t}_{iC} = (\mathbf{x}_{iC}, \mathbf{c}_{iC})$. Note that the experimental data assumes a fixed value of \mathbf{c} .

The mean function for this GP is expressed as a linear model: $\mu(\mathbf{t}) = \beta_0 + \sum_{j=1}^k \beta_j t_j$.

This is simpler than the KOH model, where they express the mean function as $\mathbf{h}(\mathbf{x})^T \boldsymbol{\beta}$. The $\boldsymbol{\beta}$ vector is unknown and needs to be estimated. The covariance function is taken from spatial statistics and is basically the same as in KOH:

$$K(\mathbf{s}, \mathbf{t}) = \sigma_y^2 \exp\{-\theta \sum_{j=1}^k (s_j - t_j)^2\}, \text{ where } \sigma_y^2 \text{ and } \theta \text{ also need to be estimated.}$$

The formulation of the following terms is almost exactly the same as KOH. The joint normal density of the combined computer observations and experimental data set $\mathbf{y} = (\mathbf{y}_C, \mathbf{y}_E)$ is a multivariate normal: $f(\mathbf{y} | \mathbf{c}_E, \sigma_E^2, \sigma_C^2, \sigma_Y^2, \beta, \theta)$.

The mean of \mathbf{y} is $\nu(\beta, \mathbf{c}_E) = \begin{bmatrix} \nu_C(\beta) \\ \nu_E(\beta, \mathbf{c}_E) \end{bmatrix}$, where each individual i^{th} component of the top

term is given by: $\nu_{iC}(\beta) = \mu(t_{iC})$ for $1 \leq i \leq n_c$ and each i^{th} component of the lower term is given by $\nu_{iE}(\beta, \mathbf{c}_E) = \mu(t_{iE}(\mathbf{c}_E))$ for $1 \leq i \leq n_E$.

The covariance matrix of \mathbf{y} is:

$$V(\theta, \sigma_E^2, \sigma_C^2, \sigma_Y^2, \mathbf{c}_E) = \begin{bmatrix} V_{CC}(\theta, \sigma_C^2, \sigma_Y^2) & V_{CE}(\theta, \sigma_Y^2, \mathbf{c}_E) \\ V_{CE}(\theta, \sigma_Y^2, \mathbf{c}_E)^t & V_{EE}(\theta, \sigma_E^2, \sigma_Y^2, \mathbf{c}_E) \end{bmatrix}$$

The block entries of this matrix are given in further detail in Equation (10) in the paper.

Each term in this covariance matrix involves some combination of individual covariance terms from the $K(\mathbf{s}, \mathbf{t})$ covariance function defined above: the upper left term involves covariances between the computer model observations, the lower right term between the experimental data points, etc.

Cox et al. then calculate the conditional distribution of the experimental data given the computer data. This is a normal distribution with mean:

$v_{E|C} = E[y_E | \mathbf{y}_C, \mathbf{c}_E, \sigma_E^2, \sigma_C^2, \sigma_Y^2, \beta, \theta] = v_E(\beta, \mathbf{c}_E) + V_{CE}' V_{CC}^{-1} (y_c - v_c)$. Also needed in the approach is the marginal density for the computer data y , given by $f(\mathbf{y}_C | \sigma_C^2, \sigma_Y^2, \beta, \theta)$.

The approach for the approximate nonlinear least squares estimation is then:

If the parameters $\sigma_C^2, \sigma_Y^2, \beta, \theta$ are known (LPS note: what assumed values do we start with?), then for a given value of \mathbf{c}_E the prediction of \mathbf{y}_E given the computer data can be calculated according to the conditional expectation formula given above. This prediction is taken to be the conditional expectation $v_{E|C} = Y(\mathbf{t}_E(\mathbf{c}))$.

Estimates of the parameters $\sigma_C^2, \sigma_Y^2, \beta, \theta$ are calculated by maximizing the marginal computer likelihood $f(\mathbf{y}_C | \sigma_C^2, \sigma_Y^2, \beta, \theta)$. This is a multinormal distribution. Cox et al. outline a few ways to do this maximum likelihood estimation.

Once they get “optimal” MLE estimates of the parameters $\sigma_C^2, \sigma_Y^2, \beta, \theta$, these are plugged back into the conditional expectation formula and an estimate for the regression function of the experimental data is given by the right hand side of this formula:
 $\hat{g}(\mathbf{t}_E(\mathbf{c})) = v_E(\beta, \mathbf{c}_E) + V_{CE}' V_{CC}^{-1} (y_c - v_c)$.

The ANLS method then minimizes the approximate residual sum of squares with respect to \mathbf{c} : $ARSS(c) = \sum_{i=1}^{n_E} [y_{iE} - \hat{g}(\mathbf{t}_E(\mathbf{c}))]^2 = \sum_{i=1}^{n_E} [y_{iE} - \hat{g}(\mathbf{x}_{iE}, \mathbf{c})]^2$, and this value of \mathbf{c} is considered the optimal calibration.

The authors then discuss how they estimate the optimal values of the parameters in step (2). The first way is to optimize the parameters over the full data set, not just the computer observations. Thus, they maximize the likelihood: $f(\mathbf{y} | \mathbf{c}_E, \sigma_E^2, \sigma_C^2, \sigma_Y^2, \beta, \theta)$.

They call this FMLE for Full MLE. They also have a method where they separate out some of the parameters which are estimated from the computer data only. The separated MLE approach (SMLE) estimates $\sigma_C^2, \sigma_Y^2, \beta, \theta$ from the marginal computer likelihood $f(\mathbf{y}_C | \sigma_C^2, \sigma_Y^2, \beta, \theta)$. Then these parameters are plugged back into the conditional likelihood estimates for both the mean and variance of the experimental data, and these are used to get MLE estimates for σ_E^2 and \mathbf{c}_E . This is basically the same as the ANLS method, except there is a different objective function used to obtain the optimal \mathbf{c} values:

$$\det(V_{E|C}) + (y_E - v_{E|C}) V_{E|C}^{-1} (y_E - v_{E|C})$$

Finally, the authors have a “partial” MLE (PMLE), which is in between the separated and full MLE.

There are some important points to remember here: the calculations of the MLE will not be trivial (although better than Bayesian estimates!) A standard way to calculate MLE is

to minimize the negative log of the likelihood function, and it looks like this is what Cox et al. are doing. It does not look like they are using an optimization function per se, but some theoretical results derived for multivariate normal distributions which state that the MLE estimate of a parameter vector has an approximately normal distribution with mean equal to the true parameter vector (?) and a covariance matrix equal to the inverse of the Hessian of the negative log of the likelihood evaluated at the maximum. With the linear algebra solvers available at SNL, we do not anticipate that it would be difficult to solve such for the inverse Hessian even for large data sets. However, we need to understand the MLE method they are using in more detail.

After outlining their approach, Cox et al. demonstrate it on five “toy model” simulations. These are nice in that one can easily see simple exponential functions with all of the parameters of interest. They showed the results for ANLS, SMLE, PLME, FMLE, and a regular NLS regression. They reported their estimates for all the parameters, and the absolute difference between the “true” c_E and the estimated c_E . Overall, the PLME method performs the best on this test set, both in terms of minimizing the difference between the estimated and true parameters and in having confidence intervals that “cover” the true parameter estimate the majority of the time.

The authors also briefly discuss an application to the Tokamak data set. They have data from two machines, so the results are somewhat confounded by this. They reported their estimates of four calibration parameters in the code, but it was hard to judge how useful these calibrations were: the authors themselves say that the estimates are not terribly accurate because of the large confidence intervals. They did have 74 experimental data points and 64 computer runs, which is comparable to what we might have.

Overall, we think this is a good approach. It would be very useful to implement a MLE approach and understand the multivariate density functions, the covariance matrices, etc. before we go to the Bayesian formulation.

**The Design and Analysis of Computer Experiments, by
Santner, Williams, Notz.
Springer Series in Statistics, 2003.**

The most useful information we obtained on Bayesian Design of Experiments was found in the book *The Design and Analysis of Computer Experiments*, by Santner, Williams, Notz, Springer Series in Statistics, 2003. This book and associated papers seemed to cover the most important issues: optimization of control parameters under the influence of environmental factors (uncontrollable variables), experimental design using a hierarchy of models (low to high fidelity), competing objectives, robust optimization, space-filling experimental designs, surrogate frameworks, and even a little validation.

Non Bayesian Approach

This section briefly reviews some pragmatic engineering issues associated with the non-Bayesian Taguchi approach.

(Traditional) Taguchi Approach:

The first step is a 2-level screening experiment typically with a large number of control and environmental variables (as possible) each at only two levels. This experiment typically ignores all interactions. Expert knowledge and uncertainty are introduced via the initial values and ranges of these experimental variables. The goal is to identify the most important variables. This identification is pragmatic (such as the Pareto 80/20 rule of thumb) rather than by statistical significance. An experiment at the predicted maximum (minimum or target level as appropriate) is done to provide some confidence that all variables were identified and things are approximately linear over the restricted experimental domain. (If not, more screening experiments are done to uncover additional variables, to reduce the initial domain or to determine interactions.)

The least important variables are then confounded in order retain their influence on the remaining experiments, but not at the expense of too many experimental runs. The less important control variables are confounded together as one or a small number of pseudo variables (our terminology). The environmental variables are confounded together similarly if necessary (Taguchi's outer array). The high setting (level) for a pseudo variable is defined by setting the confounded variables individually to the level the maximized the objective. The low setting sets the confounded variables to their opposite levels.

Dr. Taguchi suggested that variables that directly influence the system energy are the most likely candidates for possible interactions. There is also a hint toward assuming only two-way interactions until experiments prove otherwise. Every screening experiment is followed by an experiment at the predicted maximum. If the actual values disagree from the prediction, then continued screening experiments are preformed to detect interactions or to provide clues to missing experimental variables. When there is reasonably close agreement between the predicted and validated maximum, this design

point becomes the origin for the next experiment via an informal hill climbing “pattern search”. If the magnitude of one of the pseudo variables shifts then more screening experiments are performed to discover the significant experimental variables.

Later (non screening) experiments are conducted at many levels (3-7 typically) for the non-confounded variables in order to measure nonlinearity and to zoom in on a solution. The confounded variables and environmental variables generally continue to have two levels as a simple check that the “ignored” variables haven’t become significant. These experimental designs have non-confounded interaction columns to identify (generally two-way) interactions.

Dr. Taguchi also advocated robust optimization which was achieved by using the “signal to noise” ratio of the objective function. This creates an additional level of confounding but frequently finds a solution which is relatively insensitive to variation of the control variables.

Some of these Taguchi engineering heuristics could be used in the Bayesian approach. For example, environmental variables should be subject to screening experiments to determine factor sensitivity. The influence of the environmental variables could be confounded into a few types to reduce the number of model evaluations.

Most other experimental design methodologies (other than Taguchi and Bayesian) focus on strategies to determine design points that sample the entire experimental domain. Orthogonal arrays, Latin Hypercube sampling, and distance metrics are popular for determining the design points. The experimental results are used to create a surface response (generally limited to linear and quadratic terms). The surface response can be used for both traditional optimization and for robust optimization. These techniques are also used in the Bayesian experimental design.

Note that the original research on experimental designs focused on agricultural experiments (hence the current terminology: blocks, plots, treatments, etc.) Because anything forgotten had to wait a year (next growing season), these experimental designs tried to do everything at once. Hence they determine significant factors and interactions simultaneously. Modern approaches attempt to minimize the number of experiments (since they may be costly) by using sequential methods which augment the existing design points based on the past results. They look to get more data near regions containing optima and “neglected” regions. This sequential design approach works with Bayesian methods.

Bayesian Approach to Design of Experiments

This section summarizes parts of the book *The Design and Analysis of Computer Experiments*, by Santner, Williams, and Notz. This book defines concepts and background material on Bayesian experimental design. The book seems to primarily focus on the theory of experimental design for the case of control variables (needing optimization, possibly robust optimization) under the constraint of environmental variables. The authors use the notation \mathbf{x}_c to denote the vector of control variable

settings, \mathbf{x}_e the vector of environmental settings, \mathbf{X}_e denotes treating the environmental variables as a random variable, and $y(\cdot)$ denotes the output (response). Thus $y(\mathbf{x}_c, \mathbf{X}_e)$ is a random variable (response) with a distribution induced by the distribution of \mathbf{X}_e .

The mean response is computed over the expected distribution of the environmental variables for a given vector of control variables for each optimization step of the control variables.

$$\mu(\mathbf{x}_c) = E\{y(\mathbf{x}_c, \mathbf{X}_e)\}$$

From here, a number of robust designs are defined. For example, \mathbf{x}_c^π is $\pi(\cdot)$ robust provided

$$\int \mu(\mathbf{x}_c^\pi, \boldsymbol{\theta})\pi(\boldsymbol{\theta})d\boldsymbol{\theta} = \max_{\mathbf{x}_c} \int \mu(\mathbf{x}_c, \boldsymbol{\theta})\pi(\boldsymbol{\theta})d\boldsymbol{\theta} \quad (\text{Page 21})$$

where $\pi(\cdot)$ is a prior density over the $\boldsymbol{\theta}$ values. M-robust, V-robust, and G-robust designs are also discussed.

One concern is how to calculate μ . If we use a MCMC method to sample the distribution of the environmental variables, it may lead to a significant number of model evaluations. Taguchi might suggest confounding the least important environmental variables, but this implies “confounding” their distributions.

Section 2.3.1 of [Santner et al.] contains a philosophical discussion of the difference between the frequentist and Bayesian approaches. The following quote defines their approach to this book (their italics):

“In sum, our attitude toward using the Bayesian approach to problems of the design and analysis of computer experiments is not dogmatic. We *do* attempt to control the characteristics of the functions produced by our priors, but *do not* rigidly believe them. Instead, our goal is to choose flexible priors that are capable of producing many shapes for $y(\cdot)$ and then let the Bayesian machinery allow the data to direct the details of the prediction process.”

Bayesian experimental design uses a response function similar to a linear regression model, but called a Gaussian random function (GRF) model:

$$Y(\mathbf{x}) = \sum_{j=1}^p f_j(\mathbf{x})\beta_j + Z(\mathbf{x}) = \mathbf{f}^T(\mathbf{x})\boldsymbol{\beta} + Z(\mathbf{x}) \quad (*)$$

In the Gaussian random function model, $f_i(\cdot)$ are known regression functions, $\boldsymbol{\beta}$ is a vector of unknown regression coefficients, and $Z(\cdot)$ is a stationary Gaussian process having zero mean, variance σ_Z^2 , and correlation function $R(\cdot)$. Gaussian processes are defined more completely in the section below. The “Bayesian” part of the experimental design involves specifying priors on parameters in $f_i(\cdot)$, $\boldsymbol{\beta}$, and $Z(\cdot)$, and updating these priors with the data to obtain posteriors. There are many prediction methods based on the Gaussian random function model.

One use of Bayesian experimental design is in the design of experiments on a computer code at multiple levels of fidelity. This is becoming a big area of research interest for us at Sandia. In this approach, an autoregressive model is formulated as:

$$Y_i(\mathbf{x}) = \rho_{i-1}Y_{i-1}(\mathbf{x}) + \delta_i(\mathbf{x}), \quad i = 2, \dots, m$$

where Y_1 is the result from the least complex code (lowest fidelity) and higher Y_i are results from more complex codes (successively greater fidelity). Here the $Y_i(\mathbf{x})$ denotes the prior for the i^{th} code level, m is the highest level, $\delta_i(\cdot)$ is a process independent of $Y_1(\cdot), \dots, Y_{i-1}(\cdot)$. Under the assumption that $Y_i(\cdot)$ is stationary for each i , this model is implied by:

$$\text{Cov}\{Y_i(\mathbf{x}), Y_{i-1}(\mathbf{w}) \mid Y_{i-1}(\mathbf{x})\} = 0, \quad \text{for all } \mathbf{w} \neq \mathbf{x}.$$

The authors state: “which means that no additional second-order knowledge of code i at \mathbf{x} can be obtained from the lower-level code $i-1$ at $\mathbf{w} \neq \mathbf{x}$ if the value of code $i-1$ at \mathbf{x} is known. Therefore, the spatial autoregressive model can be interpreted as imposing a Markov property on the hierarchy of codes.”

Later in the example, the authors write:

“...suppose that $y_p(\cdot)$ represents the output from fast, but *poorer*, code and $y_g(\cdot)$ the output from the slow, but *good*, code. *Our goal is to predict the output of the good code at input site \mathbf{x}_0 , i.e., to predict $y_g(\mathbf{x}_0)$, based on $y_p(\mathbf{x}_1^p), \dots, y_p(\mathbf{x}_{n_p}^p)$ and $y_g(\mathbf{x}_1^g), \dots, y_g(\mathbf{x}_{n_g}^g)$.*”

$$\begin{aligned} Y_p(\mathbf{x}) &= \mathbf{f}_p^T(\mathbf{x})\boldsymbol{\beta}_p + W_p(\mathbf{x}), \\ Y_g(\mathbf{x}) &= \mathbf{f}_g^T(\mathbf{x})\boldsymbol{\beta}_g + \rho W_p(\mathbf{x}) + W_a(\mathbf{x}) \end{aligned}$$

Here “the regression function $\mathbf{f}_p^T(\cdot)$ specifies the large-scale, nonstationary structure of $y_p(\cdot)$ and $W_p(\cdot)$ is a stationary Gaussian process that determines the local features of the code; $W_p(\cdot)$ is assumed to have zero mean, variance σ_p^2 , and correlation function $R_p(\cdot)$.” $Y_g(\mathbf{x})$ is the prior for the more accurate code.

Obtaining sequential design points

This section is a summary of “Sequential Design of Computer Experiments to Minimize Integrated Response Functions”, Williams, Santner, Notz, *Statistics Sinica* 10(2000)

Quoting their abstract: “We introduce a sequential experimental design for finding the optimum of $l(\mathbf{x}_c) = E\{y(\mathbf{x}_c), \mathbf{X}_c\}$, where the expectation is taken of over the distribution of the environmental variables. The approach is Bayesian; the prior information is that $y(\mathbf{x})$ is a draw from a stationary Gaussian stochastic process with a correlation function from the Matern class having unknown parameters. The idea of the method is to compute the posterior expected “improvement” over the current optimum for each untested site; the design selects the next site to maximize the expected improvement.”

The algorithm in brief:

- S0: Chose an initial set of design points using a space-filling criterion. (They use maximin distance design selection from a set of Latin hypercube sampling designs.)
- S1: Estimate the correlation parameter vector by the maximizer of the posterior density given the vector of responses.
- S2: Chose the $(n+1)^{\text{st}}$ control variable site to maximize the posterior expected improvement given the current data
- S3: Choose the environmental variable site corresponding to this new control site (above) to minimize the posterior mean square prediction error given the current data.
- S3: Determine if the algorithm should be halted. If not, the new control site is added to the set of design points and the algorithm returns to step S1.

DISTRIBUTION

MS0370 M. S. Eldred, 9211
MS0847 K. J. Dowding, 9133
MS0847 A. A. Giunta, 9133
MS0770 J.C. Helton, 9133
MS1110 B. A. Hendrickson, 9215
MS0370 S. A. Mitchell, 9211
MS0828 W. L. Oberkampf, 9133
MS0828 M. Pilch, 9133
MS0828 V. J. Romero, 9133
MS0370 L. P. Swiler, 9211 [5]
MS0370 T. G. Trucano, 9211 [2]

R. G. Hills, New Mexico State University

MS9018 Central Technical Files, 8945-1
MS0899 Technical Library, 9616 [2]
MS0612 Review and Approval Desk, 9612 for DOE/OSTI