SANDIA REPORT
SAND2005-3294
Unlimited Release
Printed April 2006

Bayesian Methods in Engineering Design Problems

Laura P. Swiler

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd.
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND 2005-3294
Unlimited Release
Printed April, 2006

Bayesian Methods in Engineering Design Problems

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

Abstract

This report discusses the applicability of Bayesian methods to engineering design problems. The attraction of Bayesian methods lies in their ability to integrate observed data and prior knowledge to form a posterior distribution estimate of a quantity of interest. Conceptually, Bayesian methods are desirable because they have the property of taking prior estimates and updating them with data over time. Bayesian statistics has been dominated by non-Bayesian approaches to inference for many years. However, over the past decade, there has been an emergence of Bayesian methods, driven by the availability of computational techniques.

This report outlines Bayesian approaches which could be applied to engineering problems, particularly design optimization problems. This report first outlines the fundamental principles of Bayesian statistics. We discuss some simple applications of Bayesian inference, and then present more complex applications of Bayesian techniques applied to problems of calibration, optimization under uncertainty (OUU), and verification and validation (V&V). Specific applications of Bayesian methods to engineering problems include probability of failure estimation, Bayesian regression, Gaussian process models, and hierarchical or multi-fidelity models.

Page Left Blank

Table of Contents

Bayesian Methods in Engineering Design Problems

1. Introduction

This report discusses the applicability of Bayesian methods to engineering design problems. The attraction of Bayesian methods lies in their ability to integrate observed data and prior knowledge to form a posterior distribution estimate of a quantity of interest. Conceptually, Bayesian methods are desirable because they have the property of taking prior estimates and updating them with data over time. Bayesian statistics has been dominated by non-Bayesian approaches to inference for many years. However, over the past decade, there has been an emergence of Bayesian methods, driven by the availability of computational techniques.

This report was prepared as part of a Laboratory Directed Research and Development (LDRD) project at Sandia National Laboratories called PRIDE, Penetrator Reliability Investigation and Design Exploration. As part of this LDRD, our charter was to critically examine the Bayesian framework and identify Bayesian approaches which could be applied to engineering problems, particularly design optimization problems. This report first outlines the fundamental principles of Bayesian statistics. We discuss some simple applications of Bayesian inference, and then present more complex applications of Bayesian techniques applied to problems of calibration, OUU, and V&V at Sandia National Laboratories.

A recent text on Bayesian analysis [Gelman et al., 2004] states: "In general, work in Bayesian statistics now focuses on applications, computations, and models. Philosophical debates, abstract optimality criteria, and asymptotic analyses are fading into the background. It is now possible to do serious applied work in Bayesian inference without the need to debate the fundamental principles of inference. … Given the conceptual simplicity of the Bayesian approach, it is only in the intricacy of specific applications that the novelty arises." We agree with this statement. We endorse the conceptual framework of Bayesian analysis. We have found, however, that implementing a Bayesian approach is non-trivial except in very few cases. The current Markov Chain Monte Carlo (MCMC) sampling methods that are used to generate posterior distributions are computationally expensive and convergence of the chain can be difficult to implement and assess. We address the usefulness of Bayesian analysis for engineering design problems. The current thinking is to emulate simulation models with cheaper response functions such as regression models or Gaussian process models. The parameters that govern these surrogate models (called hyperparameters) are then modeled in a Bayesian framework, and posterior distributions of the hyperparameters lead to posterior ensembles of response surface models. This approach can be applied to engineering design problems, and may be especially applicable to multi-fidelity problems.

The outline of this report is as follows: Section 2 covers fundamentals of Bayesian inference, including hypothesis testing, conjugate pairs, MCMC methods used to generate posterior distributions, etc. Section 3 discusses some simple applications of Bayesian methods to engineering problems (e.g., probability of failure estimation, Bayesian regression), and Section 4 discusses some applications to more complex problems such as OUU, calibration, and hierarchical models. Section 5 is a conclusion with thoughts about future directions.

2. Fundamentals of Bayesian Inference

2.1. Discrete case

Bayes' rule relates the posterior density of a parameter to the likelihood function and the prior density of that parameter. In the discrete case, the Bayesian formulation for the posterior probability density function *h* is:

$$h(\theta \mid \mathbf{x}_1,...,\mathbf{x}_N) = \frac{f(\mathbf{x}_1 \mid \theta)...f(\mathbf{x}_N \mid \theta)g(\theta)}{\sum_\theta f(\mathbf{x}_1 \mid \theta)...f(\mathbf{x}_N \mid \theta)g(\theta)} \qquad (1)$$

where $\mathbf{x}_1,\ldots\mathbf{x}_N$ 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 data x_i's and not on θ; the denominator is a normalizing constant. Thus, Bayes formula is often written as:

$$h(\theta \mid \mathbf{x}_1,...,\mathbf{x}_N) \propto L(\mathbf{x}_1,...,\mathbf{x}_N \mid \theta)g(\theta) \qquad (2)$$

where $L(\mathbf{x}_1,\ldots\mathbf{x}_N|\theta) = f(\mathbf{x}_1|\theta) *\ldots* f(\mathbf{x}_N|\theta)$ = 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.

2.2. 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 $L(\mathbf{x}_1,\ldots\mathbf{x}_N|\theta)$ instead of the conditional probability density functions $f(\mathbf{x}_i|\theta)$. So, one way of expressing Bayes' Theorem in the continuous case is:

$$h(\theta \mid \mathbf{x}_1,...,\mathbf{x}_N) = \frac{L(\mathbf{x}_1,...,\mathbf{x}_N \mid \theta)g(\theta)}{\int L(\mathbf{x}_1,...,\mathbf{x}_N \mid \theta)g(\theta)d\theta} \qquad (3)$$

2.3. 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, $T\equiv T(\mathbf{x}_1,\ldots\mathbf{x}_N)$. Then Bayes' Theorem states that

$$p(H \mid T) = \frac{p(T \mid H)p(H)}{p(T \mid H)p(H) + p(T \mid A)p(A)} \qquad (4)$$

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:

$$p(A \mid T) = \frac{p(T \mid A)\,p(A)}{p(T \mid H)\,p(H) + p(T \mid A)\,p(A)} \tag{5}$$

Taking the ratio of 4 and 5, we have:

$$\frac{p(H \mid T)}{p(A \mid T)} = \left[\frac{p(H)}{p(A)}\right]\left[\frac{p(T \mid H)}{p(T \mid A)}\right] \tag{6}$$

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 $\varepsilon>0$, 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. Also, in the Bayesian approach, the two hypotheses are treated symmetrically so you can find evidence in favor of the null, which you can't do with classical methods.

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]

2.4. 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. 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 advantage of the Bayesian approach is that one gets an entire (posterior) distribution estimate on a parameter as part of the inference procedure, not just a point estimate on a parameter. For complex models, it can be quite difficult to get good uncertainty estimates of parameters under a classical approach.

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. In the "Empirical Bayes" approach, one estimates 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 can result in incoherent estimators. In most mainstream Bayesian approaches now, people use MCMC methods to estimate the parameters.

There are deep philosophical differences and ongoing debates between statisticians who consider themselves frequentists and those who are Bayesian. This report does not address these differences, nor does it address the many subtleties associated with a subjectivist interpretation of probability. Interested readers are encouraged to look at [Jaynes, 2003] and [Lindley, 1965] for more background. Another good reference which discusses pitfalls of Bayesian methods, specifically when used with risk assessment, is found in [Ferson, 2005]. Ferson states that the use of Bayesian methods in risk assessment sometimes produces overconfidence and arbitrariness in the computed answers, due to misuse of equiprobability for incertitude, overuse of averaging to aggregate information, and reliance on precise values and particular distributions when the available information does not justify such specificity. We generally agree with this criticism, but do not address these limitations in this report since we focus on how one could use Bayesian methods for particular problems in engineering design under uncertainty.

2.5. Simple Example of Bayesian Inference

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 signal trial, then the probability mass function for x is:

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

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:

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

Suppose n=5 and x = 2. Then $h(0.3|x)=0.13$ and $h(0.6|x)=0.87$. 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:

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

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:

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

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:

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

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 distributions come from the same family of distributions, is called a "conjugate prior" or a conjugate pair. More examples of conjugate priors are listed in Table 1.

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.

2.6. 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

2.7. 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 [Gilks et al., 1996; 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 $\{X_0, X_1, X_2, \ldots\}$ 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:

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

Determining n and M is not trivial. In practice, people often ignore the first 1000 samples. Another commonly used technique is to plot the chain (the simulated distribution vs. iteration number) to see the behavior and graphically determine when it looks like it has converged.

One benefit of MCMC is that you do not need to know the normalizing constant in Bayes' formula. The normalizing constant typically prevents an easy analytical solution for the posterior. This is one reason Bayesian statistics were not considered practical to implement until MCMC methods became widespread.

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 $f(X)$.

2.7.1. 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 $\alpha(X,Y) = \min(1, \frac{f_X(Y)q_Y(Y \mid X_i)}{f_X(X)q_X(X_i \mid Y)})$
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). The conditionals are often nontrivial to calculate. 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. [Gamerman, 1997] In some cases, it can be difficult to tune the proposal density parameters to obtain these acceptance rates. 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 further *k* iterates are obtained. Convergence is accepted when the histograms cannot be distinguished.

2.7.2. 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, but only when the full conditional distributions are tractable and can be sampled from easily. Note that the new points are always accepted in Gibbs sampling. Gibbs sampling is a special case of the Metropolis-Hasting algorithm, where the proposal distribution is chosen to be the full conditional and thus all the terms in the acceptance ratio cancel out and the acceptance ratio is always one.

Gibbs Sampling Algorithm

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

Repeat until converged:

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

a. $X_1^i \sim \pi(X_1 \mid X_2^{i-1}, ...X_d^{i-1})$

b. $X_2^i \sim \pi(X_2 \mid X_1^i, X_3^{i-1}, ...X_d^{i-1})$

.

.

.

c. $X_d^i \sim \pi(X_d \mid X_1^i, X_2^i, ...X_{d-1}^i)$

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.

2.8. Bayesian Software

There are many software packages available to perform various aspects of Bayesian computation. In the reference section, we provide a list of websites of interest. We have looked at five software packages in detail: FirstBayes, BUGS, YADAS, FBM, and Netlib. Below is a short summary of each of these packages. They are used and explained in more detail throughout subsequent sections of this report.

Anthony O'Hagan of Sheffield University, UK, created a teaching program called FirstBayes 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 one-dimensional conjugate priors. FirstBayes runs on Windows and requires manipulation of system configuration settings to start. FirstBayes has a GUI and can be useful to provide some ideas for test problems or prototype examples. 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 (Bayesian Inference Using Gibbs Sampling) is software based on the Gibbs sampling MCMC method. BUGS is one of the MCMC packages with the longest history. It was developed by MRC Biostatistics group and Imperial College School of Medicine, St. Mary's, London. A UNIX version is available but no longer supported. The Windows version requires a commercial license. BUGS has been tailored for reliability and biostatistics applications, though it can be used for general problems. Capabilities include the ability to handle hierarchical/nonHierachical models, conjugate and nonconjugate distributions, polynomial and logistic regression, weibull analysis and survival models, random or fixed effects, mixture models, and spatial models.

We have used the UNIX version of BUGS with a command line interface. It requires some work to understand the model specification, but the problem set up and execution is fairly straightforward. The user does not have much control over the actual sampling (choice of proposal distribution, for example).

YADAS (Yet Another Data Analysis System) has been developed by Todd Graves in the Statistical Sciences Division at LANL. YADAS is open source software written in Java. It has very similar capabilities to BUGS. Overall, it is more flexible than BUGS and allows more user control, but is somewhat harder to use. The user has to write Java classes vs. scripts in the BUGS command interface.

Radford Neal at University of Toronto has developed 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.

Netlab, developed by Ian Nabney and Christopher Bishop of Aston University, UK, is a collection of Matlab M-files. The Netlab library is most similar to FBM. It was developed for data analysis and neural network modeling. Netlab has a lot of capabilities, including pattern recognition/classification, Principal Component Analysis, K-means clustering, self-organizing maps, multi-layer perception networks, and radial basis function networks. The Bayesian component focuses on updating hyperparameters which govern some of these data modeling methods (such as RBFs, neural networks, or Gaussian processes).

To summarize the software tools: there are a lot of bits and pieces available, but there is no one Bayesian software tool which addresses Sandia's needs in computational modeling for engineering design, especially for updating multivariate distributions. Many of the tools require substantial statistical knowledge and domain expertise to formulate the problem correctly, and to interpret the results to ensure that the posterior distributions are approximately correct.

3. Application to Engineering Problems I

This section discusses the application of Bayesian methods to engineering problems. These applications are widely cited in the literature, specifically reliability estimation and Bayesian regression.

3.1. Example Reliability Problem

To demonstrate the application of Bayesian methods to a reliability problem, we start with a test function commonly used in optimization, the Rosenbrock function. It is 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 1, for variable bounds $-2 \le x_1, x_2 \le 2$.

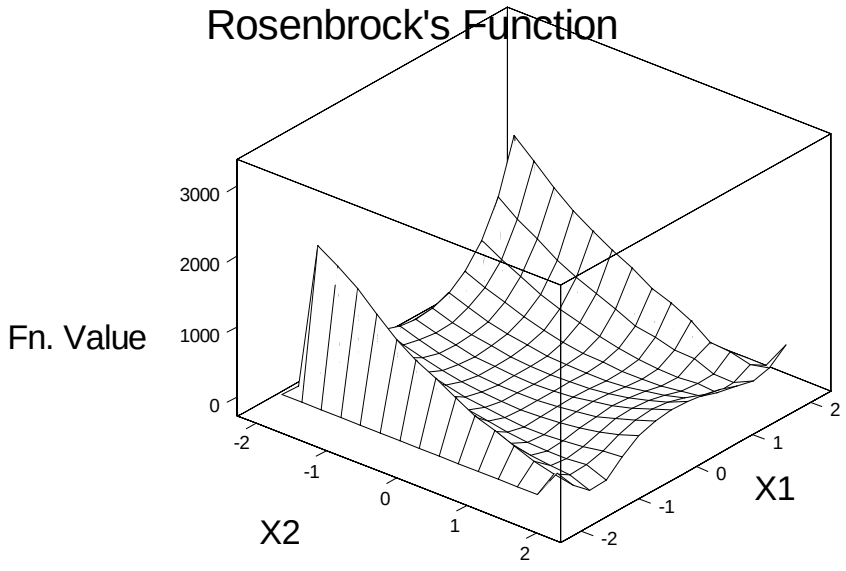

Figure 1. Rosenbrock's Function

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 a surrogate)

We used the LHS sampling method within DAKOTA to generate 110 sample values.

3.2. Binomial Model

In this example, 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:

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

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*~Beta(α,β). The posterior distribution for *p* given *k* failures out of *n* trials is:

$\Pr(p \mid k) \propto p^{k+\alpha-1}(1-p)^{n-k+\beta-1}$, so the posterior distribution is a Beta(α+k,β+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:

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

3.2.1. FirstBayes solution

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 Beta(10,90). We used the FirstBayes software to generate the posterior distribution. The prior information is shown in Figure 2.

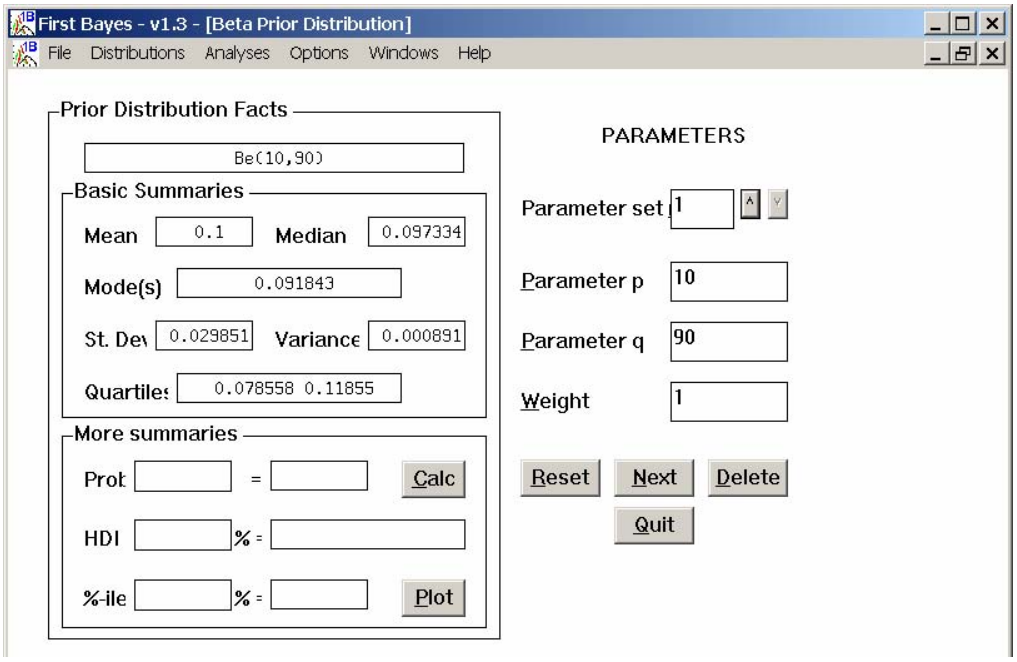

Figure 2. Beta Prior for Binomial Failure Example

The plot showing the prior, the posterior, and the likelihood function is shown in Figure 3:

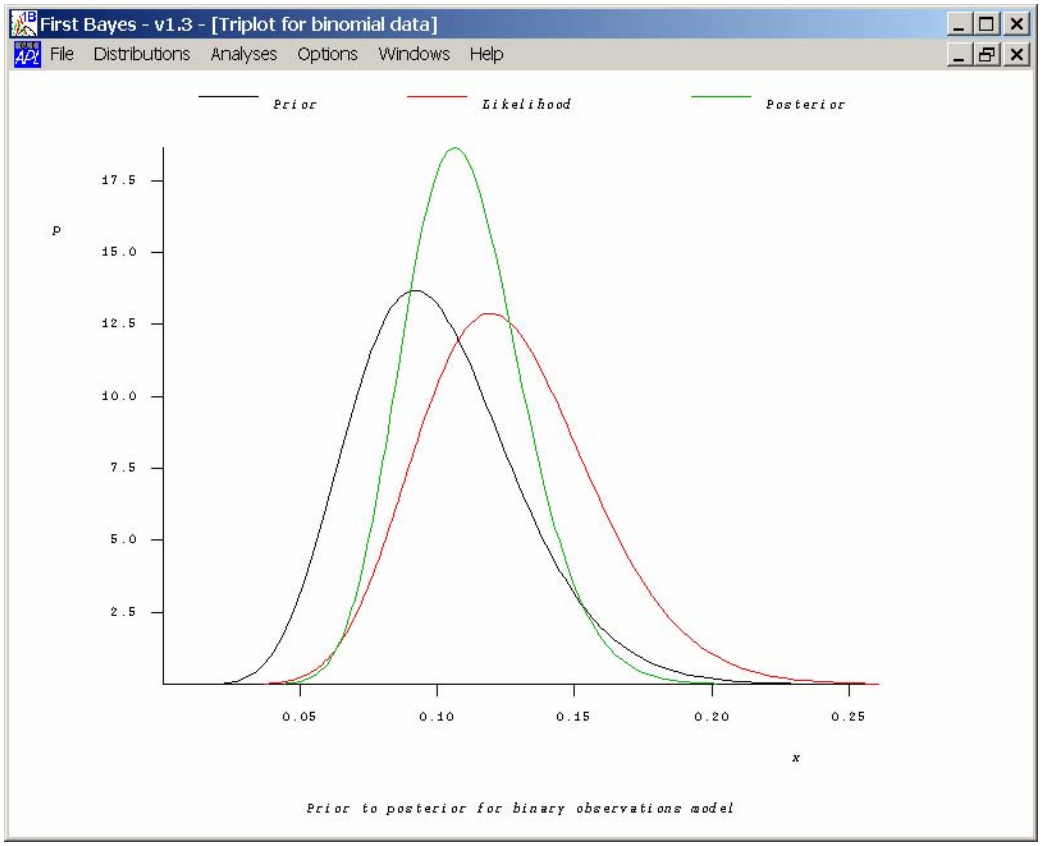

Figure 3. Bayesian Updating of the Beta Distribution for p, the Failure Probability

This shows that the posterior distribution is between the prior and the likelihood function. In fact, the posterior distribution shown in Figure 4 is a Beta(23,187) which is the formula Beta(α+k,β+n-k) since we had 13 failures and 97 successes in the original 110 points.

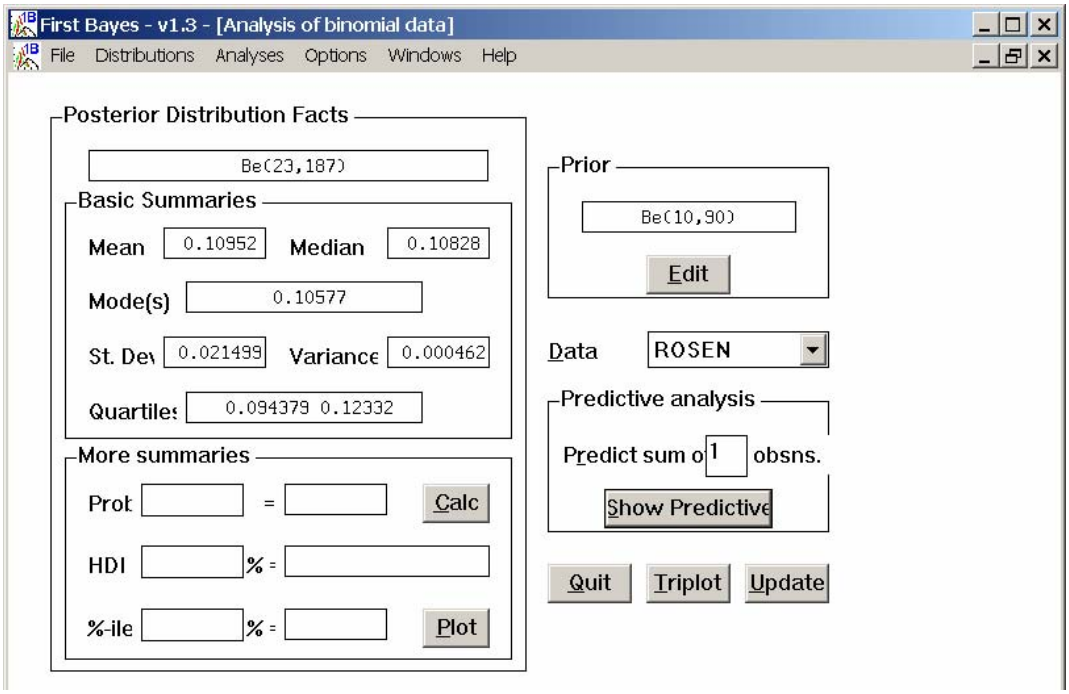

Figure 4. Posterior Distribution Summary

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 5.

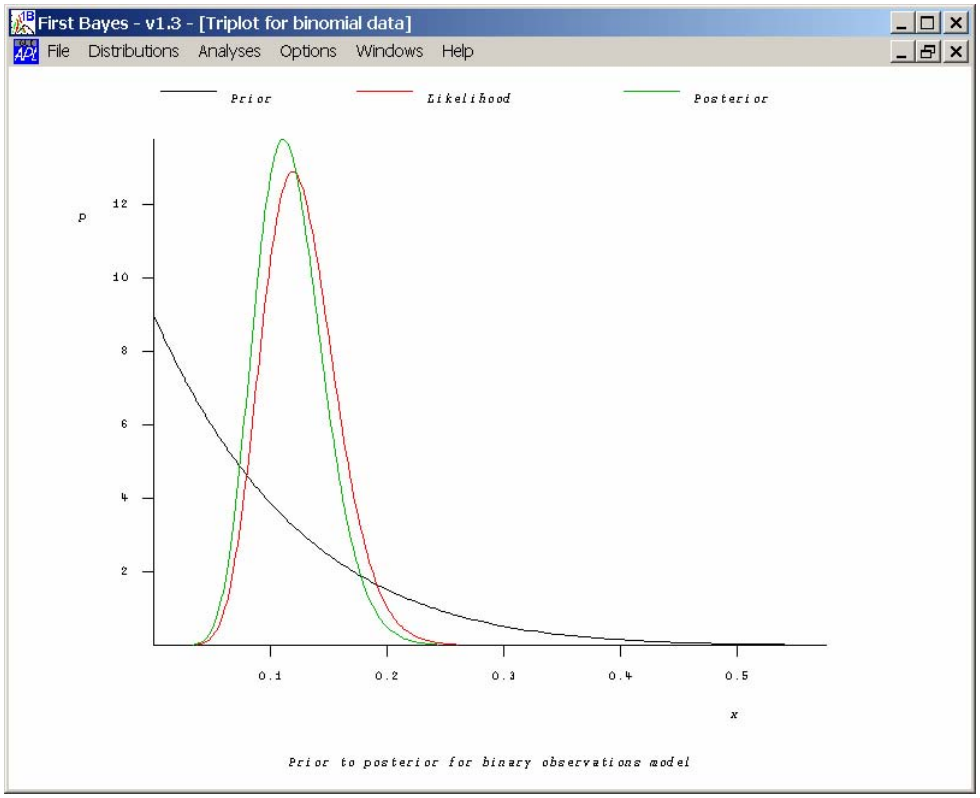

Figure 5. Binomial Failure Example with Be(1,9) prior distribution on p

3.2.2. BUGS solution

In BUGS, the input file for this binomial reliability model 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 trialsY[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.
```

3.2.3. YADAS Solution

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 most concise formulation 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 6. Note that in Figure 6, 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 different: 0.109 vs. 0.122. At this point, our 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 6, and the posterior data had significantly more spread than the BUGS output.

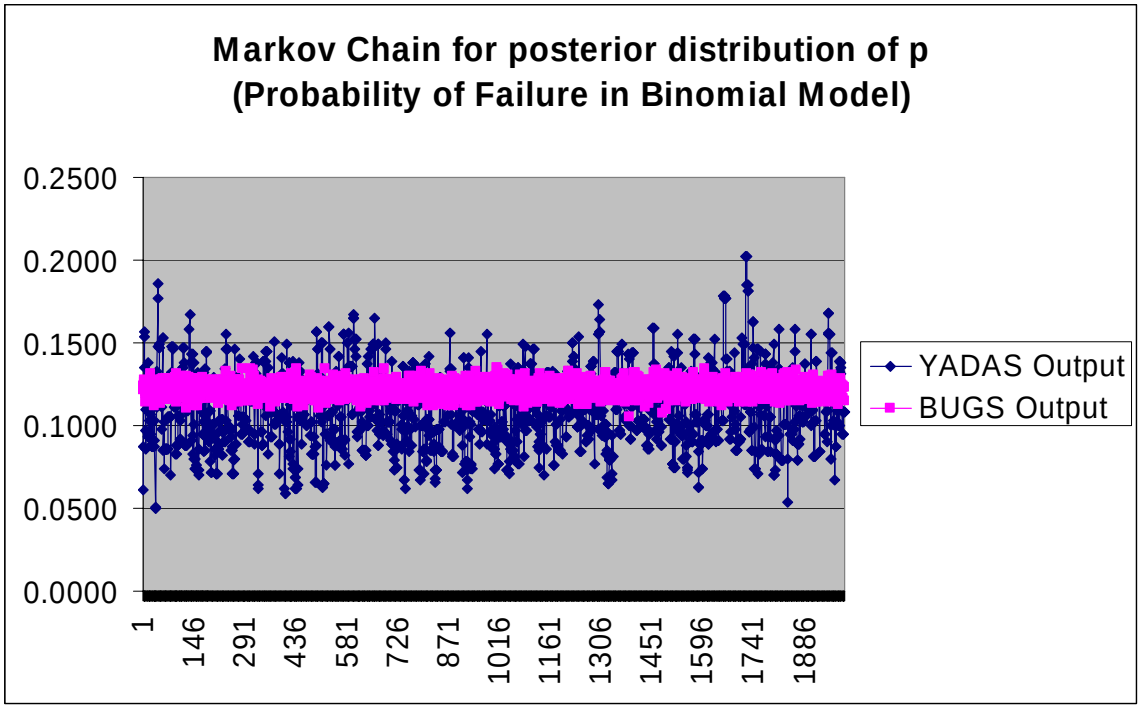

Figure 6. Comparison of YADAS and BUGS output

In summary, Bayesian methods have been used for estimation of failure probabilities in binomial failure models. If the prior distribution on the failure probability is specified with a beta distribution, the posterior can be determined analytically, with the beta parameters being updated by the number of failures/number of trials in the new data set. Thus, the MCMC sampling methods are not necessarily needed in the Bayesian reliability application. We discussed them above to provide comparison with the analytic results. The reliability application would be ideal in a situation where data on failures is continually being gathered over time.

3.3. Bayesian Regression

Bayesian regression is of interest to us because of the wide use of surrogates to approximate expensive computer simulations. The idea is to assume a prior on the coefficients of the regression equation, then update the prior with data.

The classical linear regression model is $E[y_i| \beta,X]= \beta_0+\beta_1X_{i1}+\ldots+\beta_kX_{ik}$, where i = 1….*n* for *n* observed values of *k* independent X variables. In ordinary linear regression, we assume the conditional variances are equal: $var(y_i| \theta, X) = \sigma^2$. The parameter vector we are trying to estimate is: $\theta = (\boldsymbol{\beta},\sigma^2) = (\beta_0, \beta_1X_1, \ldots ,\beta_k, \sigma^2)$. The key assumption in a Bayesian formulation of classical regression is that there is a distribution on θ, and that the posterior distribution of θ is given by: $p(\theta|\mathbf{X},y) \propto p(\theta)p(y|\mathbf{X}, \theta)$.[Gelman et. al, 1996] For most situations, it is assumed that the X values are known (they are chosen in the experiment), thus their probability distribution is known and fixed: it is not something that is updated. Also, in most formulations, the dependency on X is suppressed in the notation and just assumed. So the posterior distribution is written simply as: $p(\theta|y) \propto p(\theta)p(y|\theta)$.

A standard noninformative prior that is used is one that is uniform on θ. This is equivalent to being uniform on ($\boldsymbol{\beta}$, log σ) and is expressed as $p(\boldsymbol{\beta},\sigma^2|X) \propto \sigma^{-2}$. With this prior, the conditional posterior of $\boldsymbol{\beta}$ given σ^2 is normal: $\beta \mid \sigma^2, Y \sim N(\hat{\beta}, V_\beta\sigma^2)$. The estimate of $\boldsymbol{\beta}$ is the same as that given in classical linear regression:
$\hat{\beta} = (X^TX)^{-1}X^TY$. The variance term is given by: $V_\beta = (X^TX)^{-1}$. The marginal posterior density of σ^2 given the data is an inverse χ^2 distribution: $\sigma^2 \mid y \sim$ inverse χ^2 (n-k, s^2), where s^2 is the standard non-Bayesian estimate of σ^2 obtained in classical regression: $s^2 = \frac{1}{n-k}(Y - X\hat{\beta})^T(Y - X\hat{\beta})$. Thus, the Bayesian estimates for the mean of $\boldsymbol{\beta}$ and for σ^2 are the same as those obtained by classical regression, but these parameters have posterior distribution in the Bayesian framework as opposed to point estimates in the classical framework.

To compute the posterior distribution $p(\boldsymbol{\beta},\sigma^2|Y)$, first one calculates $\hat{\beta}, V_\beta$, and s^2 from the matrix formulas given above. Then, using s^2, one draws σ^2 from the inverse χ^2 distribution. Finally, given $\hat{\beta}, V_\beta$, and σ^2, one draws a sample value from the posterior distribution $(\beta \mid \sigma^2, Y) \sim N(\hat{\beta}, V_\beta\sigma^2)$. Most standard linear regression packages can perform the estimation of $\hat{\beta}, V_\beta$, and s^2. In terms of prediction, one wants to predict the outcome $\tilde{y}$ given a new set of data $\tilde{X}$. To obtain the predicted outcome, one first draws $\boldsymbol{\beta}$ and σ^2 from their posterior distributions, then draws a predicted outcome according to:
$\tilde{y} \sim N(\tilde{X}\beta, \sigma^2I)$. The mean of this posterior distribution for $\tilde{y}$ is $E[\tilde{y} \mid \sigma^2, y] = \tilde{X}\hat{\beta}$. This is the same as the estimate obtained by classical linear regression.

3.3.1. Regression analysis in FirstBayes

FirstBayes only allows a regression using one independent variable and one dependent variable. We chose to use x_1 as the independent variable (denoted by X in the FirstBayes software) and the Rosenbrock function as Y. The analysis in FirstBayes assumes that Y = alpha + beta * X. Figure 7 below shows that with a noninformative prior, the posterior distribution of alpha is a *t*-distribution with mean 475.6. Note that in FirstBayes, the prior is not allowed to be chosen for the linear regression model: it is only

specified as a "weak" prior as seen on the left side of the screen shot. Figure 8 shows that the posterior distribution of beta is a *t*-distribution with a mean of –33.3. A few things to note: these *t*-distributions are essentially normals because they have 108 degrees of freedom (110 data points – 2 estimated regression coefficients). The means of these posterior distributions are the same as the point estimates that are predicted by classical regression. The comparison can be seen in the regression function computed in Excel, shown in Figure 9. The estimate of the intercept is 475.6 and the estimate of the coefficient of X1 is –33.3. The estimate of σ^2 in FirstBayes is a scaled inverse χ^2-distribution with mean 3.424E7, which is the same as the residual sum of squares in the classical regression. Of course this regression is a very poor fit, as seen by an R^2 term of nearly zero and the plot of the regression line vs. the actual function shown in Figure 10.

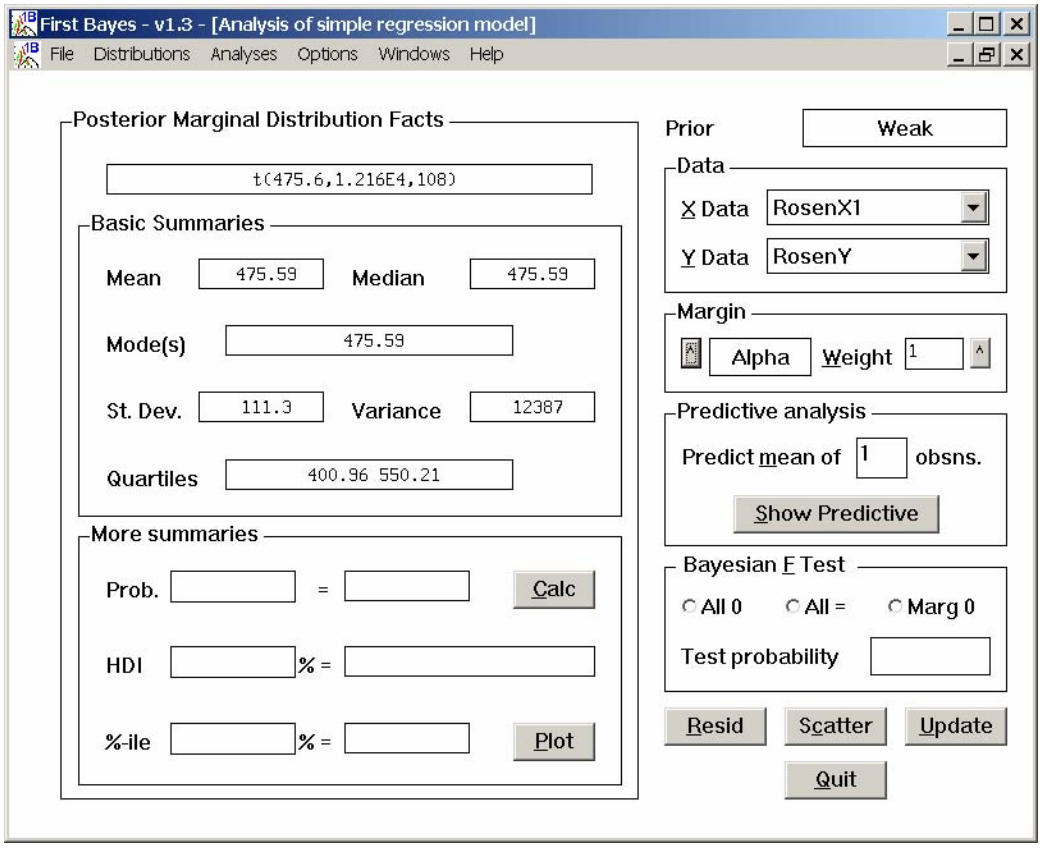

Figure 7. Posterior Distribution of alpha in a simple linear model in FirstBayes

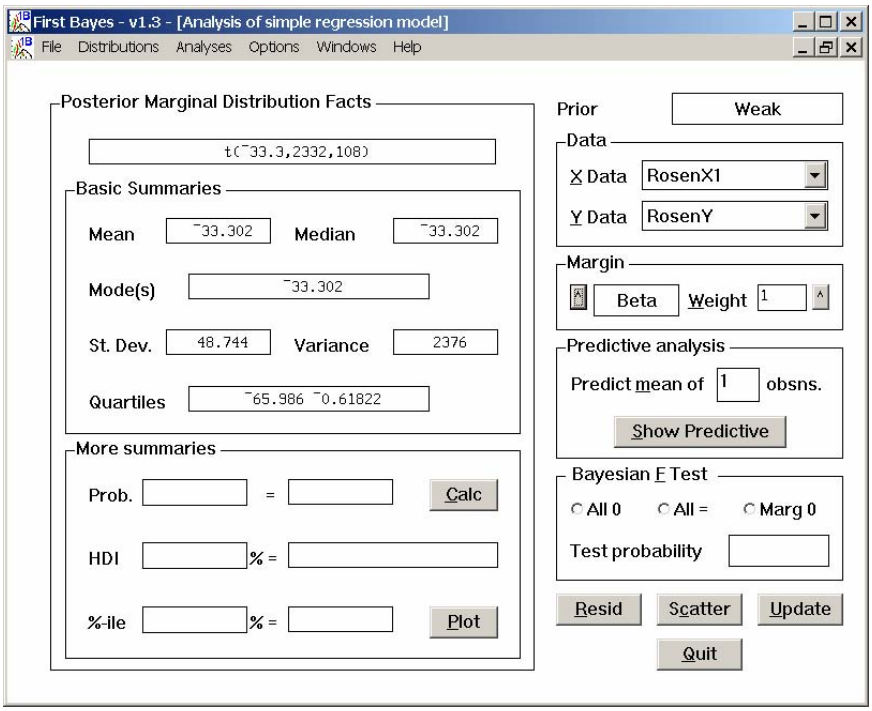

Figure 8. Posterior Distribution of beta in a simple linear model in FirstBayes

SUMMARY OUTPUT

Regression Statistics	
Multiple R	0.066212
R Square	0.004384
Adjusted R Square	-0.00483
Standard Error	563.0526
Observations	110

ANOVA

	df	*SS*	*MS*	*F*	*Significance F*
Regression	1	150764.8	150764.8	0.47555639	0.491921086
Residual	108	34239047	317028.2		
Total	109	34389812			

	Coefficients	*Standard Error*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*
Intercept	475.5867	110.2633	4.313193	3.5721E-05	257.0261154	694.147338
X Variable 1	-33.3015	48.29068	-0.68961	0.49192109	-129.0219345	62.4188469

Figure 9. Classical Regression Results for Rosenbrock's Function with 1 independent variable

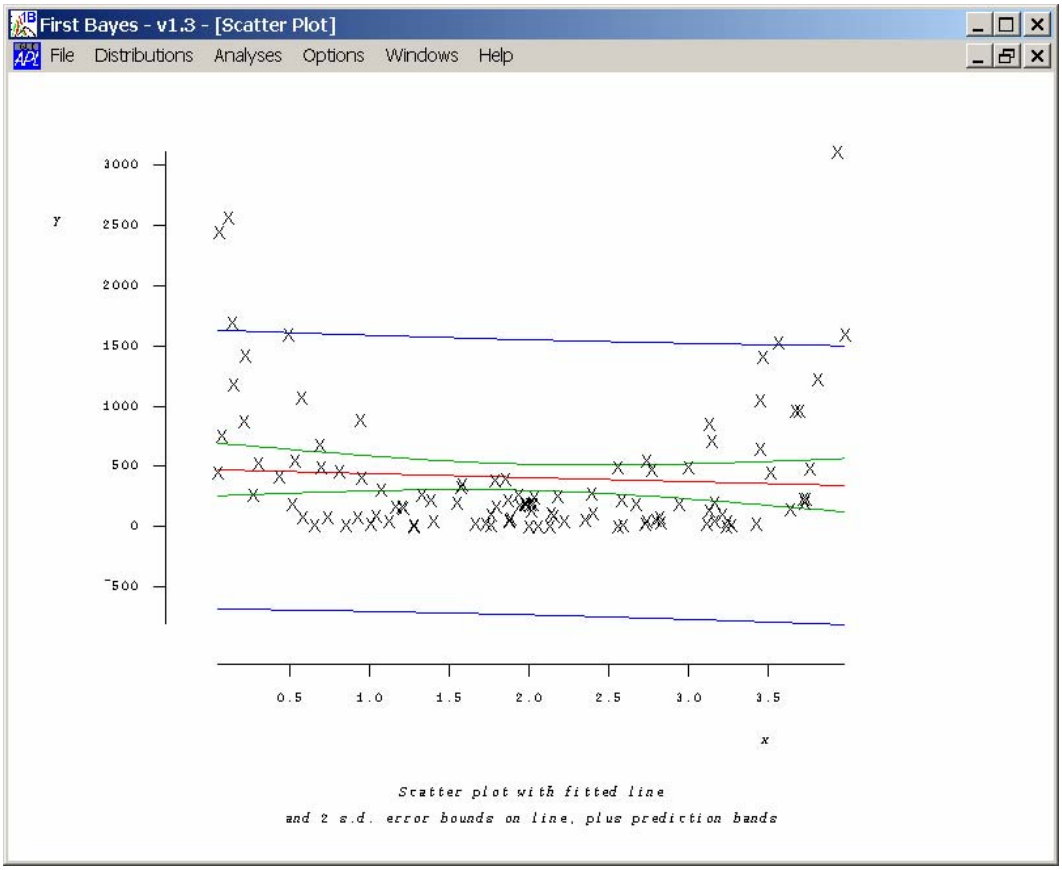

Figure 10. Regression line and confidence bounds for a simple linear model in FirstBayes

3.3.2. Linear Regression in BUGS

We continued the linear example by looking at the results from BUGS. The BUGS software does allow a Bayesian regression analysis on multiple independent variables. The BUGS formulation for this model is:

```
model line;
const
  N = 110;  # number of observations
var
  x1[N],x2[N],Y[N],mu[N],alpha,beta1,beta2,tau,sigma;
data x1,x2,Y in "vol1/lin/blin.dat";
inits in "vol1/lin/lin.in";
{
   for (i in 1:N) {
       mu[i] <- alpha + beta1*x1[i] + beta2*x2[i];
       Y[i]   ~ dnorm(mu[i],tau);
   }
   alpha    ~ dnorm(400.0,1.0E-4);
   beta1    ~ dnorm(-30.0,1.0E-4);
   beta2    ~ dnorm(-220.0,1.0E-4);
   tau      ~ dgamma(54,1.33E+7);
   sigma    <- 1.0/sqrt(tau);
 }
```

In this formulation, the expected value of Y (mu) is the sum of the independent parameters times the regression coefficients alpha, beta1, and beta2. These regression coefficients are distributed as normal variables. The independent variable Y is distributed as a normal variable with mean equal to mu and variance equal to tau. According to the posterior distributions in a Bayesian analysis, the predicted value of y is: $\tilde{y} \sim N(\tilde{X}\beta, \sigma^2 I)$, where the posterior distribution of the error is: $\sigma^2 \mid y \sim$ scaled inverse χ^2 (n-k, s^2). The scaled inverse Chi-squared distribution is equivalent to an inverse gamma distribution according to the following transformation: if $\sigma^2 \sim$ scaled inverse χ^2 (n-k, s^2), then $\sigma^2 \sim$ inverse gamma ((n-k)/2, (n-k)*s^2 /2). Then, if $\sigma^2 \sim$ inverse gamma, the reciprocal, $1/\sigma^2 \sim$ gamma((n-k)/2, (n-k)*s^2 /2). Thus, we see in the last line that sigma is 1/sqrt(tau), where tau is distributed as a gamma distribution. This is a common transformation in Bayesian analysis: people tend to use the gamma distribution, not the inverse χ^2.

We first performed a classical regression analysis on this problem with the two independent variables x_1 and x_2. The results are shown in Figure 11:

SUMMARY OUTPUT

Regression Statistics	
Multiple R	0.475545
R Square	0.226143
Adjusted R Square	0.211678
Standard Error	498.7161
Observations	110

ANOVA

	df	*SS*	*MS*	*F*	*Significance F*
Regression	2	7777016	3888508	15.6342211	1.1E-06
Residual	107	26612796	248717.7		
Total	109	34389812			

	Coefficients	*Standard Error*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*
Intercept	436.1627	47.80396	9.123986	4.8892E-15	341.3968	530.9285
X Variable 1	-31.6632	42.77383	-0.74025	0.4607712	-116.457	53.13098
X Variable 2	-227.837	41.14549	-5.53735	2.2122E-07	-309.403	-146.271

Figure 11. Classical Regression Results for Linear Model of Rosenbrock's function

The results are still not very good (R^2 value of .22). We tried many combinations in BUGS to get the updating working right. We did not get good results unless we used the information produced above in the classical analysis. Thus, we assumed a value of 400 as a prior mean for alpha (the exact value of the intercept is 436 in classical regression), a value of –30 for the coefficient of x_1, and a value of –220 for the coefficient of x_2. The variance terms need some explanation. The estimate of σ^2 in classical regression is s^2, where s^2 is the sum of squares of the residuals divided by the number of degrees of freedom (in this case, 108). Thus, s^2 in this example is 26612796/108 = 246414.8, and the square root of this term is the estimate of σ. In this case, the estimate of σ is 496.4. Recall that the BUGS formulation requires the transformation to a gamma distribution with parameters ((n-k)/2, (n-k)*s^2 /2). In this case, (n-k)/2 is 54, and 54* s^2 = 1.33E7 which is the second parameter of the gamma distribution. That is why the input specification portion of the input file looks like:

```
alpha    ~ dnorm(400.0,1.0E-4);
beta1    ~ dnorm(-30.0,1.0E-4);
beta2    ~ dnorm(-220.0,1.0E-4);
tau      ~ dgamma(54,1.33E+7);
sigma    <- 1.0/sqrt(tau);
```

With these inputs, we ran 500 updates to initialize the Markov chain (this is the number recommended as the standard initialization in BUGS), then we ran the chain out for 10000 more updates. The summary statistics on the posterior distributions are given below:

```
 update(10000)
     time for    10000  updates was  00:00:01
Bugs>stats(alpha)
             mean       sd      2.5% :  97.5%  CI    median      sample
           4.298E+2   4.276E+1   3.468E+2   5.129E+2   4.299E+2     10000
Bugs>stats(beta1)
             mean       sd      2.5% :  97.5%  CI    median      sample
          -3.181E+1   3.921E+1  -1.085E+2   4.391E+1  -3.169E+1     10000
Bugs>stats(beta2)
             mean       sd      2.5% :  97.5%  CI    median      sample
          -2.265E+2   3.777E+1  -3.007E+2  -1.523E+2  -2.263E+2     10000
Bugs>stats(tau)
             mean       sd      2.5% :  97.5%  CI    median      sample
           4.044E-6   3.925E-7   3.311E-6   4.844E-6   4.029E-6     10000
Bugs>stats(sigma)
             mean       sd      2.5% :  97.5%  CI    median      sample
           4.990E+2   2.436E+1   4.543E+2   5.495E+2   4.982E+2     10000
```

The mean posterior estimates of the parameters are reasonably close to the estimates obtained by classical regression, which is we expect from the derivations outlined above. The confidence intervals on these parameters are large: for example the intercept alpha has a 95% confidence interval of 346.8 to 512.9, and the 95% CI for the beta1 parameter (coefficient of x1) is from –108.5 to 43.9. These large confidence intervals are due to the variability in the data and the poor linear fit, since the prior variances on these distributions were small (1E-4). Also, note that the posterior estimate of sigma is 499, which is very close to the classical estimate of 496.4.

If we start BUGS with a “dumb” estimate of the parameters as normals centered on one, such as the following input specification, the results are a lot worse:

Input file:

```
{
   for (i in 1:N) {
        mu[i] <- alpha + beta1*x1[i] + beta2*x2[i];
        Y[i]   ~ dnorm(mu[i],tau);
   }
   alpha    ~ dnorm(1.0,1.0E-2);
   beta1    ~ dnorm(1.0,1.0E-2);
   beta2    ~ dnorm(1.0,1.0E-2);
   tau      ~ dgamma(1.0,1.0);
   sigma    <- 1.0/sqrt(tau);   }
```

Output statistics:

```
Bugs>stats(alpha)
          mean      sd     2.5% :  97.5% CI   median     sample
       1.057E+1  9.808E+0 -8.408E+0  2.991E+1  1.059E+1    10000
Bugs>stats(beta1)
          mean      sd     2.5% :  97.5% CI   median     sample
      -1.506E-2  9.786E+0 -1.951E+1  1.909E+1  1.072E-1    10000
Bugs>stats(beta2)
          mean      sd     2.5% :  97.5% CI   median     sample
      -5.104E+0  9.859E+0 -2.427E+1  1.412E+1 -5.082E+0    10000
Bugs>stats(tau)
          mean      sd     2.5% :  97.5% CI   median     sample
      2.166E-6  2.939E-7  1.627E-6  2.780E-6  2.149E-6    10000
Bugs>stats(sigma)
          mean      sd     2.5% :  97.5% CI   median     sample
      6.842E+2  4.689E+1  5.997E+2  7.840E+2  6.821E+2    10000
```

In this case, the posterior estimates have not converged to their "true" estimates even after 10000 iterations of the Markov chain, where "true" refers to those predicted by derivation (namely, the means of these parameter distributions should be equal to the classical regression estimates). This shows the importance of getting very good prior estimates.

We extended the linear model to allow for three additional terms: $x_1 x_2$ and x_1^2 and x_2^2
The classical regression results are shown in Figure 12. As expected, R^2 is getting better (up to nearly 70%!)

SUMMARY OUTPUT

Regression Statistics	
Multiple R	0.831785
R Square	0.691867
Adjusted R Square	0.677053
Standard Error	319.2034
Observations	110

ANOVA

	df	*SS*	*MS*	*F*	*Significance F*
Regression	5	23793167	4758633	46.70326	4.42316E-25
Residual	104	10596645	101890.8		
Total	109	34389812			

	Coefficients	*Standard Error*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*
Intercept	-91.8079	58.19001	-1.57773	0.117666	-207.2007224	23.5849266
x1	-21.4352	27.54554	-0.77817	0.438235	-76.05892254	33.1886104
x2	-229.535	26.46961	-8.67164	6.14E-14	-282.0252323	-177.0449
x1x2	-10.5518	24.56796	-0.4295	0.668452	-59.27094267	38.1673093
x1^2	311.563	25.62508	12.15852	1.08E-21	260.7475695	362.378416
x2^2	106.1224	26.54916	3.997204	0.00012	53.47449247	158.770307

Figure 12. Classical Regression Results for Quadratic Model of Rosenbrock's Function

The input file for BUGS becomes:

```
model line;
const
  N = 110;  # number of observations
var
  x1[N],x2[N],x12[N],x1sq[N],x2sq[N],Y[N],mu[N],alpha,beta1,beta2,beta12,beta1sq,beta2sq,tau,sigma;
data x1,x2,x12,x1sq,x2sq,Y in "vol1/quad/quad.dat";
inits in "vol1/quad/quad.in";
{
   for (i in 1:N) {
      mu[i] <- alpha + beta1*x1[i] + beta2*x2[i] + beta12*x12[i] + beta1sq*x1sq[i] + beta2sq*x2sq[i];
      Y[i]  ~ dnorm(mu[i],tau);
   }
   alpha   ~ dnorm(-90.0,1.0E-2);
   beta1   ~ dnorm(-20.0,1.0E-2);
   beta2   ~ dnorm(-230.0,1.0E-2);
   beta12  ~ dnorm(-10,1.0E-2);
   beta1sq ~ dnorm(310,1.0E-2);
   beta2sq ~ dnorm(106,1.0E-2);

   tau     ~ dgamma(52,5.3E+6);
   sigma   <- 1.0/sqrt(tau);

}
```

The output from BUGS looks very good:

```
Bugs>update(10000)
     time for    10000  updates was  00:00:02
Bugs>stats(alpha)
              mean       sd      2.5% :  97.5%  CI    median      sample
           -8.995E+1   9.684E+0  -1.088E+2  -7.054E+1  -9.023E+1     10000
Bugs>stats(beta1)
              mean       sd      2.5% :  97.5%  CI    median      sample
           -2.017E+1   9.417E+0  -3.856E+1  -1.694E+0  -2.025E+1     10000
Bugs>stats(beta2)
              mean       sd      2.5% :  97.5%  CI    median      sample
           -2.300E+2   9.197E+0  -2.480E+2  -2.120E+2  -2.299E+2     10000
Bugs>stats(beta12)
              mean       sd      2.5% :  97.5%  CI    median      sample
           -1.019E+1   9.265E+0  -2.856E+1   7.855E+0  -1.011E+1     10000

Bugs>stats(beta1sq)
              mean       sd      2.5% :  97.5%  CI    median      sample
            3.103E+2   8.693E+0   2.934E+2   3.274E+2   3.102E+2     10000
Bugs>stats(beta2sq)
              mean       sd      2.5% :  97.5%  CI    median      sample
            1.059E+2   8.660E+0   8.909E+1   1.228E+2   1.058E+2     10000
Bugs>stats(sigma)
              mean       sd      2.5% :  97.5%  CI    median      sample
            3.166E+2   1.534E+1   2.883E+2   3.484E+2   3.160E+2     10000
```

These posterior parameter estimates have means very close to the classical estimates. Note that the size of the 95% CI has decreased, probably because of the increase in independent variables. For example, the 95% CI of alpha is –108.8 to –70.5. Also, the estimate of sigma predicted by classical regression is 319.2 and with the Bayesian estimate, the mean of the estimate of sigma is 316.6.

This example demonstrates that it is possible to perform Bayesian estimation of the parameters using MCMC with a generalized linear model (in this case, a quadratic model) IF one has good estimates of the priors. We used the classical regression estimates as priors, which are not strictly appropriate since the classical regression estimates are based on data, and thus we are not strictly separating the prior from the data. In the cases that we have examined, the Bayesian confidence intervals for the regression coefficients are larger than those in classical regression, which is to be expected because the Bayesian estimate accounts not only for uncertainty in the data but also for uncertainty in the regression coefficients themselves.

3.4. Bayesian Regression Models in Optimization

This section discusses how one might use the posterior distribution of the regression coefficients in an optimization process. Regression models are often used as surrogate models for expensive computer simulations. Jin et al. state [2001]: "When using computationally expensive simulation programs in engineering design, it becomes impractical to rely exclusively on simulation codes for the purpose of design optimization. A preferable strategy is to utilize approximation models which are often referred to as metamodels since they provide a 'model of the model' to replace the expensive simulation model."

When using a Bayesian approach to construct the regression metamodel, the user first must specify a prior on the regression coefficients (or specify a non-informative prior, which is often used), then update the prior with the data from the expensive simulation runs to obtain a posterior distribution on the coefficients. This posterior distribution can then be sampled from to obtain samples of posterior regression models, in the sense of having a "family" or "ensemble" of possible regression functions according to the posterior distributions. At this point, optimization can be done on the regression models to compare how the optima of the individual regressions differ, and get a sense of the spread of the optima across the posterior family of surrogate models.

We constructed an example of generating a Bayesian regression model using data generated as part of some parameter studies for an earth penetrator, then optimized the posterior regression functions to obtain the distribution of the optimal designs. This example is not fully realistic: we did not include all the design parameters or uncertain variables. It is simply meant to illustrate how Bayesian approaches might be used in engineering design optimization, and more generally, in optimization under uncertainty (OUU).

We took a partial data set from studies performed as part of the Pen-X earth penetrator design sensitivity analyses. These parameter studies were done using the three-dimensional explicit transient dynamics code Presto to model mechanical deformation at impact. Presto is a Lagrangian FEM code. Each run of the Pen-X penetrator model in Presto is expensive, even in the low-fidelity case.

For the Bayesian regression, we identified three independent variables affecting design performance: L1 (one of the section lengths), IV (impact velocity), and CR (cavity radius). The model exhibits quadratic behavior with respect to IV and CR, so the full set of variables in the regression was L1, IV, CR, IV^2, and CR^2. The dependent variable that we are trying to predict with the regression (and optimize with respect

to the independent variables) is displacement. Displacement refers to maximum displacement of the ground due to penetrator impact.

We assumed noninformative priors, and wrote code to solve for the posterior distribution in MATLAB according to the equations outlined in Section 3.3 and found in Gelman et. al's book [1995] Note that we are calculating an analytic form of the posterior distribution on the regression coefficients: we are not using MCMC methods. We then drew samples from the posterior distribution for the regression coefficients β (which is a multivariate normal). Based on the sets of sampled coefficients, we constructed sample realizations of the regression response surface.

Figure 13 shows the posterior realizations of the regression functions. The red lines in Figure 13 are 20 particular realizations of the response surface based on sampling the posterior. The diamonds along the red lines are points where we actually evaluated the posterior regression functions. We had to do some manipulation to choose the points being shown so that the regression as a function of 1-D even though it is a function of 3 independent vars (L1,IV, and CR) plus the squared terms. The black Xs show the optimal solution of each posterior regression model. Note that the posterior family is quite wide, and the optimal solution distribution is fairly wide - from a value of 12.4 to 13.6 (this translates to 12,400 ft/sec to 13,600 ft/sec in terms of optimal value of impact velocity to maximize displacement).

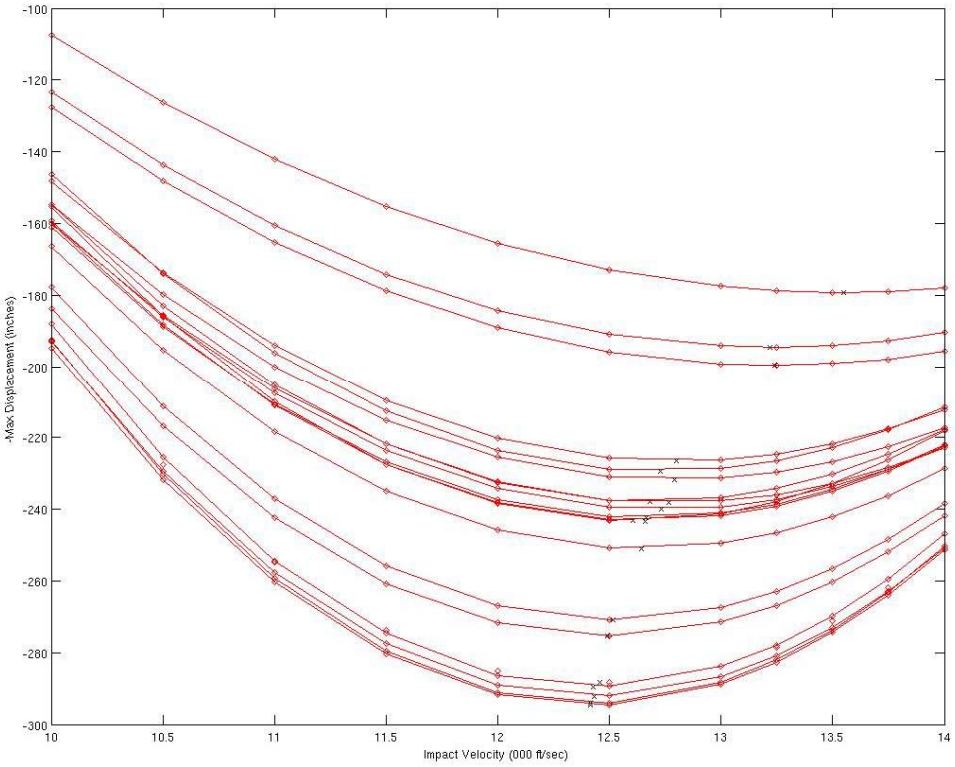

Figure 13. Example of Bayesian Quadratic Regression used in Optimization

At this point, we are examining the idea of many realizations of a regression surface in an optimization framework. In a trust region approach, generating posterior realizations offers many possibilities:

- We could generate a family of posterior regression models in the first trust region, then follow each of these initial optima through the rest of the optimization process.

- We could generate one posterior distribution of the entire space and not allow shrinkage (non-windowed trust region model), but update that posterior as additional points are taken.
- We could generate the posterior distributions and families of optima in each trust region. This is probably too expensive, even if the optimization is done on a regression model.

At this point, we are also considering how to incorporate the Bayesian regression models into an Optimization under Uncertainty (OUU) framework.

3.5. Bayesian Methods in Optimization

The previous section discussed Bayesian methods in optimization focused on regression models. This section discusses the application of Bayesian methods to more general optimization problems. In the evolutionary algorithms community, there has been interest in using Bayesian methods to help generate a probability distribution on which "genes" in the solution chromosome should be selected. For example, Pelikan et al. [1999], Pelikan et al. [2000] describe how they estimate a probability distribution on promising solutions in order to generate new candidate solutions in a genetic algorithm. They use a Bayesian network to model a multivariate probability distribution which is updated as new solution candidates are generated and evaluated (thus producing new data on the fitness landscape). In general, we think that the idea of combining a Bayesian network which encodes relationships between variables in a problem and updating the probability distribution governing that relationship to use in generating the next set of candidate solutions for an optimization technique is very useful. However, evolutionary algorithms in general are too expensive (in terms of number of function evaluations needed) for our needs in design optimization. Thus, we have not pursued this area of Bayesian optimization at this time.

Another optimization area of interest to us is robust design. There was a renewed in Taguchi's work in the statistics community during the mid-1990s. Taguchi had the idea that products lack high quality because of inconsistency in performance, one tries to choose values of control variables that result in a process that is robust or insensitive to environmental variation. Taguchi's approach separated the control variables (what we think of as design variables) from the noise variables (what we think of as uncertain variables) and developed separate experimental designs for each of these. The work in the mid-1990s [Vining and Myers, 1990; Khattree, 1996; Myers et al. 1997] focused on treating both the control variables and the noise variables in a combined array, so that one does not need a separate design of experiments for each. This results in an experimental design that is simpler to execute and avoids biases that appear in main effects estimates due to interactions that are ignored when the design is highly fractionated.

After the data from the experimental design runs is collected, regression is used to analyze the interaction terms between the noise and control variables. Preliminary findings using the low-fidelity data from the penetrator model suggests that the interaction terms are very small, meaning that the process variance generated from the noise variables is constant and there is no opportunity for reducing the variance by a choice of the control variables. This is an important point: if indeed there are no interaction terms, we have to incorporate the uncertainty in the noise variables but it will have a "constant" effect on the results which cannot be improved by a choice of the design variables. In this case, "robust" design will not be possible, at least in terms of what robust design means in the statistical community. We believe that higher fidelity models will reveal more interaction terms and more opportunity for using ideas from robust design.

The Bayesian approach to robust design optimization is as follows [Peterson, 2000; Miro-Quesado et al., 2002]: maximize the posterior predictive probability that the process satisfies a set of constraints on the

responses. The twist on a standard Bayesian analysis is that the predictive density is integrated not only with respect to the response variables, but also with respect to the noise variables.

Overall, it is important to note that we did not find a Bayesian approach to "classical" optimization methods such as Newton's method, etc. The Bayesian approach used in evolutionary algorithms has much potential, but not for our class of problems. The robust design approach has direct relevance to engineering design, but it is not clear at this point how much a Bayesian approach will augment what can be done with the statistical approach of performing combined noise/control variable experiments to understand interactions. Finally, we briefly mention that surrogate methods are often used in conjunction with Bayesian methods because the surrogates are much cheaper to evaluate when one needs to take tens of thousands of samples to obtain a posterior. The next section discusses a common surrogate method used, Gaussian processes. We have seen Gaussian process models used primarily to characterize the space and make predictions about function values at new input points, but it is certainly possible to use them additionally in optimization. This is topic of future research interest.

4. Application to Engineering Problems II

In Section 3, we discussed the application of a Bayesian approach to two problems: reliability estimation and regression. This section discusses the application of Bayesian analysis to Gaussian process models. There has been a surge of interest in Gaussian process models over the past ten years. They are often used as surrogates or emulators for noisy engineering functions because of their ability to capture the local behavior of the system response. Once the parameters governing the Gaussian process are determined, they can be used for optimization or sensitivity analysis because the Gaussian process estimates of system response are computationally cheap to compute.

We present the use of a Gaussian process model as an emulator for a function, and then we discuss two applications of interest: calibration under uncertainty, and multi-fidelity models.

4.1. Gaussian Processes

Gaussian Process models are used in response surface modeling, especially response surfaces which "emulate" complex computer codes. 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. Several of their web sites and technical reports are listed in the references.

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 $\Re^d$, where d is the number of inputs). The stochastic process is defined by giving the joint probability distribution for every finite subset of variables $Y(\mathbf{x}_1), ..Y(\mathbf{x}_k)$. 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 $y(x) = \sum_j w_j \phi_j(x) = \mathbf{w}^T \boldsymbol{\phi}(x)$, 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 is:

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

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 $\{\mathbf{x}_1, \mathbf{x}_2, .. \mathbf{x}_n\}$ and a set of associated observed responses or "targets" $\{z_1, z_2, .. z_n\}$, 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^2_ε. 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 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 $N(\mathbf{0},\mathbf{K}+\sigma^2_\varepsilon \mathbf{I}_n)$. The distribution of the predicted term z_{n+1} is conditional on the data $\{z_1, z_2, .. z_n\}$. It is Gaussian with the following mean and variance:

$$E[z_{n+1} \mid z_1, z_2, .. z_n] = \mathbf{k}^T \mathbf{C}^{-1} \mathbf{z}$$
$$Var[z_{n+1} \mid z_1, \ldots, z_n] = C(\mathbf{x}_{n+1}, \mathbf{x}_{n+1}) - \mathbf{k}^T \mathbf{C}^{-1} \mathbf{k}$$

where **k** is the vector of covariance between the *n* known targets and the new *n+1* data point: $k = (C(\mathbf{x}_1, \mathbf{x}_{n+1}), \ldots C(\mathbf{x}_n, \mathbf{x}_{n+1}))^T$, **C** is the $n * n$ covariance matrix of the original data, and **z** is the *n*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 **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 called Netlab, which is a collection of Matlab M-files [Nabney and Bishop]. In addition to Gaussian processes, this code also has capabilities 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, and MCMC methods.

The second 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.

4.1.1. Gaussian Process Model of the Rosenbrock function

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 our 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 $-2 \le x_1, x_2 \le 2$.

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. 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 [Booker]. 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. Another way is to perform an adaptive partitioning of the space, and find different sets of covariance parameters which govern local regions of the space [Lee, Gramacy, and Macready].

4.1.2. Netlab implementation of Gaussian Processes

Below are two graphs showing the Gaussian process output vs. one input, X1, for the Rosenbrock function. Figure 14 is based on 11 input points, while Figure 15 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.

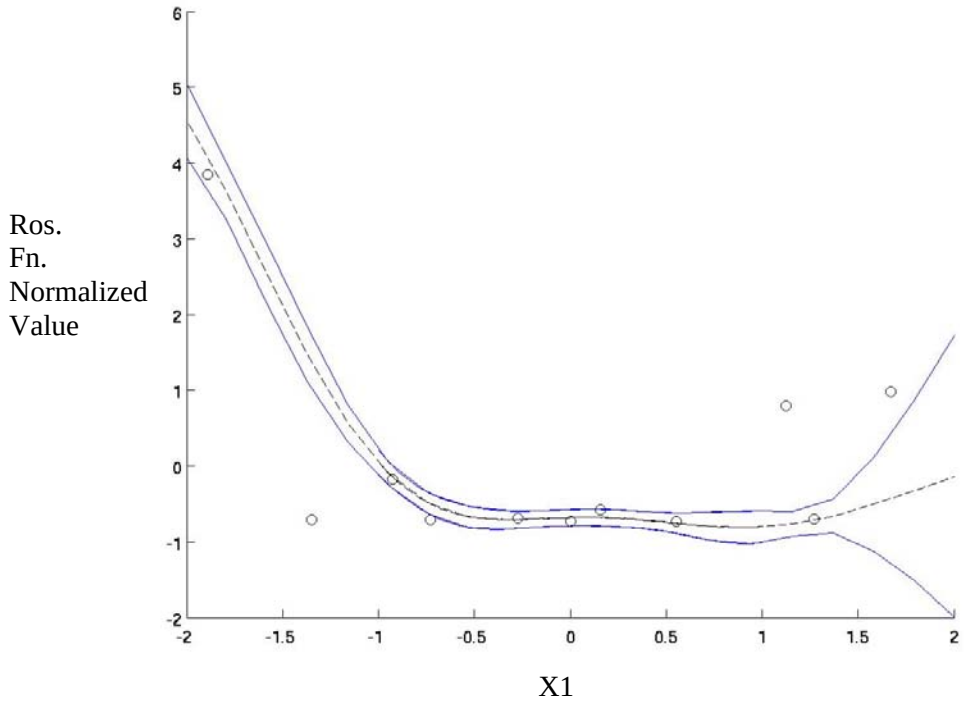

Figure 14. Gaussian Process for Rosenbrock Function based on 11 input points

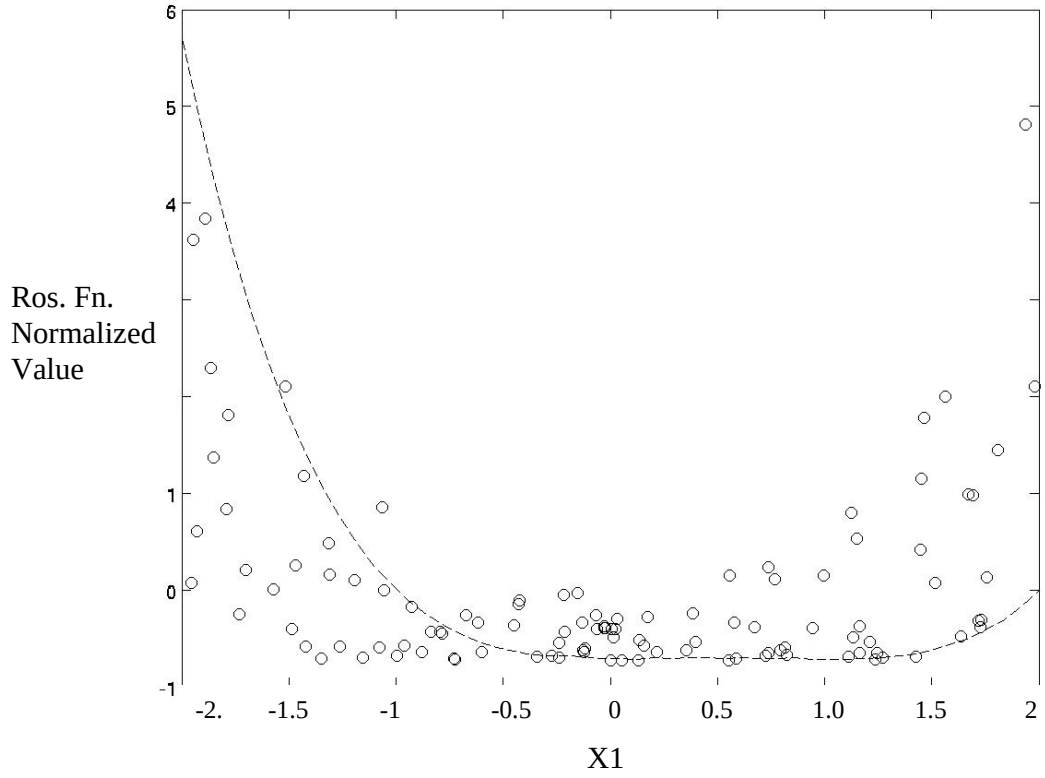

Figure 15. Gaussian Process for Rosenbrock Function based on 110 input points

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), calculated the covariance/inverse covariance matrix, defined 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.

4.1.3. FBM Implementation of Gaussian Processes

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:

     10.00

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 ϕ =

θ^{-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.

4.1.4. 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 Section 4.2 below.

4.2. Calibration under Uncertainty

The problem of 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. We call this approach Calibration under Uncertainty (CUU).

Recent research in the Bayesian statistics community has yielded advances in formal statistical methods that address Calibration under Uncertainty. One approach 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 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 [Trucano et al., 2006].

CUU is therefore a progression of thought that leads to an overlap of the concepts of calibration and validation. For example, the formalism discussed below 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 validation as well as calibration.

4.2.1. Gaussian Process of Model Discrepancy

KOH assume that the calibration inputs are supposed to take fixed but unknown values $\boldsymbol{\theta} = (\theta_{1\ldots}\theta_{q2})$. The output of the computer model when the variable inputs are given values $\mathbf{x} = (x_1, x_2, \ldots x_{q1})$ and when the calibration inputs are given values $\mathbf{t} = (t_1, t_2, \ldots t_{q2})$ is denoted by $\eta(\mathbf{x},\mathbf{t})$. KOH differentiate between the unknown value $\boldsymbol{\theta}$ of the calibration inputs which we wish to determine (calibrate) and a known particular set of their values, $\mathbf{t}$, which we set as inputs when running the model. 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 $\mathbf{z} = (z_1, z_2, \ldots z_n)^T$. 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 $N(0,\lambda)$. Assumption of a constant value of ρ implies that the underlying process $\zeta(\mathbf{x})$ is stationary.

In the approach recommended by KOH, 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 $\delta(\mathbf{x}) \sim N(m_2(\mathbf{x}), c_2(\mathbf{x}, \mathbf{x}'))$. KOH assume that the mean functions are: $m_1(\mathbf{x},\mathbf{t}) = \mathbf{h}_1(\mathbf{x},\mathbf{t})^T\boldsymbol{\beta}_1$ 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. Finally, KOH 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.

The details of calculating the full joint posterior distribution $p(\boldsymbol{\theta},\boldsymbol{\beta},\phi|\mathbf{d})$ are given in KOH; space does not permit reproducing them here. The important thing to note is that this joint posterior density is a Gaussian process, with a complex mean and variance structure. The covariance matrix of the posterior involves four "submatrices" which depend on the correlation structure of the individual Gaussian processes $\eta(\mathbf{x},\mathbf{t})$ and $\delta(\mathbf{x})$. The posterior distribution is not tractable 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 address this by fixing many of these parameters and use a two stage process, where they estimate the hyperparameters relating to the covariance matrix for the model term, c_1, separately and before estimating the hyperparameters relating to the covariance matrix of the discrepancy term, c_2.

We are investigating the feasibility of using a GP formulation such as provided by KOH as a practical calibration method for engineering design problems. Katherine Campbell of LANL has also looked at KOH's work with an emphasis on implementation [Campbell, 2002]. She concluded that information about model quality gained through the formulation of a model discrepancy term could be useful, but that a user should be careful in situations of limited observational data and "avoid exaggerating the contribution of the Bayesian updating process." Higdon et. al [2004] have applied the idea of Gaussian process modeling in calibration to a number of engineering design problems. Their work is highly recommended for a better understanding of this subject.

4.2.2. Modification of the Gaussian process model for discrepancy

We implemented a simple version of a Gaussian process model for discrepancy. 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:

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

Many of the model discrepancies that we have seen in practice have a linear trend, so 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, $\delta(\mathbf{x}_i) = z_i$ - 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 = $\mathbf{h(x)}^T\boldsymbol{\beta}$, where β are the coefficients of the regression terms $\mathbf{h(x)}$. In this example, x is one dimensional, corresponding to time, and so $\mathbf{h(x)}^T$ = [1 x] and $\boldsymbol{\beta}$ = $[\beta_o, \beta_1]$.
3. Calculate the mean and variance of the resulting model discrepancy term $\delta(\mathbf{x}_i)$', where $\delta(\mathbf{x}_i)' = \delta(\mathbf{x}_i) - \mathbf{h(x)}^T\boldsymbol{\beta}$. 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: $\delta(\mathbf{x}_i) = \delta(\mathbf{x}_i)' + \mathbf{h(x)}^T\boldsymbol{\beta}$, where $\delta(\mathbf{x}_i)' \sim N(0, K+\sigma^2 I)$. The covariance matrix K has entries K(x,x'):

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

4. Define the Gaussian process and estimate its parameters. 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 z, and C as the covariance matrix = $K+\sigma^2 I$, the log likelihood is:

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

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. The red solid line is the original delta term: $\delta(\mathbf{x}_i) = z_i$ - Code Output. The blue solid line is the regression equation fit to $\delta(\mathbf{x}_i)$, and the green solid line is the difference, $\delta(\mathbf{x}_i)' = \delta(\mathbf{x}_i) - \mathbf{h(x)}^T\boldsymbol{\beta}$. The green circles are the GP predicted points (which go through the $\delta(\mathbf{x}_i)$' data exactly, and revert to a zero mean process in the "prediction zone" where time is greater than 2000 seconds. Finally, the green dotted lines show the predicted variance of the GP, and the red dashed lines show the mean and 2 sigma limits on the prediction of the $\delta(\mathbf{x}_i)$ term as it is extrapolated out from 2000 to 3000 seconds. This example serves to show how the modeling of $\delta(\mathbf{x}_i)$ as a GP may be useful in terms of predicting model results at new locations, and understanding the confidence limits around those results.

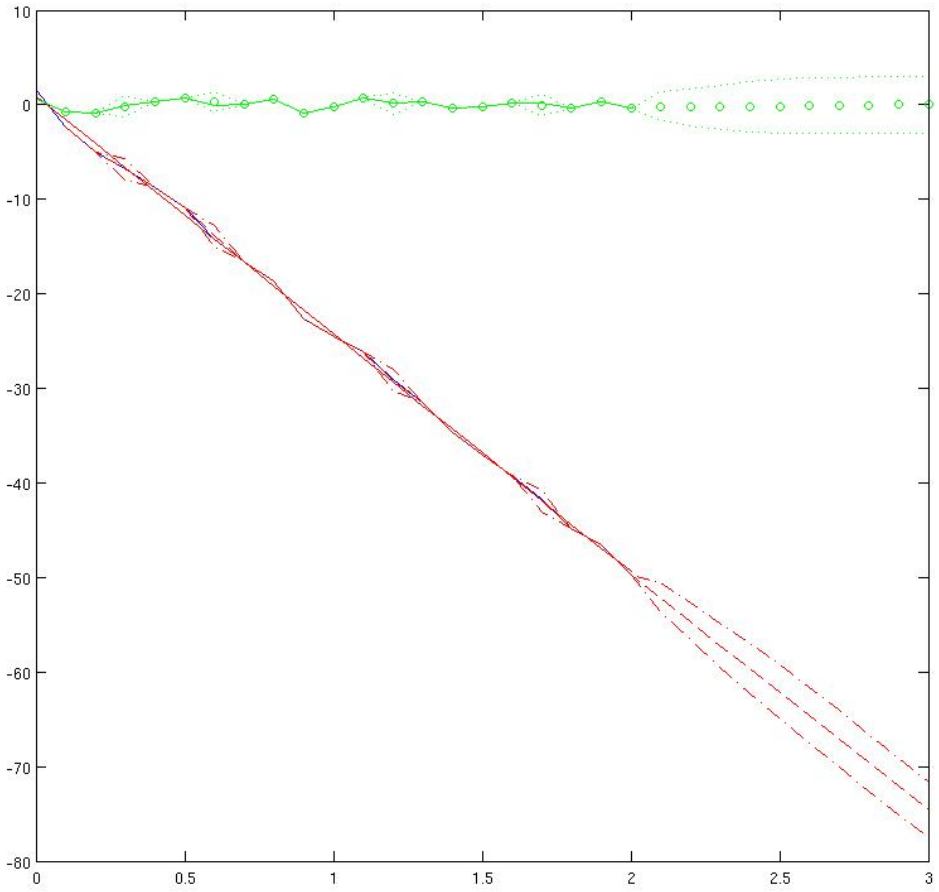

Figure 15. Mean delta vs. time (in 1000s seconds on X-axis), with GP prediction of delta

4.3. High/low fidelity autoregressive models

Many computational models of high physical fidelity are very expensive in terms of run time. In these cases, we would like to develop an approach to response surface modeling which allows us to construct a response surface based on some low fidelity function evaluations and update the coefficients governing that response surface with a few high fidelity function evaluations. This approach of correcting a low-fidelity response surface and updating it is used in some trust region [Eldred et al., 2004]. A variation on this approach has been developed by Kennedy and O'Hagan (2000), who propose constructing an autoregressive model where a higher-fidelity code output is assumed to be an autoregressive function of the lower fidelity code output. Huang et. al (2004, 2005) have expanded on Kennedy and O'Hagan's approach and we have looked at their implementation in detail. The overall idea for multi-fidelity models using a Bayesian autoregressive approach makes the following assumptions:

- Different levels of the same code are correlated in some way.
- The codes have a degree of smoothness in the sense that output values for similar inputs are reasonably close.
- Prior beliefs each level of code can be modeled using a Gaussian process.

We choose a notation similar to Huang's. If there are l levels of code, $l = 1, \ldots, m$, the assumption is that:

$$f_l(\mathbf{x}) = f_{l-1}(\mathbf{x}) + \delta_l(\mathbf{x})$$

where $\delta_l(\mathbf{x})$ is independent of $f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_{l-1}(\mathbf{x})$. This means that every level of code differs by the previous level by some delta function. In KOH (2000), they assume a slightly more complex autoregressive function:

$$f_l(\mathbf{x}) = \rho_{l-1} f_{l-1}(\mathbf{x}) + \delta_l(\mathbf{x})$$

The delta term $\delta_l(\mathbf{x})$ is meant to model the "systematic error" of a lower-fidelity system, (l–1), as compared to the next higher-fidelity system, l. $\delta_l(\mathbf{x})$ is usually small in scale as compared to $f_l(\mathbf{x})$. In KOH, both the $\delta_l(\mathbf{x})$ and $f_l(\mathbf{x})$ terms are modeled as Gaussian processes.

The major difference between what we have done and what Huang has done is that we have modeled the mean of the GP with a regression term and he has modeled it as a constant. Another difference is that we estimate the GP for the lower level model, $f_1(\mathbf{x})$, separately from $\delta_1(\mathbf{x})$. Finally, there is an issue of "matching" the models at the points. Both KOH and Huang evaluate the model at the same data points (the same **x** values). We evaluated the low and high fidelity data both at the same points to construct the delta term and at different points.

4.3.1. Implementation of Autoregressive GP Model

In kriging, the true, unknown response is assumed to be the sum of a linear model, a term representing the systematic departure (bias) from the linear model, and noise (Cressie 1993). The approaches above basically involve a kriging approach for the delta terms:

$$\delta_l(\mathbf{x}) = \mathbf{b}_l(\mathbf{x})^{\mathrm{T}} \boldsymbol{\beta}_l + Z_l(\mathbf{x}) + \varepsilon_l \qquad (l = 1, 2, \ldots, m)$$

where $\mathbf{b}_l$ and $\boldsymbol{\beta}_l$ are the basis functions and coefficients, respectively, of the linear model. Z_l is the systematic departure and ε_l is the random error. Z_l, is modeled as a zero-mean stationary Gaussian process. Huang, Giunta (1998), and others often assume a constant term for the basis.

The covariance between two points $\mathbf{x} = (x_1, \ldots x_d)$ and $\mathbf{x}' = (x'_1, \ldots x'_d)$ for the $\delta_l(\mathbf{x})$ function is:

$$\mathrm{cov}\left[\delta_l(\mathbf{x}), \delta_l(\mathbf{x}')\right] = \sigma^2_{Z,l} \exp\left[\sum_{j=1}^{d} -\theta_{l,j}(x_j - x'_j)^2\right]$$

where $\sigma^2_{Z,l}$ is the variance of the stochastic process, and $\theta_{l,j}$ is a "roughness" parameter associated with the dimension j. A larger $\theta_{l,j}$ implies a higher "activity", or lower spatial correlation, within the dimension j.

In our initial implementation, we first estimated a Gaussian process for $f_1(\mathbf{x})$, then estimated $\delta_1(\mathbf{x})$, then summed the two results to obtain $f_2(\mathbf{x})$:

$$f_2(\mathbf{x}) = f_1(\mathbf{x}) + \delta_2(\mathbf{x})$$

4.3.2. High/low fidelity autoregressive model results

The computational model of interest is the three-dimensional explicit transient dynamics code Presto. Presto is a Lagrangian Finite Element code developed at Sandia National Laboratories. The application here focuses on the mechanical deformation of a weapon at impact. This example has two levels of

fidelity: a low fidelity model of a weapon with approximately 10K finite elements, and a high fidelity model with approximately 50K elements and more detail in the modeling of internal structural elements.

As part of a preliminary investigation, we performed an orthogonal array (OA) parameter study on the model. This allowed us to identify the important parameters in our model. In the following discussion, x is an eight dimensional input space. We ran both the low and high fidelity models at 13 points in the parameter space. These points are shown in Table 2. The output of the low and high fidelity models is displacement. The displacement predictions from the computational codes are denoted as f_{1TRUE} and f_{2TRUE} to differentiate them from the Gaussian process estimates of the low and high fidelity results, which are $f_1(x)$ and $f_2(x)$, respectively. The code output is shown on Table 2 as well. We normalized the output. You can see that the low fidelity code predictions of displacement are larger than the high fidelity code predictions. It is this difference, the "delta term," that we are trying to estimate with a Gaussian process. Then, we will use the Gaussian process to predict what the delta term will be in the case of 6 new points which have values in the X parameters that are outside the domain given in Table 2.

X1	X2	X3	X4	X5	X6	X7	X8	f1true	f2true
15	5	10	0.375	0.0065	2625	12500	4.6	12.86	10.87
20	10	10	0.75	0.013	2750	12500	4.6	13.98	12.04
15	10	15	0.375	0.013	2750	13000	4.6	13.51	11.58
20	5	15	0.75	0.0065	2750	13000	4.7	15.69	13.73
15	10	10	0.75	0.013	2625	13000	4.7	14.46	12.31
15	5	15	0.375	0.013	2750	12500	4.7	13.66	11.48
15	5	10	0.75	0.0065	2750	13000	4.6	13.38	11.18
20	5	10	0.375	0.013	2625	13000	4.7	16.09	14.04
20	10	10	0.375	0.0065	2750	12500	4.7	15.18	12.85
20	10	15	0.375	0.0065	2625	13000	4.6	15.24	13.37
15	10	15	0.75	0.0065	2625	12500	4.7	13.96	11.72
20	5	15	0.75	0.013	2625	12500	4.6	14.30	12.38
20	10	15	0	0	2500	12000	4.5	13.23	11.43

Table 2. Computational model results, f_{1TRUE} and f_{2TRUE}, as a function of eight input variables.

Figure 16 shows the low and high level model results as a function of the first input variable, X1.

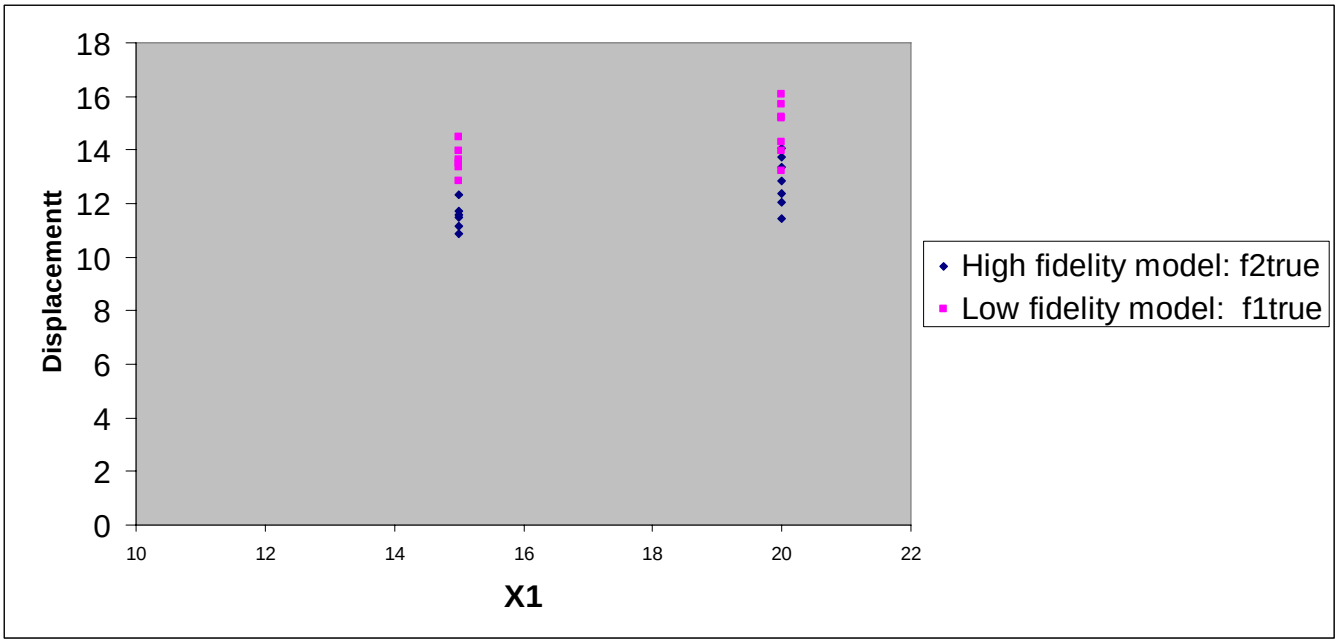

Figure 16. Displacement as a function of X1 for high and low fidelity codes, f_{1TRUE} and f_{2TRUE}.

The first step is to create a Gaussian process estimate of the low fidelity model. Then we create a Gaussian process estimate of the delta term. To obtain the GP estimate $f_1(x)$, we took the 13 points from the low-fidelity parameter study. The low fidelity GP model is: $f_1(x) = b_1(x)^T\beta_1 + Z_1(x) + \varepsilon_1$, where Z_1 is modeled as a zero-mean stationary Gaussian process. The coefficients of the regression term are estimated by a standard linear regression procedure, and we used maximum likelihood estimation of the covariance parameters governing the GP term Z_1.

After obtaining $f_1(x)$, we calculated the desired delta function between the high and low-fidelity models as: $f_2(x) - f_1(x) = \delta_2(x)$. That is, we took the actual high-level results from the 13 OA run, subtracted the GP estimate of the function, to obtain the desired values for $\delta_2(x)$. Then, we estimate the GP parameters based on the 13 input points in the table above. In this case, $\delta_2(x)$ is given as:

$$\delta_2(x) = b_2(x)^T\beta_2 + Z_2(x) + \varepsilon_2.$$

With the Gaussian process models of $f_1(x)$ and $\delta_2(x)$ developed, we now can use these to predict $f_2(x)$ at some new points. We chose six new points shown in Table 3. We ran the high fidelity model at these points to check the accuracy of our GP estimate, but we did NOT run the low fidelity model at these points. Instead, we used the GP estimate $f_1(x)$. If the low fidelity function evaluations were cheap enough computationally, one could use the code results for the low fidelity model and not use a GP approximation of the low fidelity model. Note that our approach has two Gaussian process terms added together to get the estimate of the high fidelity model: $f_2(x) = f_1(x) + \delta_2(x)$. However, in practice, it may be desirable just to create a Gaussian process model for the delta term if the "true" low fidelity calculations are available.

X1	X2	X3	X4	X5	X6	X7	X8	F2true	F2predicted	Error	Relative % Error
17	7	13	0.5	0.01	2700	12500	5	14.70	14.49	0.21	1.46
25	10	10	0.75	0.013	2800	12500	4.6	12.77	13.41	-0.63	4.96
15	10	15	0.9	0.02	2700	12000	4.8	11.18	11.46	-0.28	2.50
20	5	15	0.2	0.005	2800	14000	4.7	15.41	15.40	0.01	0.06
15	10	15	0.2	0.005	2800	14000	4.8	14.89	14.67	0.22	1.46
20	5	15	0.9	0.02	2700	12000	4.7	11.90	12.18	-0.29	2.40

Table 2. New X input points where we compare the GP prediction, $f_{2predicted}$, with the high fidelity model, f_2.

Figure 17 shows the "true" high level results for these six points, the GP predictions of these results, as well as the GP predictions of the delta term and the low level model results.

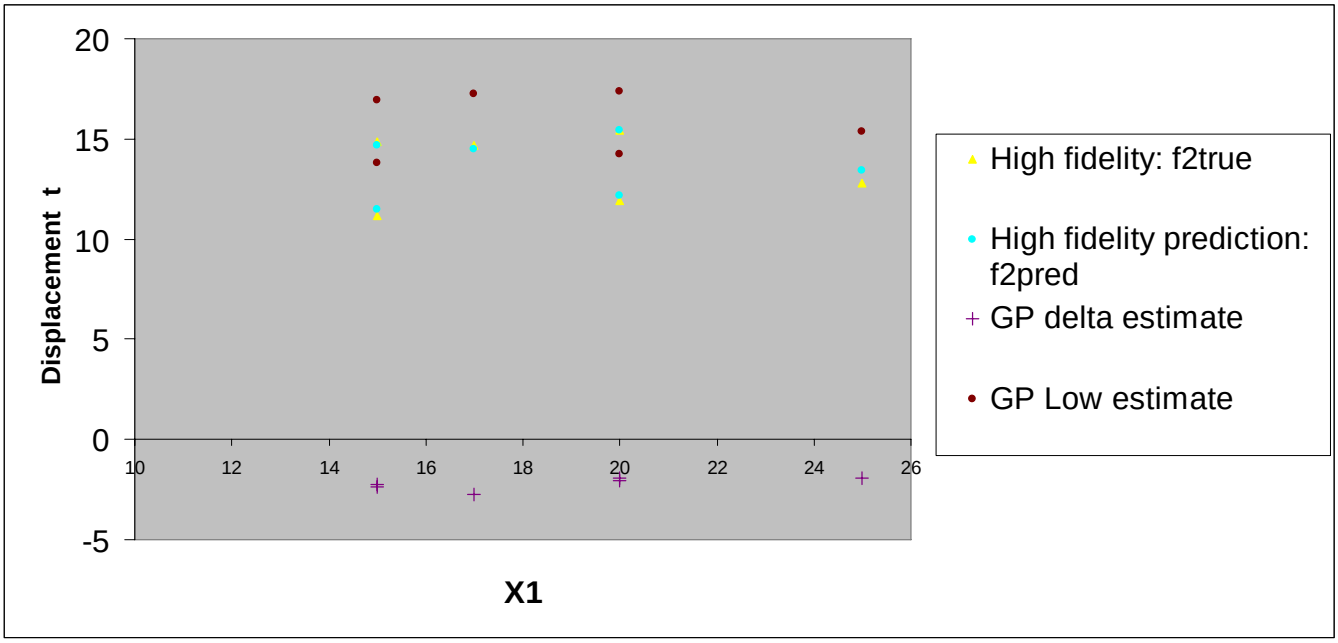

Figure 17. High fidelity prediction, $f_{2predicted}$, compared with the true high fidelity model, f_2.

Overall, we have very good agreement: the displacement predicted by the GP autoregressive model and the displacement obtained by the high fidelity "true" code calculation are very similar. The percentage error in the GP model is less than 5% in all six cases shown in Table 3 and is less than 3% in five of the cases. The largest error, for point 2, is due to the fact that this point represents a significant extrapolation of X1: the data upon which the GP models were built (the 13 points in Table 2) only involved X1 at values of 15 and 20, but this point has X1 at a value of 25. Note that we constructed Gaussian process models for the low fidelity model and for the delta term only based on 13 points in 8 dimensional space. Given that we are using these GP models to predict the output at 6 new points (where each new point involves extrapolation on at least one dimension), the predictions look good. Also note that the prediction of the low fidelity model gives higher estimates of displacement than the high fidelity model and the delta term is always negative. This is what we expect based on the original 13 data points.

Based on these results, we can say that an autoregressive approach based on GP models seems reasonable to pursue. We are currently addressing some issues relating to implementation of the covariance terms in the GP models. Using Gaussian process models for low fidelity results and an estimate of the delta term

between high and low fidelity to predict high fidelity results is a promising approach. This could have many applications, especially in optimization and uncertainty quantification problems.

5. Summary

This memo presented several applications of Bayesian approaches to problems in engineering design: reliability, regression modeling, calibration, and multi-fidelity models.

Bayesian reliability estimation is the easiest to implement, at least when one assumes the form of a binomial failure model with a beta distribution on the failure parameter. However, Bayesian reliability estimation has been primarily used in experimental regimes where one is acquiring large amounts of data on a continual basis. In computational modeling, there may be some applicability, but we do not see this as a primary focus area for the PRIDE LDRD, since we are not interested in "did it pass or fail" but by how much: what is the variability in the results and how robust is the design? Bayesian reliability does not directly address these issues.

The area of Bayesian regression is more promising, especially since surrogate models are now heavily used in optimization of design problems. We see the potential of generating posterior distributions on the regression coefficients (and thus generating "ensembles" or "families" of posterior regression models) for use in optimization. For example, Bayesian regression might be used with multi-start approaches and/or with trust region approaches. In trust region optimization, we could generate a family of posterior regression models in the first trust region, then follow each of these initial optima through the rest of the optimization process. Another approach is to generate one posterior distribution at the beginning of the optimization process based on a discrete set of sample points in the space, then update that distribution as additional points are taken.

The Bayesian statistics community has developed some ideas for calibration and also for multi-fidelity approaches based on Gaussian processes combined with Bayesian updating. Specifically, Gaussian processes are used to model the "code discrepancy" or model discrepancy term in calibration (the difference between the computational model results and experimental data). In multi-fidelity modeling, a "delta" term is used to correct a lower fidelity model to match or approximate a higher fidelity model. The delta term is also approximated with a Gaussian process. In both the calibration and multi-fidelity case, the terms governing the Gaussian process (e.g., the parameters of the covariance matrix) are "updated" using a Bayesian approach. We have found that use of Gaussian process models requires a good understanding of the method itself and an understanding of the problem in enough detail to normalize parameters, identify reasonable covariance parameters, etc. The methods are not "black-box" methods that can be used without some statistical understanding. However, GPs offer the ability to account for uncertainties as well as provide an estimate of a delta or discrepancy term. That is why they are very useful in design problems and why we think they have particular applicability to optimization under uncertainty. Our preliminary research has shown that a multi-fidelity model, where a high fidelity simulation is approximated by a lower fidelity simulation plus a GP delta term, is viable. A GP provides a reasonable functional form for the delta term; a GP approach can greatly help reduce the number of high-fidelity function evaluations necessary; and the GPs have good prediction capabilities. For these reasons, we are excited about using the GP/Bayesian approach for design optimization problems under uncertainty.

6. References

6.1. Books

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.

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

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

Gelman, A. J. B. Carlin, H. S. Stern and D. B. Rubin (2004), *Bayesian Data Analysis*, 2nd edition. 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.

Jaynes, E.T. (2003). Bretthorst, GL, editor. *Probability Theory: The Logic of Science*. Cambridge University Press.

Lindley, D.V. (1965). *Introduction to Probability and Statistics from a Bayesian viewpoint.* Cambridge University Press, Cambridge.

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

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

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.

6.2. Papers

6.2.1. Bayesian Models / Bayesian Calibration

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

Ferson, S. 2005. *Bayesian methods in risk assessment*. Applied Biomathematics Technical Report, available on-line at http://www.ramas.com/bayes.pdf.

Higdon, D., Kennedy, M., Cavendish, J., Cafeo, J., and R. D. Ryne. "Combining Field Data and Computer Simulations for Calibration and Prediction." *SIAM Journal on Scientific Computing*. **26**, 448.466.(2004)

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)

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.

Trucano, T.G., L. P. Swiler, T. Igusa, W. L. Oberkampf, M. Pilch, Calibration, Validation, and Sensitivity Analysis: What's What." Accepted for special issue of *Reliability Engineering and System Safety*, to be published in 2006.

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

6.2.2. Bayesian Optimization

Pelikan, M., D. E. Goldberg, and E. Cantú-Paz. BOA: The Bayesian Optimization Algorithm. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99), 1:525-532, 1999.

Pelikan, M., D. E. Goldberg, and E. Cantú-Paz. BOA: The Bayesian Optimization Algorithm, Population sizing, and Time to Convergence. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000), pp. 275-282, 2000.

6.2.3. Response Surface Modeling/Robust Design

Myers, R.H., Y. Kim, and K.L. Griffiths. "Response Surface Methods and the Use of Noise Variables." Journal of Quality Technology, Vol. 29, No. 4, 1997.

Khattree, R. " Robust Parameter Design: A Response Surface Approach." Journal of Quality Technology, Vol. 28, No. 2, 1996.

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).

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.

Vining, G.G., and R.H. Myers. "Combining Taguchi and Response Surface Philosophies: A Dual Response Approach." *Journal of Quality Technology*, Vol. 22, No. 1, 1990.

6.2.4. Surrogate Modeling

Huang, D., T.T. Allen, W. I. Notz, and R. A. Miller, "Sequential Kriging Optimization Using Multiple Fidelity Evaluations", submitted to Structural and Multidisciplinary Optimization (2004).

Huang, D., Allen, T. T., Notz, W. I., and Zheng, N., "Global Optimization of Stochastic Black-Box Systems via Sequential Kriging Meta-Models", accepted by the *Journal of Global Optimization* (2005).

Eldred, M.S., Giunta, A.A., and Collis, S.S, "Second-Order Corrections for Surrogate-Based Optimization with Model Hierarchies," paper AIAA-2004-4457 in *Proceedings of the 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Albany, NY, Aug. 30 - Sept. 1, 2004.

Jin, R., Du, X, and W. Chen. "The use of metamodeling techniques for optimization under uncertainty." In Proceedings of the ASME 2001 Design Engineering Technical Conference, DETC01.

Kennedy, M. C. and A. O'Hagan. "Predicting the output from a complex computer code when fast approximations are available." *Biometrika*, **87**, pp. 1-13. 2000.

6.2.5. Gaussian Processes

Booker, Andrew. "Well-conditioned Kriging Models for Optimization of Computer Simulations." *Technical Document Series, M&CT-TECH-002,* Phantom Works, Mathematics and Computing Technology, The Boeing Company, Seattle, WA, 2000.

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

Lee, H., R. Gramacy, and W. Macready. Parameter Space Exploration With Gaussian Process Trees *Proceedings of the International Conference on Machine Learning*, 2004, pp. 353-360.

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).

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

Swiler, Laura P, *"Gaussian Processes in Response Surface Modeling,"* Conference Paper, Society of Experimental Mechanics IMAC Modal Analysis Conference 2006, January 2006

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

6.3. 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/
http://www.shef.ac.uk/~st1ao/1b.html

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

7. Distribution

1	MS0370	Brian Adams	01411
1	MS0370	Roscoe Bartlett	01411
1	MS1110	Daniel Dunlavy	01411
1	MS0370	Michael Eldred	01411
1	MS0370	David Gay	01411
1	MS0370	Patrick Knupp	01411
1	MS0370	Scott Mitchell	01411
4	MS0370	Laura P. Swiler	01411
1	MS0370	Steve Thomas	01411
1	MS0370	Timothy G. Trucano	01411
1	MS0370	Bart Van Bloemen Waanders	01411
1	MS1110	Suzanne Rountree	01415
1	MS1110	Robert Heaphy	01415
1	MS0847	Richard Field	01526
1	MS0828	Kevin Dowding	01533
1	MS0828	Anthony Giunta	01533
1	MS0828	Richard G. Hills	01533
1	MS0828	William Oberkampf	01533
1	MS0828	Martin Pilch	01533
1	MS0828	Vicente Romero	01533
1	MS0748	David G. Robinson	06861
1	MS0829	Brian Rutherford	12337
1	MS9159	Patricia D. Hough	08962
1	MS9159	Monica Martinez-Canales	08962
1	MS9042	Michael L. Chiesa	08774
1	MS9950	Randolph R. Settgast	08774
1	MS0123	LDRD Office	01011
2	MS9018	Central Technical File	08945-1
2	MS0899	Technical Library	04536

External Distribution:

Scott Ferson
Applied Biomathematics
100 North Country Road
Setauket, NY 11733-1345

Prof. Herbert Lee
UC Santa Cruz, School of Engineering,
1156 High Street
Santa Cruz, CA 95064