SANDIA REPORT
SAND2006-7669
Unlimited Release
Printed November 2006

Penetrator Reliability Investigation and Design Exploration: From Conventional Design Processes to Innovative Uncertainty-Capturing Algorithms

Monica L. Martinez-Canales, Laura P. Swiler, Patricia D. Hough, Genetha A. Gray, Michael L. Chiesa, Robert Heaphy, Steve W. Thomas, Timothy G. Trucano, Herbert K. H. Lee, Matt Taddy, and Robert B. Gramacy

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 Report 2006-7669
Unlimited Release
Printed November 2006

Penetrator Reliability Investigation and Design Exploration: From Conventional Design Processes to Innovative Uncertainty-Capturing Algorithms

Monica L. Martinez-Canales[1], Laura P. Swiler[2], Patricia D. Hough[1], Genetha A. Gray[1], Michael L. Chiesa[3], Robert Heaphy[4], Steve W. Thomas[2], Timothy G. Trucano[2], Herbert K. H. Lee[5], Matt Taddy[5], and Robert B. Gramacy[6]

[1] Computational Sciences and Mathematics Research Department, Sandia National Laboratories, P.O. Box 969, Livermore, CA 94451-9159
[2] Optimization and Uncertainty Estimation Department, Sandia National Laboratories, P.O. Box 5800, Albuquerque, New Mexico 87185-1318
[3] Engineering Mechanics Modeling and Simulation Department, Sandia National Laboratories, P.O. Box 969, Livermore, CA 94451-9042
[4] Discrete Algorithms and Mathematics Department, Sandia National Laboratories, P.O. Box 5800, Albuquerque, New Mexico 87185-1318
[5] Department of Applied Mathematics & Statistics, University of California, Santa Cruz
[6] The Statistical Lab, University of Cambridge.

Abstract

This project focused on research and algorithmic development in optimization under uncertainty (OUU) problems driven by earth penetrator (EP) designs. While taking into account uncertainty, we addressed three challenges in current simulation-based engineering design and analysis processes. The first challenge required leveraging small local samples, already constructed by optimization algorithms, to build effective surrogate models. We used Gaussian Process (GP) models to construct these surrogates. We developed two OUU algorithms using "local" GPs (OUU-LGP) and one OUU algorithm using "global" GPs (OUU-GGP) that appear competitive or better than current methods. The second challenge was to develop a methodical design process based on multi-resolution, multi-fidelity models. We developed a Multi-Fidelity Bayesian Auto-regressive process (MF-BAP). The third challenge involved the development of tools that are computational feasible and accessible. We created MATLAB® and initial *DAKOTA* implementations of our algorithms.

Acknowledgments

This project thanks the Sandia National Laboratories LDRD program, and the Computational and Information Sciences (C&IS) and Engineering Sciences (ES) Investment Areas, in particular, for their funding support.

This project also thanks the following people:

- Paul Booker for his input on the penetrator design process, with specific weight given to the experimental Pen-X program.
- Steve Thomas, Scott Mitchell, and Heidi Ammerlahn for serving as the project managers.
- Marty Pilch and Greg Thomas for championing this project.
- Randolph Settgast, an intern in the Engineering Mechanics Modeling and Simulation Department, wrote and tested some of the scripts needed to glue together the various simulation codes used in this project. Randy is now at LLNL.
- Herbert K. H. Lee, a professor of Bayesian statistics Department of Applied Mathematics & Statistics, University of California, Santa Cruz. Professor Lee's expertise given to this project was a result of a university contract funded by this LDRD project.
- Robert B. Gramacy, a former Ph.D. student of Professor Lee, provided his *tgp* code to this project. Bobby is now at The Statistical Lab, University of Cambridge.
- Matt Taddy, a current Ph.D. student of Professor Lee, conducted some of the optimization-statistics hybridization studies.

Preface

At the onset of the Sandia LDRD process in June 2003, in which ideas for this project were first proposed, there were major military operations occurring in Afghanistan (launched in 2001) and in Iraq (launched in 2003) that dominated the mainstream news outlets. Some of the news focused on the success and failures of U.S. military armaments in these operations. During this time, the U.S. Congress allowed the Department of Energy (DOE) and the Air Force Research Laboratory (AFRL) to design and test a Robust Nuclear Earth Penetrator (RNEP) to address observed shortcomings and failures during these operations.

The Afghanistan and Iraq military operations illuminated the shortcomings in our weapons design process as well. To combat rapidly-evolving conditions, we would need to do better than to refurbish older weapons for new missions. Simultaneously, Sandia started looking at ways to shorten its weapons design process from 2 years to 6 months under a program concept called *Rapid Small Build*.

The initial direction of this LDRD project was heavily influenced by the events of our times (circa 2001-2003) with a view towards rapid design using modeling and simulation. At the onset of this project, in October 2003, Sandia was also beginning to negotiate the V&V milestones associated with ASC-funded physics-based modeling and simulation efforts. By 2004, the pragmatic vision of the Sandia V&V program was influencing ASC code development as well as the algorithmic development within this LDRD project. That vision required computationally efficiency in uncertainty-accountable algorithmic development. At the end of FY2006, this project completed proofs-of-concepts of novel optimization under uncertainty algorithms.

Note: The RNEP program was essentially cancelled in 2005 [aip-152-2005].

Note: The *Rapid Small Build* concept also underlies current Sandia concepts *Rapid Prototyping* and *SBET*.

Table of Contents

Abbreviations & Acronyms

Sandia National Laboratories

AFRL	Air Force Research Laboratory
ASC	Advanced Scientific Computing
DOE	Department of Energy
HDBT	Hard and Deeply Buried Targets
ICC	Institutional Computing Cluster
LDRD	Lab-Directed Research and Development
M&S	Modeling and Simulation
Pen-X	Experimental Penetrator Program
PRIDE	Penetrator Reliability Investigation and Design Exploration project
RNEP	Robust Nuclear Earth Penetrator
SBET	Science-Based Engineering Transformation
SNL	Sandia National Laboratories
V&V	Verification and Validation

Statistical/Mathematical/Optimization

ANOVA	Analysis of Variance
FEM	Finite element model
GP	Gaussian Process
LHS	Latin hypercube sample
MCMC	Markov Chain - Monte Carlo
MVN	Multi-variate normal
OA	Orthogonal array
TGP	Treed Gaussian Process
OUU	Optimization under Uncertainty
SBO	Surrogate-Based Optimization

PRIDE

OS	Offset
AoA	Angle of Attack
TS	Target Strength
IV	Impact Velocity
CR	Cavity Radius
DispMax	Maximum earth displacement
TipY	Maximum lateral acceleration at the penetrator nose tip
TipMag	Maximum magnitude of total acceleration at the penetrator nose tip
AftY	Maximum lateral acceleration at the penetrator aft end
AftMag	Maximum magnitude of total acceleration at the penetrator aft end
MF-BAP	Multi-Fidelity Bayesian Autoregressive Process
OUU-LGP	Optimization under Uncertainty-Local Gaussian Process
OUU-GGP	Optimization under Uncertainty-Global Gaussian Process

Software

APPSPACK
CUBIT
DAKOTA
Presto
tgp Treed Gaussian Process Software

This page intentionally left blank.

1. Introduction

With decreasing resources and a real inability to physically test parts of the design space, engineering applications have been turning to modeling and simulation (M&S) as tractable alternatives to hand-design and field/laboratory testing. However, the computational overhead associated with conventional M&S approaches has limited their adoption.

This project focused on research and algorithmic development in optimization under uncertainty (OUU) problems based on earth penetrator (EP) designs. In scoping our efforts, we concentrated on addressing three main challenges in engineering design and analysis process while also taking into account the presence of uncertainty. The first challenge required leveraging small local samples, already constructed by optimization algorithms, to build effective surrogate models. The second challenge was to develop a methodical design process based on multi-resolution, multi-fidelity models. The third challenge involved the development of tools that are computational feasible and the implementation of design processes that are useful to the EP weapons community and beyond.

The strategy for the first year was to balance research in OUU and Bayesian formalism with practical design exploration. To that end, we successfully canvassed current penetrator design, explored Bayesian formalisms and other innovative optimization approaches, conceptualized a multi-fidelity penetrator design and developed a low-fidelity penetrator simulation, formalized design optimization formulations, engaged in a small design exploration study in which general linear model analysis techniques were used and understood performance measure choices on penetrator design.

Our second-year efforts were spent analyzing statistically-based parametric studies and developing reduced-order models based on Bayesian approaches for EP design. Regression models based on the most important variables were built for the ground displacement response and used in our Multi-Fidelity Bayesian Autoregressive Process (MF-BAP) implementation. Results from MF-BAP studies show that we can use cheaper lower-fidelity responses, together with a systematic model bias correction based on a Gaussian Process

(GP) emulator, to estimate mid-fidelity earth displacement responses (with a maximum 5% relative error). Using just the lower-fidelity model alone to predict mid-fidelity displacement, we obtain a maximum of 24% relative error on the parameter space tested. The lower-fidelity model plus the systematic model bias correction constitutes the reduced-order model of the higher-order models.

The third year efforts were influenced more heavily by the calls to the scientific computing community to aid efforts to make Sandia simulation-based design processes tractable. Therefore, to reduce the overall computing expenses associated with design optimization, we leveraged our experience with GPs by integrating them at both local and global levels of optimization algorithms to construct new OUU algorithms. To that end, we constructed two OUU algorithms using "local" GPs (OUU-LGP) and one OUU algorithm using "global" GPs (OUU-GGP). The development of an optimization algorithm that combines both global and local information is an area of future research.

We discuss our approach and garnered experiences in the remainder of this report.

2. Conventional Weapon Design Processes

2.1 Earth Penetrator Weapons in Practice

Earth penetrators, also known as "bunker-busters," are an important tool in our Defense arsenal [jig-alw, jig-pb] applicable to the defeat of hard and deeply-buried targets (HDBT); see Figure 1.

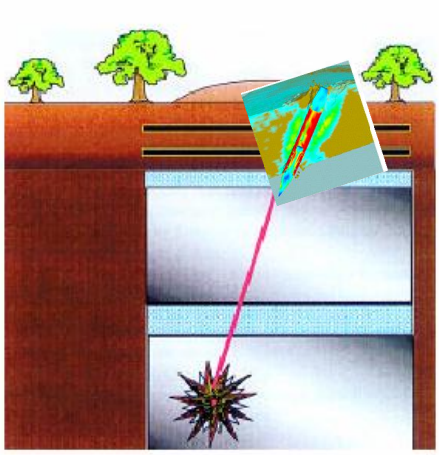

Figure 1: HDBT EP Application

A number of penetrators were used in the Operation Anaconda (March 2002) and Operation Iraqi Freedom (March 2003) offensives with mixed effects. In the Anaconda offensive, in particular, surface boulders and underground tunnels proved challenging targets for the EPs. In some cases, ensuring sufficient target penetration required sequentially staged EPs.

Occasionally, the EPs were damaged upon reaching their target. For example, at one of Sandia's test site, tail end of a penetrator snapped off after impart, see Figure 2. This post-delivery damage represents a common mission failure. Part of the problem was that the geology and target descriptions in the battle field corresponded to unexpected conditions in the operating range of the penetrator design.

Figure 2: Penetrator sticking, but not surviving.

2.2 Current Modeling Practices

Designing penetrators to survive the extremely high loads resulting from penetration of hard geologies at high impact velocities is typically a difficult and sometimes impossible task. In addition to the penetrator case surviving the impact, the electronics and nuclear package must also survive and the penetrator must achieve a depth great enough to preclude rebounding out of the target. While most of the aspects of the penetrator design goal are beyond our scope, our intent is to explore penetrator design processes involved with "stick and survive".

Historically, penetrators are designed using an ogival (or conical) nose with a cylindrical afterbody; [Gold-1999; Davi-1979]. Because penetrator design has previously been a long, arduous process using hand-tuning, the resulting designs (the B61-11, HP, and the Pen-X) have been sub-optimal and have been prone to failure due to conditions unanticipated during the actual design process. Therefore, there have been very few attempts to optimize the penetrator shape to reduce lateral loads (on the tail to prevent tail snap-off) and increase the probability of penetrator survival beyond worst-case scenarios.

2.3 Earth Penetrator Design Requirements

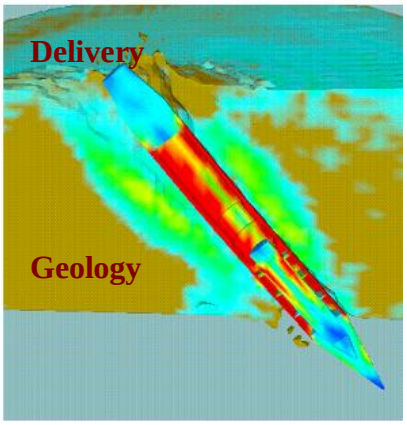

Figure 3: EP through delivery

Earth penetrators must "stick and survive" in targets with uncertain geology. Both "sticking" and "surviving" can be affected by delivery, geology, target descriptions, and other mission conditions. EPs must reliably function under variable delivery conditions such as velocity, angle of attack, or angle of impact. EPs must also reliably survive against geology uncertainties like boulders, varying rock strengths, and layered geologies and fissures corresponding to possibly severe medium discontinuities. Finally, the EPs must be able to penetrate targets of widely varying geometries, location, and construction. Failure to satisfy such design objectives will ultimately lead to failure to stick, survive, or both.

2.4 Earth Penetrator Design Optimization

Several factors, such as structural design, systems design, and intended penetrator target performance, must all be taken into account in the design process. Consider simple structural design problems. One optimization problem is to determine, for a given case weight, a case material distribution (or thickness profile) that maximizes depth of penetration, minimizes axial and lateral accelerations, ensures a stable trajectory, and ensures that the penetrator case and its contents survive impact. If nose shape is a driving design objective, the resulting additional optimization problem can be viewed as a shape optimization problem. If nose shape models strongly influence case material distribution, the resulting coupled optimization problem might be nonlinear or nonlinearly constrained. When a certain probability of penetrator survival is desired against uncertain target geologies, the resulting problem can be viewed as a chance-constrained optimization problem. Accepting that some target geologies will neutralize a certain number of EPs and a design objective is to minimize the probability of EP "death-by-boulder", the resulting optimization problem could be viewed as a stochastic programming problem. Few tools, if any, exist that can adequately handle these complex penetrator design problems, given the complexity of the problem and the modeling tools that must be used to explore it.

To model the dynamics associated with the penetration of the weapon into various geologies, we must use simulation models. An example of the simulation-based optimization process is shown in Figure 4.

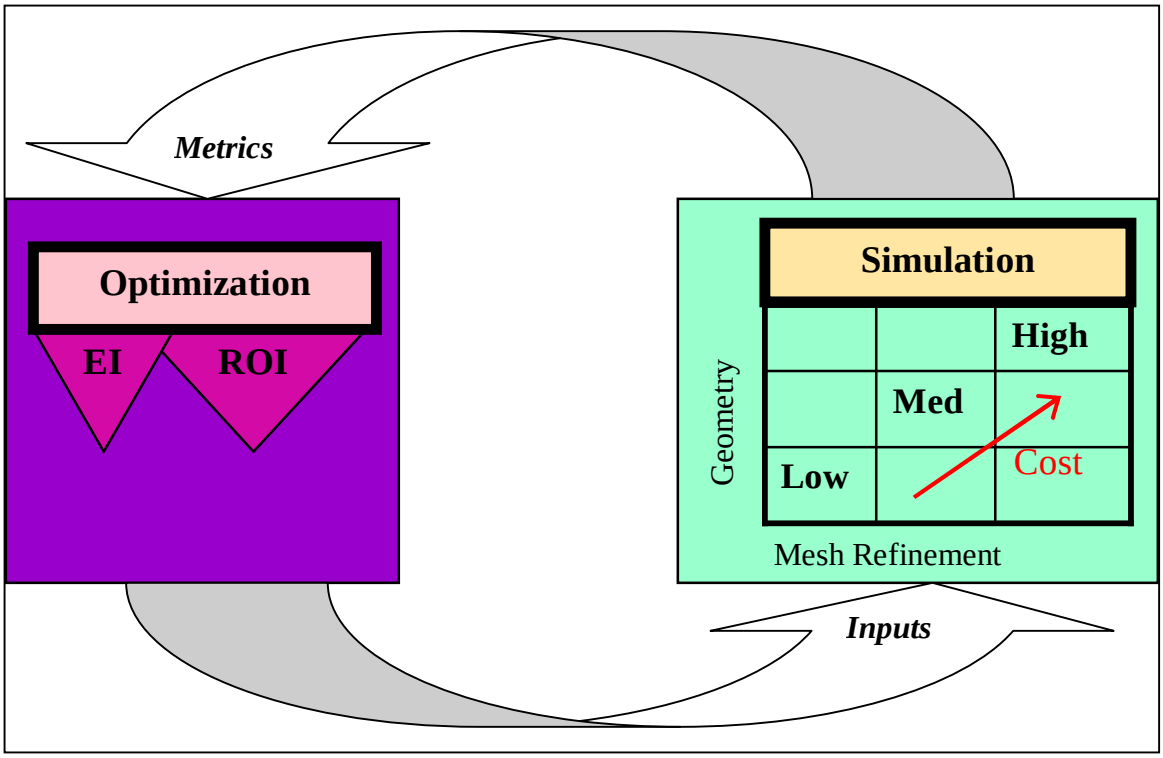

Figure 4: Simulation-Based Optimization Model. The optimization algorithms evaluate simulated designs based on some Expected Improvement (EI) or Return on Investment (ROI) metric. Based on the metric evaluation, the optimization algorithms generate simulation inputs that will lead to improved designs within a design search space. Proper usage of fidelity and related cost of the simulations must also be considered.

2.5 Earth Penetrator Design via PRIDE

We believe that the fundamental tool needed to support EP design is optimization under uncertainty (OUU). OUU refers to optimizing an uncertain quantity, such as maximizing the expected (mean) penetration depth or minimizing the 95^{th} percentile of axial acceleration on the tip. In the simplest formulation, one can use a standard optimization algorithm, but at each design point evaluated by the optimization routine, use a sampling method to sample over the uncertain variables and run the simulation many time at the samples value settings to calculate the uncertainty metric of interest. This approach is computationally expensive. That is one reason that surrogates (computationally cheaper models) are used. An example of the design optimization process incorporating the use of uncertainty and surrogates is shown in Figure 5.

OUU encompasses chance-constrained optimization, stacked multi-realization optimization to handle aleatory uncertainty, and incorporates design space exploration and uncertainty quantification to handle epistemic uncertainty. Aleatory uncertainty corresponds to irreducible (probabilistic) uncertainty and accounts for inherent variability such as in subsurface geologies. Epistemic uncertainty corresponds to reducible uncertainty (lack of knowledge) and accounts for model imprecision due to incomplete information such as in target descriptions or in the physics model themselves associated with differing representations (fidelity) of the model problem. We will only briefly touch upon epistemic uncertainty through abstractions of the response surface construction using Gaussian Process models.

While many approaches to OUU exist, none systematically and simultaneously uses uncertainty and sensitivity estimates, response-surface approximations (surrogates), statistical models, and multi-fidelity hierarchies in a way that allows the overall design process to be computationally credible (and under what metric) and does not require an excessive amount of full-fidelity simulations.

Our intent is to integrate optimization methods and metrics with uncertainty quantification techniques in a novel way that reduces the computational expense of the overall design process while increasing the amount of information propagation.

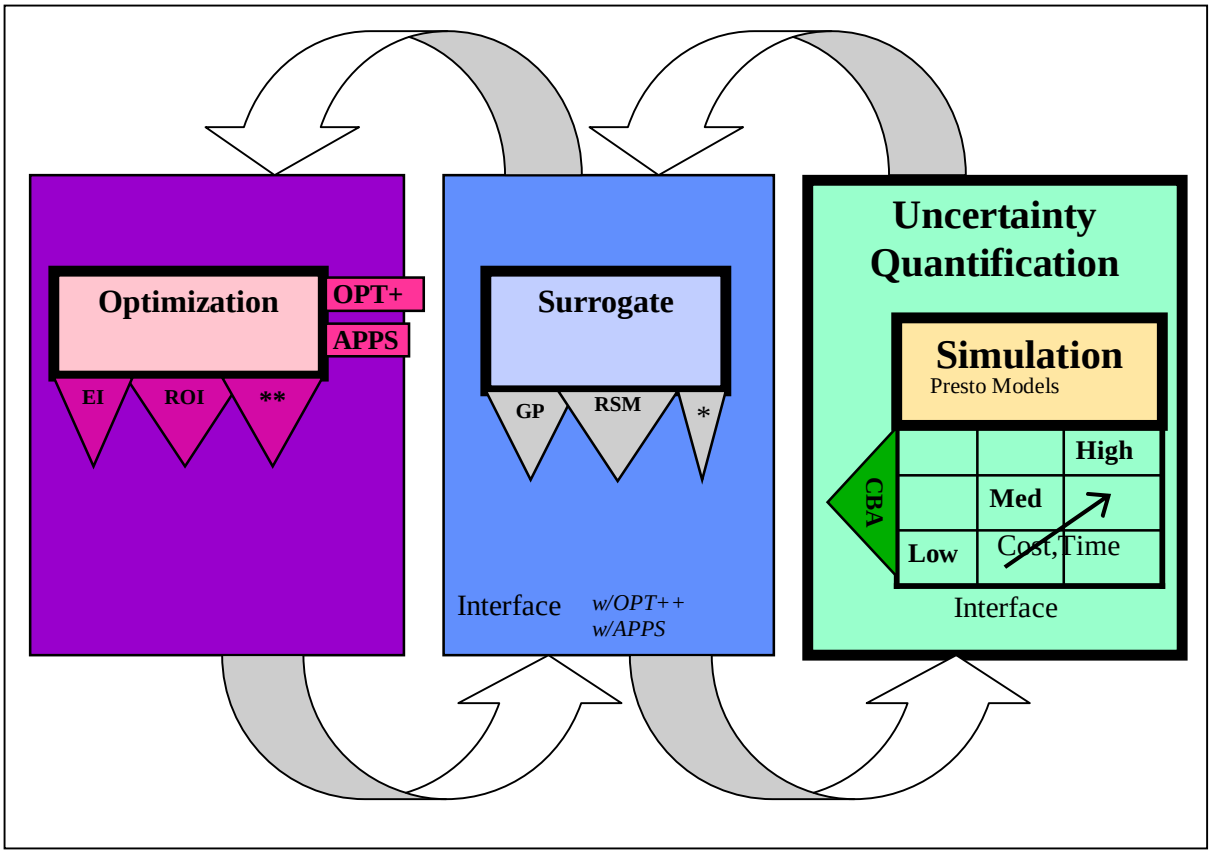

Figure 5: Our approach: Multi-Fidelity Optimization under Uncertainty

3. Test Problems

3.1 A Pen-X Model

This project initially focused on an enhanced penetration-type weapon, Pen-X, in which a large cavity is created ahead of the penetrator by a specifically designed shaped charge. Experimental and numerical studies have shown that a pre-formed cavity significantly reduces axial accelerations and allows penetration of much harder targets. This weapon type was chosen over other penetrator programs for its flexible design schedule and greater opportunities to directly affect the design processes. Another weapon option, the RNEP penetrator, has a relatively fixed design and offered little opportunity for shape optimization.

The overwhelming majority of penetration tests have been into undisturbed targets (i.e. no pre-formed hole); see, for example, Bateman et al. [Bate-1993]. Experience and modeling have shown that sharp pointed noses typically work best for this type of penetration problem. However, very little was known about what shape is optimal for penetration into pre-formed cavities. It is not intuitive that a sharp nose is best (although one could guess that a more rounded nose would be preferable). In addition, suggestions have been made that an egg-shaped body would reduce lateral loads while providing increased depth of penetration. The sensitivity studies we performed address these issues.

Multiple fidelity models of the earth penetrator were defined in terms of both mesh resolution and structural complexity. The low structural fidelity model (corresponding to30K Finite Element (FEM)) is based on a solid homogeneous body. The medium structural fidelity model includes internal blivets. The high-structural fidelity model includes an accurate model of the penetrator case and internals.

We described the penetrator structure as a cylindrical body, of length L2, capped by two ellipsoids of major radius L1 and L3, corresponding to the radii of the nose and tail, respectively, see Figure 6. The cylinder body radius R1 has an upper bound dictated by the

allowed weight and space in the delivery missile compartment. The major radii range in values that produce nose and tail shapes ranging from very blunt to very sharp. For low values of L2, the shape may even resemble an egg. The medium-fidelity model includes the low-fidelity penetrator case model description plus internal blivets. The high-fidelity model includes an accurate model of the penetrator case and internals. The penetrator was modeled as an elastic material and the target is modeled with a Mohr-Coulomb soil constitutive model. Target properties were chosen to represent material strengths ranging from low strength to high strength concrete. The penetrator and target models were based on axi-symmetric descriptions.

As an added measure of complexity, uncertain delivery and target conditions are also taken into account. Figure 6 shows the five uncertain variables that were included in the low-fidelity penetrator problem description. The angle of attack (AoA) is the deviation from the penetrator velocity and the penetrator central axis. The impact velocity (IV) is the velocity at which the penetrator impacts the ground. Offset (OS) is the horizontal deviation between the central axis of the penetrator and the central axis of the cavity. The offset occurs from a combination of an AoA and the separation of distance between the shaped charge and the penetrator. The cavity radius (CR) is the radius of the vertical shaft created by the shaped charge. Target strength (TS) is related to the strength of the target material. The smaller the target strength number, the harder the material. For later purposes, we will need the following assignments:

X1=L1, X2=L2, X3=L3, X4=OS, X5=AoA, X6=TS, X7=IV; X8=CR. (1)

We developed robust parametric mesh model in *CUBIT*, a 2D and 3D finite element mesh and geometry generation toolkit developed at Sandia National Laboratories, to mesh the penetrator and target geometry over the range of shape parameters. *CUBIT* version 9.1 was used in our studies.

We used Sandia's three-dimensional explicit transient dynamics (Lagrangian) finite element code *Presto* to model mechanical deformation at impact. We used eight-node hexahedral elements in these studies. While *Presto* has been ported to numerous high performance

parallel platforms at Sandia National Laboratories, the studies presented here were completed on the ICC Shasta, using *SIERRA* version 4.2.54 Beta and ACME version 2.2d libraries. However, because *Presto* is an explicit code, the time-step is based on the size of the smallest element in the mesh to satisfy the Courant stability condition, viz, the smaller the time-step, the longer the total run time. Therefore, for a model with a fine-enough mesh, the stability constraints on the time-step force the run time to be sufficiently long that not only do optimization studies become time-prohibitive, but even parameter studies at fine resolutions become intractable. Table 1 shows how the run time can increase as the mesh is refined. To that end, one of the goals of our penetrator parameter studies was to understand the extent to which cheaper lower fidelity models can be useful as surrogates of the expensive high fidelity models to reduce the overall cost of runs.

Table 1: Presto Run-time estimates

Fidelity	Mesh size	No. of Elements	Time step	Run time
Low	Δx	1Y	Δt	T
Medium	$\Delta x/2$	23 Y= 8Y	$\Delta t/2$	T(per 1Y per Δt)*(8Y per $\Delta t/2$) = 16T
High	$\Delta x/22$	(22)3Y=64Y	$\Delta t/22$	T(per 1Y per Δt)*(64Y per $\Delta t/4$)=256T

The responses of interest are the accelerations at the aft end and nose tip of the penetrator as well as maximum displacement of the ground due to penetrator impact. Accelerations are of interest because other penetrator studies have noted that high lateral accelerations result in tail slap which may result in the penetrator breaking, see Marin et al. [MCB-2005]. Also, electronic components, critical to the functional operation of the penetrator, may be mounted in the aft end where the lateral accelerations are typically highest. The simulated accelerations were filtered at 800 Hz using a four-pole Butterworth filter. This frequency was chosen to remove most of the high frequency response and allow a more accurate calculation of maximum values.

The depth of penetration is important for two reasons. First, a minimum penetration depth is required to prevent the penetrator from bouncing out of the target. Second, as the depth of penetration increases, the greater the energy is imparted to underground targets.

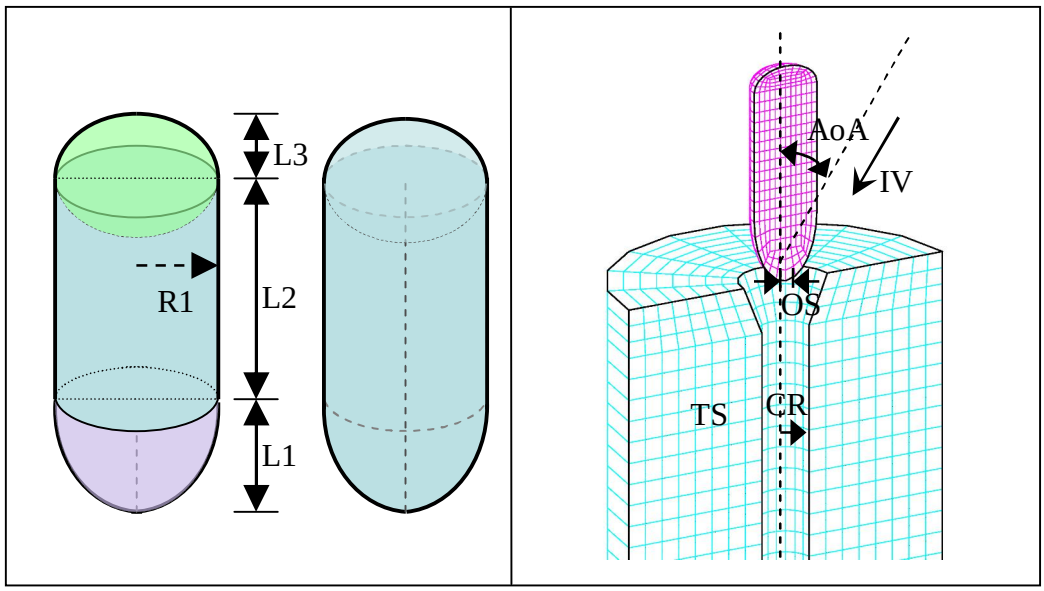

Figure 6: On the left, design variables L1, L2, L3, and radius constraint R1. On the right, a parameterized Finite Element Model of the penetrator (in pink) and the target showing the penetration shaft opening created by the Pen-X shaped charge. Also depicted on the right figure are the uncertain variables.

We fixed the penetrator body length and weight, and allowed the radii of the nose (L1) and tail (L3) to vary. We completed a small design exploration and pseudo-optimization study of the low-fidelity model acceleration response. We were interested in understanding the effect of the two variable parameters, L1 and L3, on the acceleration responses at the nose tip and aft end. We were also interested in finding the design that produces the minimum total acceleration.

A general linear model for an analysis of variance of the maximum total, vertical, and horizontal acceleration (at an aft node) versus both the nose and tail radii showed that the nose radius significantly influenced these responses more than the tail radius. The

generalized linear model was also able to detect that two designs in particular influenced the variance analysis. One of these designs corresponds to the case where the nose radius is maximally small and the tail radius is maximally large so that the penetrator nose is very blunt. This design has a minimal total acceleration at the aft node. Unfortunately, this particular penetrator design achieved poor penetration depth.

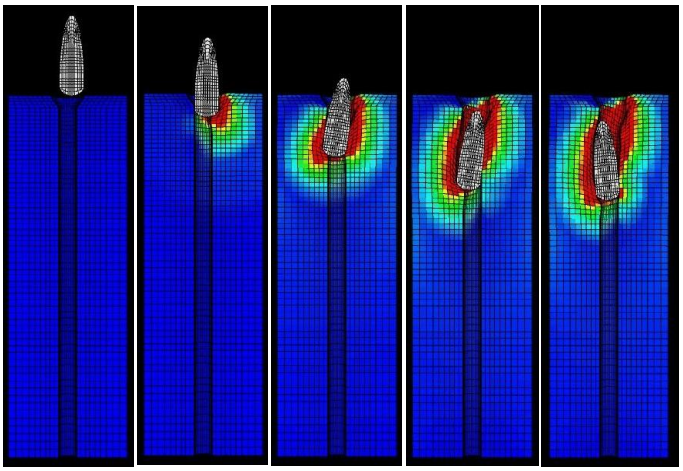

Figure 7: Small L1, large L3 design showing poor penetration.

The other design flagged by the generalized linear model analysis corresponded to the case where the nose radius is maximally large and the tail radius is maximally small so that the penetrator nose is very pointy. This penetrator design had good penetration (even though it bounced off the shaft walls) but had high lateral accelerations at the aft node. These lateral accelerations might lead to the penetrator breaking due to bending.

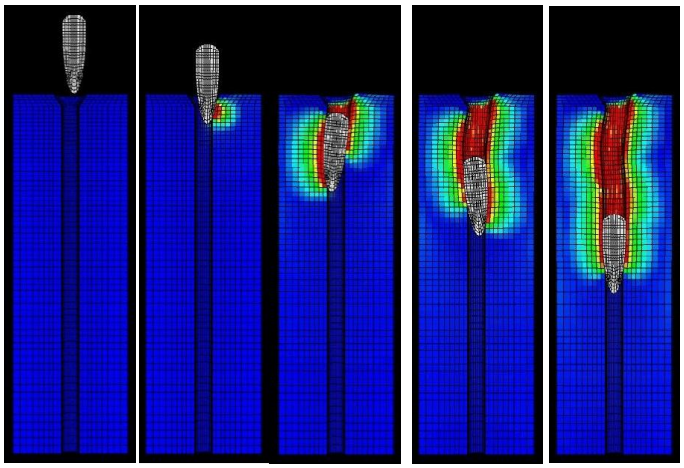

Figure 8: Large L1, small L3 design showing large lateral accelerations.

Additional details on parametric studies involving the Pen-X model can be found in Martinez-Canales et al [MSH-2007].

3.2 The Analytical Molar Test Function

To obtain rapid proof of concept in some of our later research efforts, we decided to also use an analytical function with properties similar to our penetrator application. That is, the

analytical function had to be multivariate and it had to also be characterized by multiple fidelities (low, high). It also had to have multiple minima/maxima, changing location and structure across dimensionality and fidelity.

One of the analytical functions that we chose for testing was created by H. K. H. Lee and M. Taddy (LT-2006)). The name of the function, molar test function, is derived from its resemblance to a dental molar in two dimensions. For the purposes of this study, we constrained both inputs to be bounded between -2 and 2.

The low-fidelity function is given by:

$$f_{lo}(x_1, x_2) = \left[-e^{-(x_1-1)^2} - e^{-0.8(x_1+1)^2} \right] * \left[e^{-(x_2-1)^2} + e^{-0.8(x_2+1)^2} \right]. \quad (2)$$

The equation for the high-fidelity function is:

$$f_{hi}(x_1, x_2) = \left[-e^{-(x_1-1)^2} - e^{-0.8(x_1+1)^2} + 0.05\sin\left(8(x_1+0.1)\right) \right] * \left[e^{-(x_2-1)^2} + e^{-0.8(x_2+1)^2} - 0.05\sin\left(8(x_2+0.1)\right) \right]. \quad (3)$$

The high fidelity function is the low fidelity function with an added noise term (the sine term). Both hi and low fidelity expressions possess multiple critical points.

In one dimension, the molar functions look like the following:

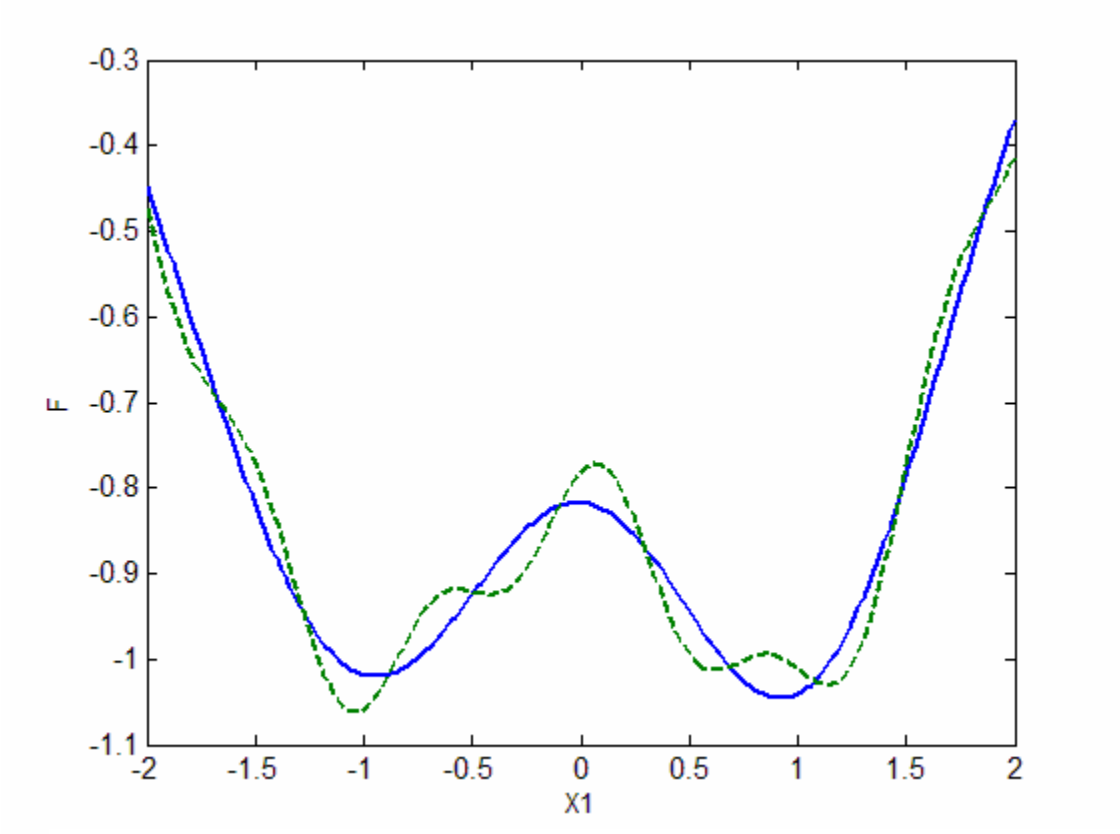

Figure 9: 1d Molar Function. Shown are the low (blue line) and high (green dashed line) fidelities.

In two dimensions, it is easier to look at the negative of the functions. Note that there are four local minima, one in each quadrant. These functions would be challenging to optimize using locally-convergent optimization methods. Without smart strategies, most locally-convergent optimization algorithms would "stick" to one of the local minima. The global minimum of f_{hi} is -1.124 at x_1 = -1.064 and x_2 = -1.064.

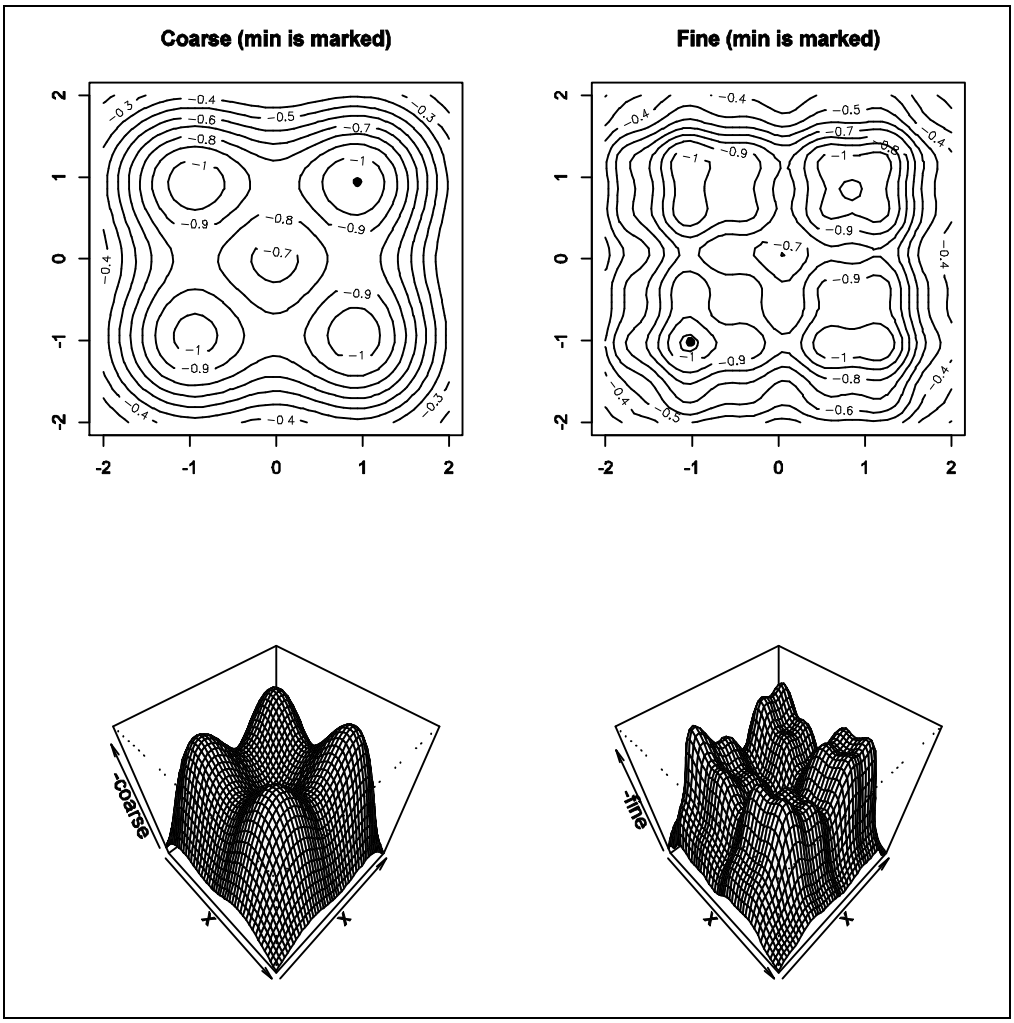

Figure 10: Plot of the 2D analytical molar function. The bottom row shows the negative image of function $f(x_1,x_2)$. The top row shows the contour lines of the functions beneath them. The true maximum is marked on the contour plots

4. Gaussian Processes is OUU

4.1 Gaussian Process Models

Our approach was to model deterministic computer output as realizations of a Gaussian Process (GP). Then, the set of GPs indexed by the simulation process parameters and their prior distributions represent our prior uncertainty regarding possible output surfaces. Newly observed or additionally generated data transforms the prior into a posterior under the Bayesian paradigm.

Following the standard practice in the [SWMW-1989; SWN-2003; FLS-2006], we start by modeling the output of the simulations as a realization of a Gaussian Process (GP). A GP is a stochastic process for which any finite collection of observations has a multivariate Gaussian (or Normal) distribution [Cres-1993]. The process is described by its mean function $\mu(\cdot)$ and its covariance function $C(\cdot,\cdot)$. For a set of locations $\mathbf{s}_1,\ldots,\mathbf{s}_n$ in S, we can write the distribution for the observations $\boldsymbol{x}$ as

$$\boldsymbol{x} \sim \mathbf{MVN}(\mu, \Sigma), \tag{5}$$

where $\boldsymbol{x} = (x(\mathbf{s}_1), \ldots, x(\mathbf{s}_n))$, $\boldsymbol{\mu} = (\mu(\mathbf{s}_1), \ldots, \mu(\mathbf{s}_n))$, and $\boldsymbol{\Sigma}$ is the variance-covariance matrix with elements $C(\mathbf{s}_i,\mathbf{s}_j)$. We typically take the mean function to be linear in location, although lower-order polynomials can be used, as can the assumption of a constant mean. If we make the typical assumptions of stationarity (C does not depend on location, but only on the distance between locations) and isotropy (the distance metric is spherically symmetric), we can simplify the expression for the covariance matrix to $C(\mathbf{s}_i,\mathbf{s}_j) = \lambda^{-1}\rho(d)$, where $\rho()$ is the correlation function and d is the Euclidean distance between $\mathbf{s}_i$ and $\mathbf{s}_j$. Then, ρ is taken to be a parametric family such as the Matérn class [Mate-1986], or the power family, which is what we use here:

$$\rho(d) = \exp[-(d / \theta)^{\gamma}], \tag{6}.$$

The choice of γ=1 gives the exponential correlation function, while γ=2 give the Gaussian correlation function which is the one we use hereafter. This correlation function is easily

expanded to account for anisotropy by allowing different range parameters θ_i for each dimension.

The model can be fit either by optimization in a maximum likelihood framework, or using Markov chain Monte Carlo in a Bayesian framework. Here we take the Bayesian approach, which allows incorporation of prior information about location or uncertainty in the parameter distributions. This allows full propagation of uncertainty, so that predictive uncertainty can be incorporated into the selection of new potential optimization points as described in Section 6.3.2. We use standard priors for the parameters, after standardizing the data [GL-2006]. More information on choices of priors and fitting of the model can be found in [HO-1994; BCG-2003; GL-2006].

4.1.1 Treed GP

In practice, many situations, including computer simulations, call for more flexibility than is reasonable under the assumption of stationarity. Here we use treed Gaussian process (TGP) models [GL-2006] to achieve non-stationarity. This approach partitions the input space and fits separate GPs within each partition. The partitions are fit simultaneously with the individual GP parameters using reversible jump Markov chain Monte Carlo, so that all parts of the model can be learned automatically from the data. The posterior predictive distribution thus takes into account uncertainty from the data, from the fitted parameters, as well as the fitted partitions, and is ideal for use within the oracle process in Section 6.3.2. Implementation is via the library TGP for the open source statistical package R. See, http://www.cran.r-project.org/src/contrib/Descriptions/tgp.html.

Because simulations often require significant computational time and resources, we are striving to reduce the number of runs needed by the optimization methods like *APPSPACK*. Moreover, since *APPSPACK* is a local optimization method, we are investigating ways to add robustness and introduce global properties. To accomplish these goals, we are using ideas from the design and analysis of computer experiments literature and using random functions to model the deterministic computer output function.

4.1.2 Two New Gaussian Process Implementations

By using the cheaper GP emulator as a surrogate of the complex model, we achieved reasonable computationally savings in parameter space exploration. By using the GP as a corrector between multiple model fidelities, not only did we observe a reduction in computation costs, but we also more closely emulated the engineering process. We implemented two versions of our Gaussian process, one in MATLAB, and one in *DAKOTA*. The MATLAB implementation allowed rapid proof-of-concept prototyping, development, and paved the way for an initial Dakota implementation. Both use essentially the same formulation; the only differences are the software implementation details. The MATLAB implementation is shown in Appendix A. In MATLAB, one can create a Gaussian process by calling the function *libgpfull*. Note that the function can be renamed if desired. The function *libgpfull* takes as input a set of observed data, Yobs, corresponding to a set of input points X. Note that Yobs should be an (n x 1) vector, and X should be an (n x p) vector, where n is the number of data points and p is the dimension of X. The call also requires defining a set of new values of X for which one wants predicted values. Xnew must be (m x p), where there are m new values of X. The call is then [Yest] = libgpfull(Yobs, X, Xnew), where Yest is an (m x 1) vector returning the Gaussian process predictions at the m new points. If desired, one can augment the output to be [Yest, sigest], where sigest is the standard deviation of the estimated values. Note that in our approach, we normalize the inputs and the output. We use the constrained minimization routine in MATLAB called *fmincon* to obtain the covariance parameters which maximize the likelihood function, giving us MLE estimators. The GP is a zero mean process with the following covariance:

$$\mathrm{cov}(\mathbf{x}, \mathbf{x}') = \sigma_Z^2 \exp\left[-\frac{1}{2} \sum_{d=1}^{p} -\theta_d (x_d - x_d')^2 \right] + \delta\sigma_j^2 \qquad (7)$$

Where σ_Z^2 is the process variance, σ_j^2 is a jitter term added to the covariance matrix to make it better conditioned, and δ is an indicator variable which equals zero if **x** and **x'** are different points and equals 1 if they are the same. In the MATLAB code, σ_j^2 is denoted by *gpVar*, and

σ_Z^2 is one of the parameters determined in the maximum likelihood estimation, along with the lengthscale parameters θ_d.

In DAKOTA, there is a family of approximation types, such as neural networks, regression models, etc. *GaussProcApproximation* is a class that has been added to create GP approximations. *GaussProcApproximation* is a class derived from the DAKOTA Approximation class. There are three main protected member functions in this class: find_coefficients builds the GP and finds the covariance parameters; get_value($\boldsymbol{x}$) returns the predicted value of the response for the input x; and get_variance($\boldsymbol{x}$) returns the variance of the predicted response at x. In the DAKOTA implementation, only the predicted value at one point can be returned at a time. Opt++ is the optimization software used to determine the optimal hyper-parameters governing the GP via maximum likelihood estimation. The GP may be used in surrogate construction by specifying the keyword gaussian_process under the surrogate type in the DAKOTA input specification file. For more details, see: http://endo.sandia.gov/DAKOTA/licensing/votd/htmldev/classDakota_1_1GaussProcApproximation.html .

Within DAKOTA, we used OPT++ to optimize the maximum likelihood function. We enabled this capability by taking advantage of an alternate constructor used in the RBDO methods. One addition to that constructor was adding the ability to request the use of finite-difference gradients in OPT++. An outstanding issue is the inability of the optimizer to optimize the first parameter defining the maximum likelihood function.

4.2 MF-BAP: Multi-fidelity Modeling with Gaussian Processes

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 approaches [EGC-2004]. A variation on this approach has been

developed by Kennedy and O'Hagan [KOH-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. We refer to [KOH-2000]. Huang et al. [HANM-2006; HANZ-2006] 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 an 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 for 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, $l = 1, m$, the assumption is that:

$$f_l(x) = f_{l-1}(x) + \delta_l(x) \tag{8}$$

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

$$f_l(x) = \rho_{l-1} f_{l-1}(x) + \delta_l(x) \tag{9}$$

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

A major difference between what we have done in this project and what Huang has done is that we estimate the GP for the lower level model, $f_1(x)$, separately from $\delta_2(x)$. In addition, there is an issue of "matching" the models at the points to construct the $\delta(x)$ term. Both KOH and Huang et al. 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.

In our initial implementation, we focus on two levels of fidelity, a low-fidelity model (level 1) and a high fidelity model, level 2. We first estimate a Gaussian process for $f_1(x)$ based on a sample set of model runs of the low fidelity model. Then, we estimate $\delta_2(x)$ based on some model runs of both the low and high fidelity models. Finally, we sum the two results to obtain an estimate for the high fidelity model, $f_2(x)$; that is: $f_2(x) = f_1(x) + \delta_2(x)$.

The Gaussian process, auto-regressive approaches outlined above assume that the true, unknown response is the sum of a linear model or constant term, a term representing the systematic departure (bias) from the linear model, and noise. The delta term is formulated as:

$$\delta_l(x) = b_l(x)^T \beta_l + Z_l(x) + \varepsilon_l \qquad (l = 1, 2, \ldots, m) \qquad (10)$$

where b_l and β_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 and others often assume a constant term for the basis. We feel that a regression term may be necessary to model the trend often seen in delta as a function of inputs.

4.3 Multi-fidelity Prediction Results Based on the Penetrator Problem

This penetrator test problem has two levels of fidelity: a low fidelity model with approximately 10K finite elements, and a high fidelity model with approximately 50K elements and more detail in the modeling of internal structural elements. The application here focuses on mechanical deformation.

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, $\boldsymbol{x}$ 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 in the columns X1-X8. 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 $f_1(\boldsymbol{x})$ and $f_2(\boldsymbol{x})$, respectively. The (normalized) code output is shown on Table 2 as well as graphed in Figure 11 as a function of the first input variable, X1. 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 given in Table 3 which have values in the X parameters that are outside the training domain.

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

	X1	**X2**	**X3**	**X4**	**X5**	**X6**	**X7**	**X8**	**f1true**	**f2true**
Point 1	15	5	10	0.375	0.0065	2625	12500	4.6	12.86	10.87
Point 2	20	10	10	0.75	0.0130	2750	12500	4.6	13.98	12.04
Point 3	15	10	15	0.375	0.0130	2750	13000	4.6	13.51	11.58
Point 4	20	5	15	0.75	0.0065	2750	13000	4.7	15.69	13.73
Point 5	15	10	10	0.75	0.0130	2625	13000	4.7	14.46	12.31
Point 6	15	5	15	0.375	0.0130	2750	12500	4.7	13.66	11.48
Point 7	15	5	10	0.75	0.0065	2750	13000	4.6	13.38	11.18
Point 8	20	5	10	0.375	0.0130	2625	13000	4.7	16.09	14.04
Point 9	20	10	10	0.375	0.0065	2750	12500	4.7	15.18	12.85
Point 10	20	10	15	0.375	0.0065	2625	13000	4.6	15.24	13.37
Point 11	15	10	15	0.75	0.0065	2625	12500	4.7	13.96	11.72
Point 12	20	5	15	0.75	0.0130	2625	12500	4.6	14.30	12.38
Point 13	20	10	15	0.00	0.000	2500	12500	4.5	13.23	11.43

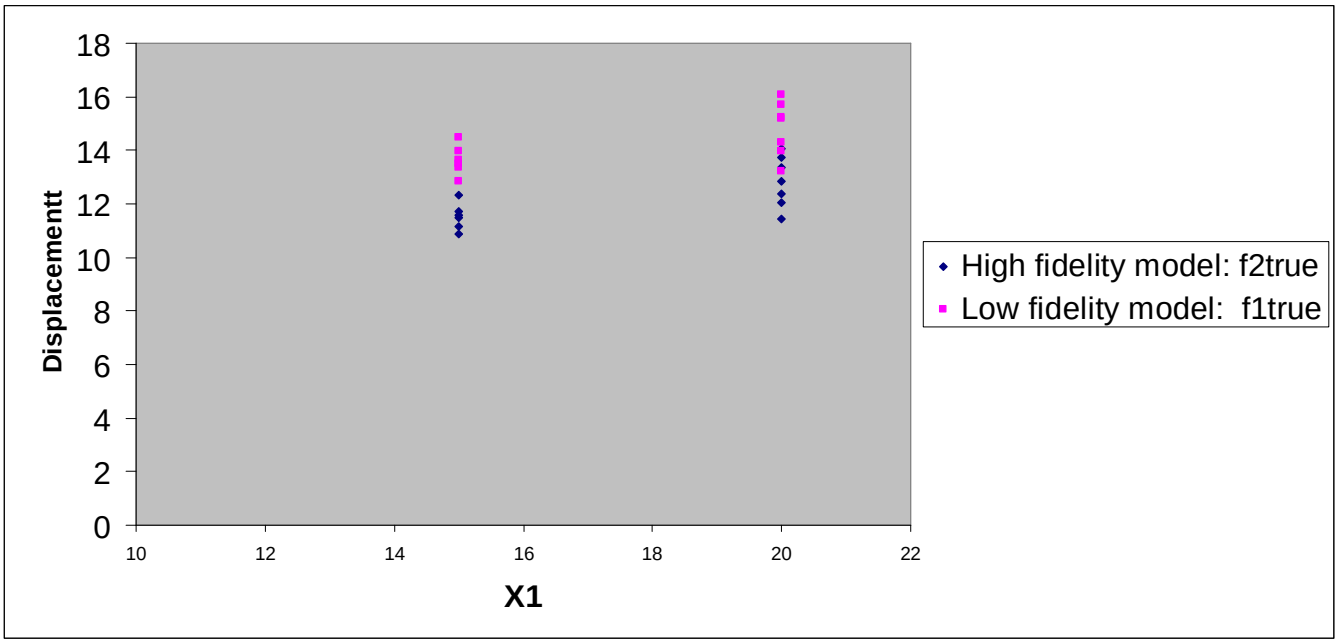

Figure 11: Displacement as a function of X1 for high and low fidelity codes, f_{1TRUE} and f_{2TRUE}.

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 a 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 estimated the GP parameters based on the 13 OA points in Table 2. 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 can now use these to predict $f_2(x)$ at some new points. We chose six new points shown in Table 2. 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.

Table 3: New X input points where we compare the GP prediction, $f_{2predicted}$, with the high fidelity model, f_2

	Point 1	Point 2	Point 3	Point 4	Point 5	Point 6
X1	17	25	15	20	15	20
X2	7	10	10	5	10	5
X3	13	10	15	15	15	15
X4	0.5	0.75	0.9	0.2	0.2	0.9
X5	0.01	0.013	0.02	0.005	0.005	0.02
X6	2700	2800	2700	2800	2800	2700
X7	12500	12500	12000	14000	14000	12000
X8	5	4.6	4.8	4.7	4.8	4.7
f2true	14.70	12.77	11.18	15.41	14.89	11.90
f2predicted	14.49	13.41	11.46	15.40	14.67	12.18
Error	0.21	-0.63	-0.28	0.01	0.22	-0.29
%Relative Error	1.46	4.96	2.50	0.06	1.46	2.40

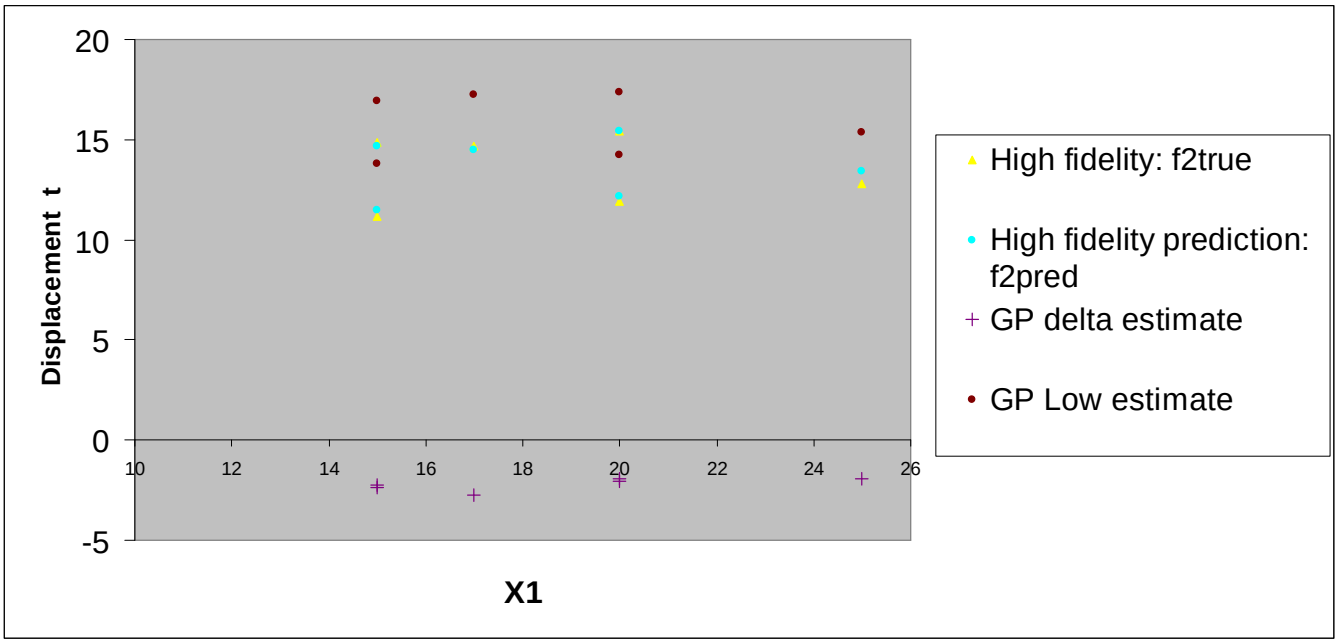

Figure 12: High fidelity prediction, $f_{2predicted}$, compared with the true high fidelity model, f_2

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. Figure 12 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.

Based on these results, it appears that 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. The next sections explain the use of the two-fidelity GP in trust-region optimization.

5. Aspects of Optimization

5.1 Surrogate-Based Optimization

Surrogate-based optimization has become a common approach for solving optimization problems that require the execution of a computationally expensive high-fidelity simulation in order to obtain objective function and constraint values. The essence of such approaches entails constructing low-fidelity models by fitting response surfaces to high-fidelity function values or by reducing the numerical or physical fidelity of the simulation. Optimization methods are then applied to these less expensive low-fidelity functions with periodic corrections from the high-fidelity simulation to ensure convergence to a local minimum of the high-fidelity function. Many variations of this type of approach can be found in the literature. Examples include [Book-2000; JSW-1998; Nash-2000]. In this work, we focus on the general trust-region framework presented by Alexandrov, Dennis, Lewis, and Torczon [ADLT-1998] and on the specific variation described by Eldred, Giunta, and Collis [EGC-2004].

The basic algorithmic structure of a trust-region surrogate-based optimization method is quite similar to that of the classical trust-region method described in standard references such as [DS-1983]. The process is iterative, with each iteration consisting of the following steps:

1. Construct surrogate and correction term for current iterate and trust region
2. Optimize corrected surrogate using appropriate method
3. Compute ratio of high-fidelity improvement to low-fidelity improvement at trial iterate
4. Accept/reject iterate and adjust trust-region size based on ratio value
5. Check for convergence

Alexandrov, et al. provide the theoretical framework under which convergence for this class of algorithms can be proved. Eldred, et al. explore different correction approaches in the context of that theoretical framework and investigate their effectiveness in practice.

5.2 Derivative-Free Optimization

Simulation-based optimization problems have an objective function whose evaluation requires the results of a complex simulation and are often characterized by a relatively small number of variables (i.e., $n < 100$). Moreover, derivatives often do not exist and/or are difficult to estimate. Therefore, derivative-free methods have become a useful method of solution.

5.2.1 The APPS Algorithm

In this work, we consider a derivative-free optimization method called Asynchronous Parallel Pattern Search (APPS) [HKT-2001; Kold-2004]. The APPS algorithm is part of a class of direct search methods which were developed primarily to address problems in which the derivative of the objective function is unavailable and approximations are unreliable [Wrig-1996; LTT-2000]. Pattern searches use a predetermined pattern of points to sample a given function domain. It has been shown that if certain requirements on the form of the points in this pattern are followed and if the objective function is suitably smooth, convergence to a stationary point is guaranteed [LT-1996; Torc-997; DLT-2000].

The majority of the computational cost of pattern search methods is the function evaluations, so parallel pattern search (PPS) techniques have been developed to reduce the overall computation time. Specifically, PPS exploits the fact that once the points in the search pattern have been defined, the function values at these points can be computed simultaneously [DT-1991; Torc-1992]. The APPS algorithm is a modification of PPS that eliminates the synchronization requirements. It retains the positive features of PPS, but reduces processor latency and requires less total time than PPS to return results [HKT-2001]. Implementations of APPS have minimal requirements on the number of processors and do not assume that the amount of time required for an objective function evaluation is constant or that the processors are homogeneous.

We consider the specific APPS algorithm as described in Kolda 2004 [Kold-2004]. It is a variant on generating set search as described in Kolda et al. [KLT-2003] and is provably convergent under mild conditions [KT-2003; KT-2004; Kold-2004]. Omitting the implementation details, the basic APPS algorithm can be simply outlined as follows:

1. Generate a set of trial points to be evaluated.
2. Send the set of trial points to the *conveyor* for evaluation, and collect a set of evaluated points from the conveyor. (The conveyor is a mechanism for shuttling trial points through the process of being evaluated.)
3. Process the set of evaluated points and see if it contains a new *best point*. If there is such a point, then the iteration is successful; otherwise, it is unsuccessful.
4. If the iteration is successful, replace the current best point with the new best point. Optionally, regenerate the set of search directions and delete any pending trial points in the conveyor.
5. If the iteration is unsuccessful, reduce certain step lengths as appropriate. In addition, check for convergence based on the step lengths.

A detailed procedural version of APPS is given in Gray and Kolda [GT-2004], and a complete mathematical description and analysis is available in Kolda [Kold-2004].

5.2.2 APPSPACK

The APPS algorithm described here has been implemented in an open source software package called *APPSPACK*. It is written in C++ and uses MPI [GL-1996; GLDS-1996] for parallelism. The details of the implementation are described in detail in Gray and Kolda [GK-2006]. There are both serial and parallel versions of *APPSPACK*, but to achieve the goals of this work, we are solely interested in the parallel version. *APPSPACK* has been successfully applied to problems in micro-fluidics, biology, groundwater, thermal design, and forging; see [GK-2006] and references therein.

APPSPACK performs function evaluations through system calls to an external executable, and no restrictions are placed on the language of this executable. This simplifies its execution and makes it amenable to customization. Of particular interest to us is the management of the function evaluation process. The procedure is quite general and merely one way of handling the process of parallelizing multiple independent function evaluations

and efficiently balancing computational load. This management system makes *APPSPACK* amenable to the hybridization technique discussed later.

6. Novel Optimization under Uncertainty Algorithms

6.1 Optimization under Uncertainty Algorithms

To reduce the overall computing expenses associated with design optimization, we leveraged further our experience with GPs by integrating them at both local and global levels of optimization algorithms to construct new optimization under uncertainty (OUU) algorithms. To that end, we constructed two OUU algorithms using "local" GPs (OUU-LGP) and one OUU algorithm using "global" GPs (OUU-GGP). The development of an optimization algorithm that combines both global and local information is an area of future research.

6.2 OUU-LGP

In the OUU-LGP approach, we used the GPs in a trust-region surrogate-based optimization method in two different ways within the trust region. In the first approach, we used the GP as a surrogate to the high-fidelity function. We implemented this approach in the DAKOTA framework, making use of both the existing trust-region (OPT++) surrogate-based optimization (SBO) infrastructure and our GP implementation. In the second approach, we used the GP model as a correction between multiple model fidelities. To circumvent some Dakota infrastructure limitations associated with multi-level surrogate constructions, we implemented and successfully tested this approach in MATLAB. Future work aims to complete this work within Dakota.

6.2.1 Trust-region surrogate-based optimization

In this work, we consider using a multi-fidelity Gaussian process model in the context of a trust-region surrogate-based optimization method. In particular, we use Gaussian process models in two related variations of the algorithm outlined above. In the first approach, we use the Gaussian process as a surrogate to the high-fidelity function. This approach has been

implemented in the DAKOTA framework and makes use of both the existing trust-region surrogate-based optimization infrastructure [appropriate DAKOTA/Giunta reference here] and the GP implementation described in Section4.1.2.

In the second approach, we demonstrate proof of principle in MATLAB since some required features are not yet available in DAKOTA. In this scenario, we use the Gaussian process model as a correction between multiple model fidelities. The implementation is similar in spirit to that in DAKOTA, but some of the specifics differ. We describe Steps 1-5 in Section 5.1 in more detail as they pertain to our method. The MATLAB file appears in Appendix B. In particular, Step 1 of the above algorithm consists of constructing a Gaussian process based on statistical sampling of the high-fidelity function and the low-fidelity within the trust region about the current iterate. The sampling is done within the more restrictive of the trust-region bounds and the bound constraints on the optimization problem. Based on these, two GP surrogates are constructed: one for the low fidelity model and one for the delta term, as described above. By summing the two GP surrogates, we form a high fidelity surrogate which is optimized in Step 2. Optimization of the high-fidelity surrogate is done with the MATLAB Optimization toolbox function *fmincom*. The *fmincon* function finds the minimum of a constrained nonlinear multivariable function using a Sequential Quadratric Programming (SQP) method. In this method, the function solves a quadratic programming (QP) subproblem at each iteration. An estimate of the Hessian of the Lagrangian is updated at each iteration using the BFGS formula, [Mat-2006]). Actual decrease is computed by taking the difference between the high-fidelity function value at the current iterate and the high-fidelity function value at the trial point predicted by the GP surrogate. The predicted decrease is computed by taking the difference between the high-fidelity function value at the current iterate and the GP surrogate value at the trial point. This ratio is then used to determine whether or not to accept the trial point and how to adjust the trust-region size. The thresholds are similar to those in the DAKOTA TRSBO method. There is no true convergence check (i.e. norm of gradient goes to zero) due to the fact that no gradients are used in this algorithm. Stopping criteria are minimum change in function value from one iteration to the next, minimum trust-region size (corresponding to a minimum step size), and the maximum number of iterations. All of these can be set by the user. Details can be found

in the MATLAB code listing in Appendix B. Note that for computational efficiency, we separated the Gaussian process calculation into two separate functions: *libgpfull* now just calculates the optimal covariance parameters as well as the inverse of the covariance matrix, and *gp_quick* takes these parameters as well as the original sample data and desired input location and provides an estimate of the output at that input location. By separating these two functions, we only perform the calculation of covariance parameters and covariance matrix once per trust region, and then use this information to calculate the expected value of the high fidelity function at numerous points within the trust region.

6.2.2 Multi-fidelity Optimization Numerical Results

Table 4 shows the results from the trust-region optimization of the two-level Gaussian process surrogate for the high fidelity function. Note that these results demonstrate that the trust-region optimization is working, but is converging to local minima depending on where the algorithm starts. This is what we expect, because the trust-region approach is a local optimization approach.

Table 4: Optimization results based on different starting points

Starting Point [x1,x2]	Optimal Point [x1,x2]	F_hi
[0,0]	[-1.0511, 0.8888]	-1.0564
[-1,-1]	[-1,-1]	-1.1194
[-2,-2]	[-1.0307, -0.9761]	-1.1174
[1,1]	[1.1369, 1.143]	-1.0609
[2,2]	[1.0493, 0.9878]	-1.0303

Further work remains to be done, testing the trust-region approach with other functions, specifically with functions in higher dimensions and with functions that exhibit a more pronounced difference between the low and high fidelity versions. In addition, comparisons of function evaluations for this approach vs. an approach of directly optimizing the high fidelity function need to be performed.

6.3 OUU-GGP

In the OUU-GGP approach, we used a Treed GP (*tgp*) algorithm [GL-2006] to enhance the pattern search optimization algorithm, *APPSPACK*. Within an oracle module separate from the search pattern, the *tgp* algorithm finds the predictive distribution for new sample points conditional on the points that have already been generated and evaluated by the search pattern. If the point recommended by *tgp* is a better point than those in the pattern, the search pattern continues from there; otherwise, it is discarded. When cleverly applied to our test problems, *APPSPACK-tgp* not only showed that the expected improvement statistic was close to zero everywhere in the parameter space, but was also able to detect the correct global minimum as opposed to a guaranteed local minima in the case of the unembellished pattern search process. We now review this work.

6.3.1 Oracles

In recent years, some optimization methods have introduced an *oracle* to predict points at which a decrease in the objective function might be observed. These points are specified in addition to the points generated by the optimization method itself. Analytically, an oracle is free to choose points by any finite process. (See [KLT-2003] and references therein.) The addition of an oracle is particularly amenable to a pattern search methods like APPS. The iterate(s) suggested by the oracle are merely additions to the pattern. Furthermore, the asynchronous nature of the *APPSPACK* implementation makes it adept at handling the evaluation of the additional points.

We used the *tgp* statistical model as our oracle as a means to add robustness and to introduce some global properties into *APPSPACK*. When the oracle is called, the *tgp* algorithm is applied to the set of evaluated iterates in order to choose additional candidate points. In other words, *APPSPACK* is still optimizing as normal, but throughout the optimization process, the iterate pairs $(x_i, f(x_i))$ are collected. Then, the *tgp* algorithm considers the current set of collected iterate pairs and recommends one or more new iterates. These new *tgp*-points are then evaluated to see if one of them is a new best point. If not, the point is merely discarded. However, if a *tgp*-point is a new best point, the *APPSPACK* search pattern continues from that location.

6.3.2 Expected Improvement Criteria

As mentioned above, the oracle treats the output of the simulations as realizations of a Treed Gaussian Process. The uncertainty about future computer evaluations can be quantified by finding the predictive distribution for new input locations conditional on the points that have already been evaluated. Since we now have a full probabilistic model for code output at new locations, any statistic depending upon this output is easily obtained.

The expected improvement at a point $\boldsymbol{x}$, $E[\min(f_{\min} - f(\boldsymbol{x}),0)]$, is a useful criteria for choosing new locations for evaluation. The paper by Jones et al. [JSW-1998] illustrates the use of this statistic in optimization. Since the improvement is a random variable, this criterion balances rewarding points where the output is highly uncertain, as well as where the function is generally predicted to be better than the present best point. Each time that the oracle is called, a number of candidate locations are generated from an optimal space filling design. The *tgp* model is then fit to the existing output, and the expected improvement is calculated at each candidate location. The points with highest expected improvement are passed back to the APPS algorithm for evaluation.

6.3.3 Theoretical Considerations

One of the main theoretical considerations in developing a hybrid method like the one described here is convergence. In this case, the APPS algorithm is provably convergent under mild conditions. This result can be leveraged since we incorporate the iterate(s) suggested by the Gaussian process as an oracle. Since the oracle points are given in addition to those generated by the pattern search, there is no adverse affect on the convergence.

Future work includes investigating improvement to the convergence as a result of the *tgp* oracle. Moreover, we hope to incorporate the findings of *tgp* into the *APPSPACK* stopping criterion. Currently, the primary stopping condition is based on the step length. This criterion was chosen because it can be shown that if the objective function is continuously differentiable, then the norm of the gradient (or an analogous measure of the constrained

measure of stationarity in the bound-constrained case) can be bounded as multiple of the step size [KLT-2003]. In other words, the steps only get smaller if the norm of the gradient is decreasing. Alternatively, *APPSPACK* offers two additional stopping criteria. One is based on whether or not the function has reached a specified threshold, and the other is defined in terms of the number of function evaluations. We would like to add the additional stopping condition that depends upon the expected improvement over the entire input space, a statistic that is calculated each time the oracle is called. This will ensure that the *tgp* oracle does not stop if there is high probability of finding a better location.

6.3.4 Numerical Results

We implemented our ideas and tested them against the molar function described in Section 3.2 with some promising results.

The version of *APPSPACK* enhanced by the *tgp*-oracle converges to f =1.126 at (x_1, x_2) = (-1.042, -1.058) which is the correct answer. Finding this solution took 63 total function evaluations, where 15 of those function evaluations resulted from points suggested by the *tgp* oracle. In Figure 13, the mean predicted output surface is illustrated in the left image and the expected improvement over the input space is shown on the right (all the green coloring). In the expected improvement image, the black dots are evaluated iterates generated by *APPSPACK* while the red dots are evaluated iterates chosen by the *tgp* oracle. Notice that the *APPSPAC*K-*tgp* hybrid method, at convergence, has generated a uniform expected improvement field over the space, that is, there were no points that the tgp oracle could choose to affect the improvement function. Further notice that the response surface generated from the collection of evaluated iterates, through convergence, maintains much of the second-order structure observed in the true function. This point is noteworthy because, as we already learned from our experiences with GPs, the response surface we building in an enhanced design optimization context is actually a cheaper alternative to the costly truth function or simulation.

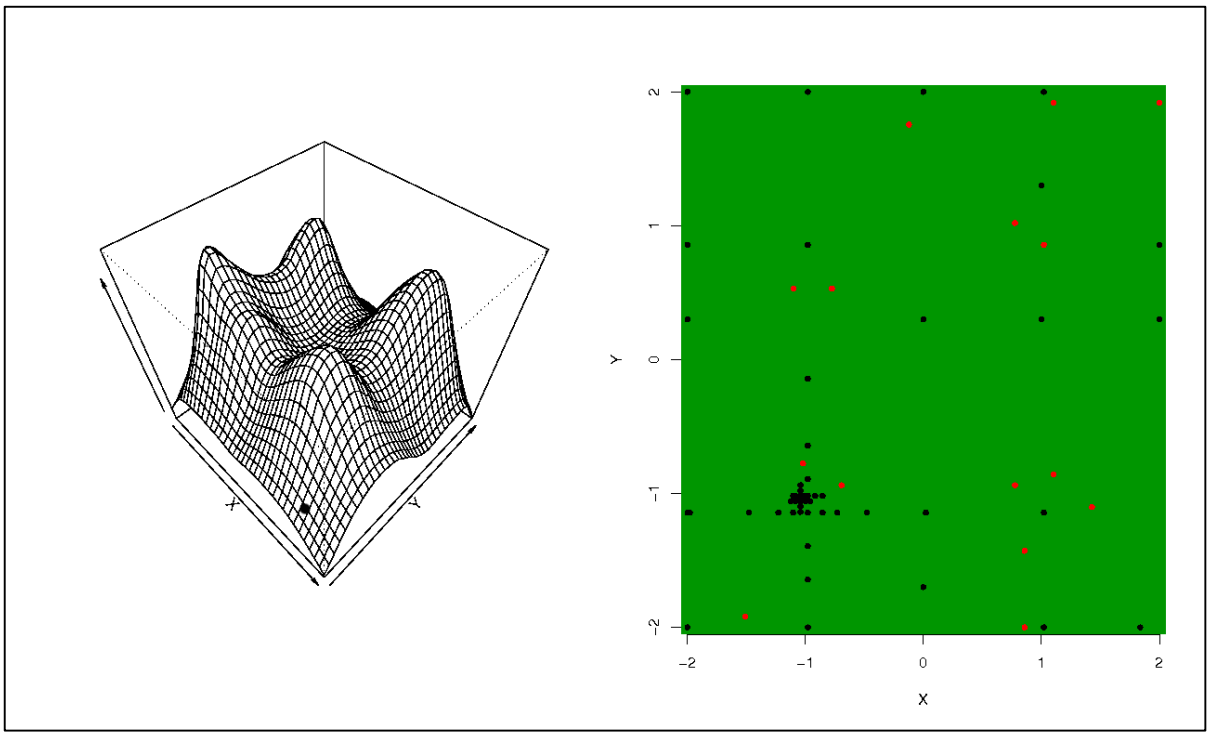

Figure 13: On the left, the resulting response surface of the molar function characterized by the generated APPSPACK iterates. On the right, the APPSPACK-tgp hybrid showing convergence to the true global maximum.

Now, using the same initial values as before, *APPSPACK*, without the tgp oracle, converges to a local maximum (x1,x2) = (1.125, 0.632) where f = 1.043, using 41 function evaluations. Examination of the expected improvement statistic shows that it is close to zero everywhere in the case of *APPSPACK-tgp*. However, in the case of *APPSPACK* alone, large variations in the expected improvement statistic were observed over the input space. These variations are illustrated by the white, yellow, and orange coloring in the right image in Figure 14. This suggests that the *tgp* oracle would have improved the optimization method by suggesting iterates from different parts of the input space, mainly the ones with remaining large uncertainty.

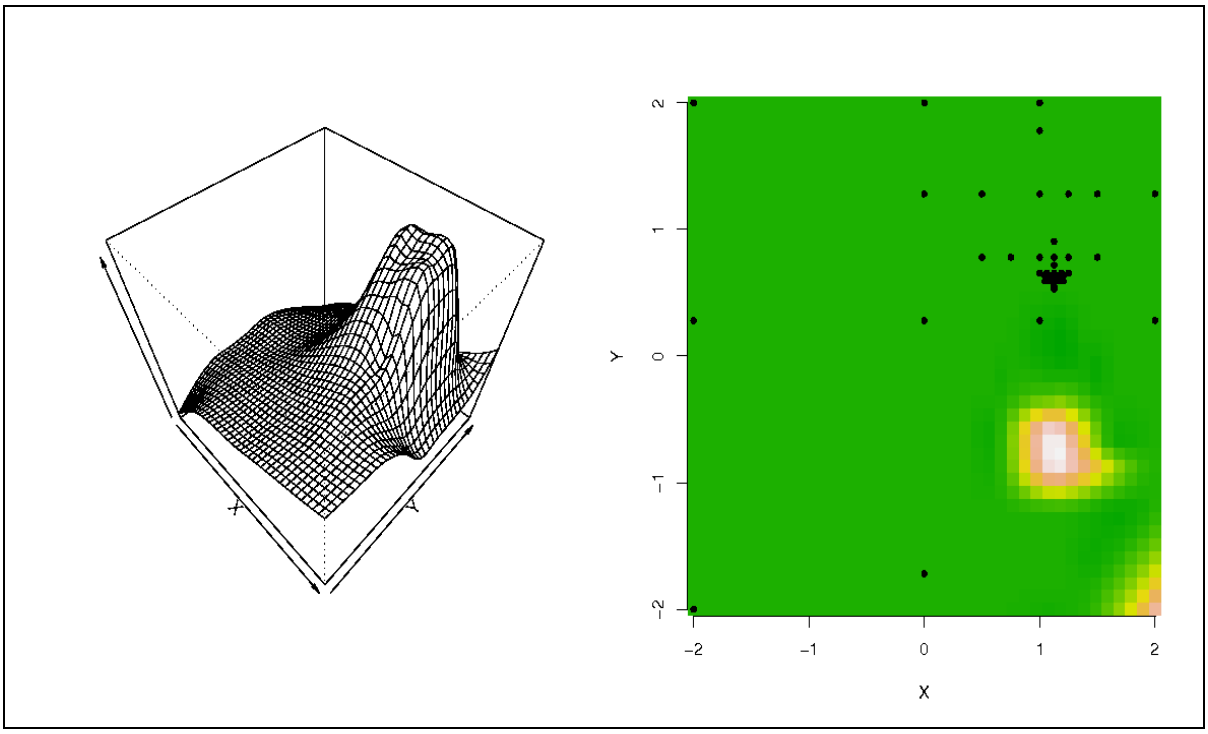

Figure 14: TGP fit to points from APPSPACK without the oracle. The mean predicted output surface (left) and the expected improvement over the input space (right) for the results of APPSPACK without the oracle. The black dots are the points evaluated during the pattern search

The mean predicted output surface (left image of Figure 14) also shows that how drastic is the cost of uncertainty in the unsampled input space – only a quarter of the second-order structure was captured. Using response surface approximations based on such a poor input space sampling would certainly spell disaster.

The maxima found with the oracle-enhanced version of *APPSPACK* were always higher than the solution from *APPSPACK* without an Oracle. However, there is currently no guarantee of locating the global solution if *APPSPACK* converges before the input space is fully explored. When the algorithm converges to the global maximum, as shown in Figure 13, we see that the expected improvement is everywhere zero. On the other hand, after convergence to a local maximum, there is still positive expected improvement over the input space, as illustrated by the white and orange-colored regions in Figure 14.

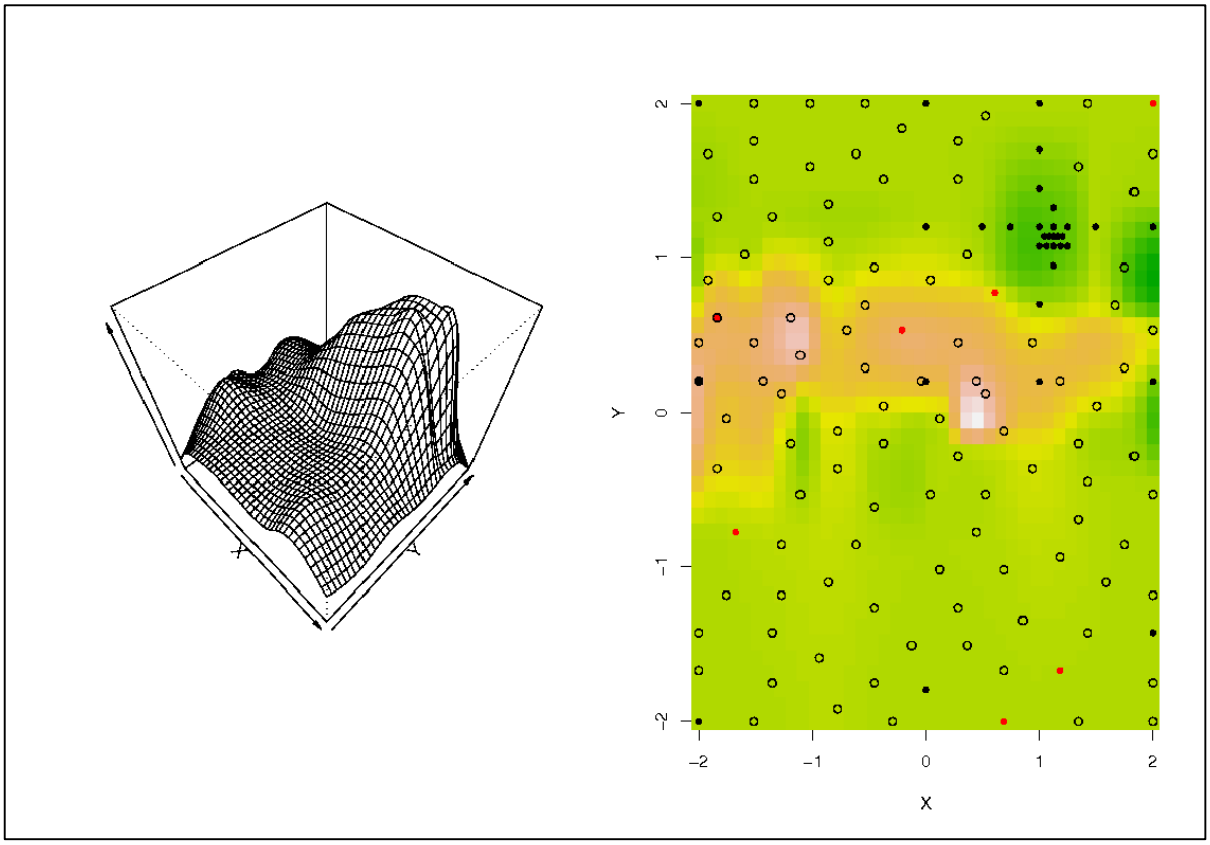

Figure 15: TGP fit after APPSPACK-*tgp* convergence to a local maximum

In Figure 15, we show an example where *APPSPACK-tgp* converges to a local maximum f = 1.061 at (x_1, x_2) =(1.125, 1.137) in 42 total function evaluations, where 8 iterates resulted from the *tgp* oracle. Figure 15 illustrates the mean predicted output surface (left image) and the expected improvement over the input space (right *image) for APPSPACK-tgp. On the right, the black dots are evaluated points chosen by APPSPACK*, the red dots are evaluated points chosen by the *tgp* oracle, and the open circles are candidate locations from a treed D-optimal design.

This observation warrants investigation into a convergence criterion that includes the expected improvement information provided by the oracle. Regardless, the *tgp* oracle allows us to make an intelligent assessment of the robustness of our solution and to obtain a probabilistic view of the global output surface.

7. Concluding Remarks

This project focused on research and algorithmic development in optimization under uncertainty (OUU) problems driven by earth penetrator (EP) designs. While taking into account uncertainty, we addressed three challenges in current simulation-based engineering design and analysis processes:

1. The first challenge required leveraging small local samples, already constructed by optimization algorithms, to build effective surrogate models. We used Gaussian Process Models to construct these surrogates. We developed two OUU algorithms using “local” GPs (OUU-LGP) and one OUU algorithm using “global” GPs (OUU-GGP) that appear competitive or better than current methods.
2. The second challenge was to develop a methodical design process based on multi-resolution, multi-fidelity models. We developed a Multi-Fidelity Bayesian Auto-regressive process (MF-BAP). This process allows us to exploit cheaper lower-fidelity information instead of depending exclusively on costly high fidelity information. Overall, the process yields overall computational savings whilst still allowing simulation-based design processes to be viable.
3. The third challenge involved the development of tools that are computational feasible and accessible. We have developed MATLAB® and DAKOTA implementations of our algorithms. Refinements of our implementation in DAKOTA are ongoing.

These algorithmic developments and implementations within Matlab/Dakota will help our S&T community (and beyond) substantially reduce the computational cost of their modeling, simulation, and design optimization efforts. Furthermore, the "R" in the "D" of our numerical methods has furthered our thinking about what is possible in different Sandia programs. For instance, the cost savings associated with the multi-fidelity GP models will allow us to fully explore the quantification of uncertainty in software packages (within Sandia's Verification & Validation Program and Sandia's S&T portfolio) that had thus far remained unanswered due to computational constraints; by extending the notion of expected improvement and efficient global optimization within our OUU algorithms, we can now begin to look at resource allocation, decision support, and margin quantification problems that were untractable until now.

References

Penetrator References

[Gold-1999] Goldstein, W. (1999). "Non-ideal projectile impact on targets," Int. J. of Impact Engineering, **22**:95-395.

[Davi-1979] Davie, N. T. (1979). "A Method for Describing Nearly Normal Penetration of an OGIVE Nosed Penetrator into Terrestial Targets," Technical Report SAND79-0721, Sandia National Laboratories, Albuquerque, NM 87185.

[jig-alw] JANE's INFORMATION GROUP. Air-launched weapons: Enhanced Paveway II and III (EGBU-24, EGBU-27, EGBU-28)

[jig-pb] JANE's INFORMATION GROUP. Air-launched weapons: GBU-27 and GBU-28 Penetration Bombs.

[MCB-2005] Marin, E. B., Chiesa, M. L., and Booker, P. M. (2005). "Parametric Studies of Penetration Events: a Design and Analysis of Experiments Approach," Technical Report SAND2005-0951, Sandia National Laboratories, Albuquerque, NM 87185.

[MSH-2006] Martinez-Canales, M. L., Swiler, L. P., and Heaphy, R. (2007). "Penetrator Reliability Investigation and Design Exploration: Penetrator Design Exploration Studies," Technical Report, Sandia National Laboratories.

Statistical Approaches

[BCG-2003] Banerjee, S., B. P. Carlin, and A. E. Gelfand (2003). Hierarchical Modeling and Analysis for Spatial Data, Chapman & Hall, Boca Raton, FL.

[FLS-2006] Fang, K.-T., R. Li, and A. Sudjianto (2006). Design and Modeling for Computer Experiments, Chapman & Hall/CRC, Boca Raton.

[HSS-1999] Hedayat, A. S., N. J. A. Sloane, and J. Stufken. (1999). Orthogonal Arrays: Theory and Applications, Springer Series in Statistics, Springer, New York

[JSW-1998] Jones, D.R., M. Schonlau, and W. J. Welch (1998). "Efficient globabl optimization of expensive black-box functions," Journal of Global Optimization, **13**:455—492.

[LT-2006] Lee, H. K. H. and M. Taddy (2006). "Simple multi-resolution test problems," Private Communication, March.

[Mate-1986] Matérn, B. (1986). Spatial Variation, Springer-Verlag, New York, second edition.

[MCB-1979] McKay, M. D., W. J. Conover, and R. J. Beckman (1979). "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code," Technometrics, **21**:239-245.

[Mont-2001] Montgomery, D. C. (2001). Design and Analysis of Experiments, 5th edition, Wiley.

[SCS-2000] Saltelli, A., K. Chan, E. M. Scott, editors (2000). Sensitivity Analysis, Wiley Series in Probability and Statistics, Wiley.

[SWM-1989] Sacks, J., W. J. Welch., T. J. Mitchell, and H. P. Wynn (1989). "Design and analysis of computer experiments," Statistical Science, **4**:409—435.

[SWN-2003] Santner, T. J., B. J. Williams, and W. I. Notz (2003). The Design and Analysis of Computer Experiments, Springer-Verlag, New York, NY.

Bayesian Approaches

[RSPA-2006] Rutherford, B. M., L.P. Swiler, T. L. Paez, and A. Urbina, (2006). "Response Surface (Meta-Model) Methods and Applications," Proceedings of the Society of Experimental Mechanics IMAC Conference, January 2006.

[Swil-2005] Swiler, L. P. (2005). "Bayesian Approaches to Engineering Design Problems," Technical Report SAND2005-3294, Sandia National Laboratories, Albuquerque, NM 87185.

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

Gaussian Processes

[Cres-1993] Cressie, N. A. C. (1993). Statistics for Spatial Data, revised edition,John Wiley and Sons.

[GL-2006] Gramacy, R. B. and H. K. H. Lee (2006). "Bayesian treed Gaussian process models'" Technical report, Dept. of Applied Math & Statistics, University of California, Santa Cruz.

[GM-web] 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.

[HKC-2004] 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**, pp. 448- 466, 2004.

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

[HANZ] Huang, D., T. T. Allen, W. I. Notz, and N. Zheng (to appear) "Global Optimization of Stochastic Black-Box Systems via Sequential Kriging Meta Models,", Journal of Global Optimization.

[HO-1994] Hjort, N. L. and H. Omre (1994). "Topics in spatial statistics," Scandinavian Journal of Statistics, **21**:289--357.

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

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

[Neal-1997] Neal, R. (1997). "Monte Carlo Implementation of Gaussian Process Models for Bayesian Regression and Classification." Technical Report 9702, Dept. of Statistics, University of Toronto.

[Neal-web] Neal, R. Flexible Bayesian Software documentation: http://www.cs.toronto.edu/~radford/fbm.software.html

[NB-web] Nabney, I. and C. Bishop. Netlab software. Documentation and software at: www.ncrg.aston.ac.uk/**netlab**/

[Book-2000] Booker, A. (2000). "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.

[Rasm-1996] Rasmussen, C. (1996). "Evaluation of Gaussian Processes and Other Methods for Nonlinear Regression." Ph.D. Thesis, University of Toronto.

[Swil-2006] Swiler, L.P. "Gaussian Processes in Response Surface Modeling." IMAC-XXIV Conference Proceedings, Society of Experimental Mechanics Meeting, Jan. 2006, St. Louis MO.

[Will-2002] Williams, C. (2002). "Gaussian Processes," A Chapter in The Handbook of Brain Theory and Neural Networks, M. Arbib, ed. Cambridge, MA: MIT Press.

Numerical Optimization

[ADLT-1998] Alexandrov, N., J. E. Dennis, Jr., R. M. Lewis, and V. Torczon (1998). "A Trust Region Framework for Managing the Use of Approximation Models in Optimization", Structural Optimization, **15**(1), pp. 6-12.

[DLT-2000] Dolan, E. D., R. M. Lewis, and V. Torczon (2000) "On the local convergence properties of parallel pattern search," Technical Report 2000-36, NASA Langley Research Center, Institute for Computer Applications in Science and Engineering, Hampton, VA.

[DS-1984] Dennis, Jr., J. E. and R. B. Schnabel (1983). Numerical Methods for Unconstrained Optimization and Nonlinear Equations, Prentice-Hall, Inc., Englewood Cliffs, NJ.

[DT-1991] Dennis, J. E. and V. Torczon (1991). "Direct search methods on parallel machines," SIAM J. Opt., **1**:448--474, 1991.

[EGC-2004] Eldred, M.S., A. A. Giunta, and S. S. Collis (2004). "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.

[ESGB-2006] Eldred, M.S., Swiler, L. P., Giunta, A.A., and S. L. Brown. (2006). "DAKOTA, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis. Version 4.0 Users Manual." Sandia Technical Report SAND2006-XXXX, April 2006.

[GK-2006] Gray, G. A. and T. G. Kolda (2006). "Algorithm 856: APPSPACK 4.0: Asynchronous parallel pattern search for derivative-free optimization," ACM TOMS, **32**(3):485—507.

[HKT-2001] Hough, P. D,, T.G. Kolda, and V. Torczon (2001). "Asynchrounous parallel pattern search for nonlinear optimization." SIAM J. Sci. Comput., **23**:134—156.

[Kold-2004] Kolda, T.G. (2004) "Revisiting asynchronous parallel pattern search," Technical Report SAND2004-8055, Sandia National Laboratories, Livermore, CA 94551, February.

[KLT-2003] Kolda, T. G., R. M. Lewis, and V. Torczon (2003). "Optimization by direct search: New perspectives on some classical and modern methods." SIAM Review **45**(3): 385-482.

[KT-2003] Kolda, T.G. and V. Torczon. (2003). Understanding asynchronous parallel pattern search. High Performance Algorithms and Software for Nonlinear Optimization, Boston, Kluwer Academic Publishers.

[KT-2004] Kolda, T. G. and V. Torczon (2004). "On the convergence of asynchronous parallel pattern search." SIAM Journal on Optimization **14**(4): 939-964.

[LT-1996] Lewis, R. M. and V. Torczon (1996). Rank ordering and positive basis in pattern search algorithms. Hampton, VA, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center.

[LTT-2000] Lewis, R.M., V. Torczon, and M. W. Trosset (2000). "Direct search methods: Then and now." Journal of Computational and Applied Mathematics **124**(1-2): 191-207.

[Mat-web] The MathWorks, Inc., fmincon documentation: http://www.mathworks.com/access/helpdesk_r13/help/toolbox/optim/fmincon.html#186882

[Nash-2000] Nash, S. G.,(2000). "A Multigrid Approach for Discretized Optimizaton Problems", Journal of Optimization Methods and Software, **14**, pp. 99-116.

[Sief-2000] Siefert, C. M.,(2000) "Model-Assisted Pattern Search", Honors Thesis, Department of Computer Science, College of William and Mary, Williamsburg, VA.

[Torc-1992] Torczon, V. (1992). Direct search methods for unconstrained optimization on either sequential or parallel machines. Department of Computational & Applied Mathematics. Houston, Rice University. **Ph.D.**

[Torc-1997] Torczon, V. (1997). "On the convergence of pattern search algorithms." SIAM J. Opt. **7**: 1-25.

[Wrig-1996] Wright, M. H. (1996). "Direct search methods: Once scorned, now respectable." In D. F. Griffiths and G. A. Watson, editors, "Numerical Analysis 1995 (Proceedings of the 1995 Dundee Biennial Conference in Numerical Analysis)", volume 344 of Pitman Research Notes in Mathematics, pages 191—208, CRC Press.

Parallel computing

[GL-1996] Gropp, W. D. and Lusk, E. (1996). "User's guide for *mpich*, a portable implementation of MPI," Technical Report ANL-96/6, Mathematics and Computer Science Division, Argonne National Laboratory, 1996.

[GLDS-1996] Gropp, W. D., Lusk, E., Doss, N., and Skjellum, A. (1996). "A high-performance, portable implementation of the MPI message passing interface standard," Parallel Comp., **22**:789--828.

News

[aip-152-2005] American Institute of Physics, FYI Number 152: October 26, 2005. http://www.aip.org/fyi/2005/152.html

[twt-10272005] The Washington Times, "Plans to Build 'Bunker-Buster' Warhead Dropped," October 27, 2005.

[jdw-03262003] JANE'S DEFENCE WEEKLY - MARCH 26, 2003 , "First Use of New Bunker Buster," **Andrew Koch** *Washington Bureau Chief. Additional Reporting* **Michael Sirak** *US Staff Reporter.*

Appendix A: Gaussian Process Implementation in Matlab

```
function [Yest] = libgpfull(Yobs, X, Xnew)

% libpgfull takes as input a set of observed data Yobs,
% a set of inputs corresponding to these outputs X,
% and a set of new inputs for which one wants a prediction, Xnew.
% Yest are the estimated values corresponding to Xnew, and sigest
% are the std. deviations of those estimates

[num_obs,num_vars]=size(X);

Y = Yobs-mean(Yobs);

for i = 1:num_vars
   Xnorm(:,i) = (X(:,i) - mean(X(:,i)))./std(X(:,i));
   Xprednorm(:,i) = (Xnew(:,i) - mean(X(:,i)))./std(X(:,i));
end

gpVar = 0.01;

cvparmsInit = 1*ones((num_vars+1),1);
lb = 0.01*ones((num_vars+1),1);
ub = 5*ones((num_vars+1),1);
cvparms = cvparmsInit;
thisLik = loglik(cvparms);

options = optimset('fmincon');
optnew = optimset(options, 'MaxFunEvals',100)
cvparms = fmincon(@loglik,cvparmsInit,[],[],[],[],lb,ub,[],optnew)

% Get inverse covariance matrix

E3=covvector(cvparms,Xnorm,Xprednorm,gpVar);
E5=E3';
E=covmatrix(cvparms,Xnorm,gpVar)\Y;
E2=E3*E;

Yest = E2 + mean(Yobs);

E4=covmatrix(cvparms,Xnorm,gpVar)\E5;
GPPredVar = E3*E4;
CovNew = covvector(cvparms,Xprednorm,Xprednorm,gpVar);
GPPredVar = CovNew-GPPredVar;
PredVar = diag(GPPredVar);
BoundLow=Yest-2*sqrt(PredVar);
BoundHigh = Yest+2*sqrt(PredVar);
sigest=sqrt(PredVar);

    function L = loglik(cvparms)
    CVM = covmatrix(cvparms,Xnorm,gpVar);
    L = log(det(CVM)) + Y'*(CVM\Y);
```

```
    % L = log(cvparms(a))+log(det(CVM)) + (1/cvparms(a))*Y'*(CVM\Y);
    end

end


function CVM = covmatrix(cvparms,Xnorm,gpVar)

[m,n] = size(Xnorm);
CVM = zeros(m);

i=1:1:m;
  for j = 1:m;
    ssqd = 0;
    for k=1:1:n ;
        ssqd=ssqd + cvparms(k)*(abs(Xnorm(i,k)-Xnorm(j,k)).^2.0);
    end;
    CVM(i,j)= CVM(i,j) + (cvparms(n+1)* exp(-.5*ssqd));
  end;

CVM = CVM + (gpVar*eye(m));

end

function CVV = covvector(cvparms,X,T,gpVar)

[m,n] = size(X);
[t,u] = size(T);


CVV = zeros(t,m);

i=1:1:t;
    for j=1:m;
        ssqd = 0;
        for k=1:1:n;
            ssqd = ssqd + cvparms(k)*(abs(T(i,k)-X(j,k)).^2.0);
        end;
        CVV(i,j)=CVV(i,j)+ (cvparms(n+1)*exp(-0.5*ssqd));
    end;

   for i=1:1:t;
    for j=1:m;
        k=1:1:n;
            if all (T(i,k)==X(j,k))
                CVV(i,j)=CVV(i,j)+gpVar;
            end;
    end;
   end;
end
```

Appendix B: Trust Region Optimization with Two-Fidelity Model Implementation in Matlab

```
% Input/Output Definitions

% x_min = vector; value of variables at solution (minimum)
% f_min = scalar; value of high-fidelity function at solution (minimum)
% stop_criteria = string; reason algorithm stopped
% hi_fi = pointer; high-fidelity function
% lo_fi = pointer; low-fidelity function
% gp_f = pointer; Gaussian process
% x_init = vector; starting values of variables
% lower_bounds = vector; lower bounds on variables
% upper_bounds = vector; upper bounds on variables
% num_samples = scalar; number of points to sample for GP construction
% min_TR_size = scalar; minimum size of trust region allowed
% min_f_change = scalar; minimum change in high-fidelity function from
%                one iteration to the next allowed
% max_iters = scalar; maximum number of iterations allowed

% Note1: Currently we assume that if one variable has a finite lower/upper
% bound, then all variables do.  Variables do not have to have both lower
% and upper bounds, nor do they have to have any bounds at all.  All
bounds
% must be finite - we don't check for infinite bounds.  All bounds must be
% consistent - we don't check that lower bounds are less than upper
bounds.

% Note2: The final three parameters are used to determine when the
%        algorithm should quit.  There is no real convergence check
%        (i.e., no check of gradient norm).

function [x_min, f_min, stop_criteria] = trmfo_gp(x_init, lower_bounds,
upper_bounds, num_samples, min_TR_size, min_f_change, max_iters)

% The first thing we do is put everything is in column format, if it isn't
% already!

   if (size(x_init,1) == 1)
      x_init = x_init';
      x_current = x_init
      f_current = inf
   else
      x_current = x_init;
      f_current = inf
   end

   if (size(lower_bounds,1) == 1)
      lower_bounds = lower_bounds';
   end
```

```
   if (size(upper_bounds,1) == 1)
      upper_bounds = upper_bounds';
   end

% Set trust region sizes of interest.  The initial trust region size
% is 1/2 the maximum difference between upper and lower bounds.  If only
% one-sided bounds are provided, the initial trust region size is the
% maximum absolute value of the bounds.  If there are no bounds, the
% initial trust region size is set to the norm of x_init.  The minimum
% trust region size is 1/2 the minimum difference between upper and
% lower bounds times the tolerance provided by the user.  If only
% one-sided bounds are provided, the minimum trust region size is the
% minimum absolute value of the bounds times the tolerance provided by
% the user.  If there are no bounds, the minimum trust region size is
% the norm of x_init times the tolerance provided by the user.

   if ((size(lower_bounds,1) ~= 0) && (size(upper_bounds,1) ~= 0))
      TR_size = 0.5*max(upper_bounds - lower_bounds);
      rel_min_TR_size = 0.5*min_TR_size*min(upper_bounds - lower_bounds);
   elseif ((size(lower_bounds,1) ~= 0) && (size(upper_bounds,1) == 0))
      TR_size = max(abs(lower_bounds));
      rel_min_TR_size = min_TR_size*min(abs(lower_bounds));
   elseif ((size(lower_bounds,1) == 0) && (size(upper_bounds,1) ~= 0))
      TR_size = max(abs(upper_bounds));
      rel_min_TR_size = min_TR_size*min(abs(upper_bounds));
   else
      TR_size = norm(x_init);
      rel_min_TR_size = min_TR_size*norm(x_init);
   end

% Set the random number to its initial state.  This should give us
% reproducibility for debugging purposes.  We can remove it later if we
% want.

   rand('state',0);

% Iterate on x until converged.

   for i = 1:max_iters

% BEGIN SAMPLING 1

% First sample is the current iterate, i.e., the center point of the
% trust region.

%      sample(:,1) = x_current;

% Set the bounds for sampling to the more restrictive of the
% user-supplied bounds and the trust region.  Then sample within those
% computed bounds.

%      sample_lbounds = max(sample(:,1)-TR_size, lower_bounds);
%      sample_ubounds = min(sample(:,1)+TR_size, upper_bounds);
```

```
%      for j = 1:num_samples
%         for k = 1:size(sample,1)
%            sample(k,j+1)= sample_lbounds(k,1) +
rand*(sample_ubounds(k,1) - sample_lbounds(k,1));
%         end
%      end

% For each sample point, compute both high- and low-fidelity function
% values and their differences.

%      for j = 1:num_samples+1
%         hi_response(j,1) = hi_fi(sample(:,j));
%         lo_response(j,1) = lo_fi(sample(:,j));
%         diff_response(j,1) = hi_response(j) - lo_response(j);
%      end

%      f_current = hi_response(1)

% END SAMPLING 1

      f_previous = f_current

% Iterate on step until acceptable one found.

      for j = 1:max_iters

% BEGIN SAMPLING 2

% First sample is the current iterate, i.e., the center point of the
% trust region.

         sample(:,1) = x_current;

% Set the bounds for sampling to the more restrictive of the
% user-supplied bounds and the trust region.  Then sample within those
% computed bounds.

         sample_lbounds = max(sample(:,1)-TR_size, lower_bounds);
         sample_ubounds = min(sample(:,1)+TR_size, upper_bounds);

         for j = 1:num_samples
            for k = 1:size(sample,1)
               sample(k,j+1)= sample_lbounds(k,1) +
rand*(sample_ubounds(k,1) - sample_lbounds(k,1));
            end
         end

% For each sample point, compute both high- and low-fidelity function
% values and their differences.

         for j = 1:num_samples+1
            hi_response(j,1) = hi_fi(sample(:,j));
            lo_response(j,1) = lo_fi(sample(:,j));
            diff_response(j,1) = hi_response(j) - lo_response(j);
```

```
        end

        f_current = hi_response(1)

% END SAMPLING 2

% Calculate the optimal values of the covariance parameters governing the
% GP (only need to do this once, then use these parameters everytime you
% call a predicted value within the optimization.  Note that there are two
% values of optimal parameters:  one for the low fidelity GP and one for
% the delta term

       [parms1, InvMtrx1] = libgpfull(lo_response, sample');
       [parms2, InvMtrx2] = libgpfull(diff_response, sample');

% Optimize the GP surrogate (gp_f = gp_lo + gp_diff).  Include the
% upper and lower bounds and the trust-region constraint.

        options = optimset('fmincon');
        optnew = optimset(options, 'MaxFunEvals',100);
        %[x_trial, gp_trial] = fmincon(@(x)gp_f(x, sample, lo_response,
diff_response), x_current, [], [], [], [], lower_bounds, upper_bounds);
        [x_trial, gp_trial] = fmincon(@(x)gp_f(x, sample, lo_response,
diff_response, parms1, InvMtrx1,parms2, InvMtrx2), x_current, [], [], [],
[], lower_bounds, upper_bounds, @(x)TR_constraint(x, x_current, TR_size),
optnew)

% Compute hi_fi function value at the point generated by optimizing
% the GP.

        f_trial = hi_fi(x_trial)

% Compute actual decrease and predicted decrease.  Do not compute or use
% the ratio, as it can lead to weird things like divide by 0.

     actual_decrease = f_current - f_trial
     predicted_decrease = f_current - gp_trial

% Determine whether to accept or reject trial step and adjust trust
% region size appropriately.  If trust region size gets too small or
% step is accepted, exit the loop.

        if (actual_decrease <= 0.25*predicted_decrease)   % step rejected
       TR_size = TR_size/2
       if (TR_size < rel_min_TR_size)
              break;
           end
        else                                              % step accepted
           x_current = x_trial
       f_current = f_trial
           if ((0.75*predicted_decrease <= actual_decrease) &&
(actual_decrease <= 1.25*predicted_decrease))
              TR_size = 2*TR_size
           end
```

```
            break;
         end
      end

% If trust region size is too small or if the change in the value
% of the high-fidelity function reached the stopping tolerance,
% then it is time to stop.

      if ((TR_size < rel_min_TR_size) || ((f_previous - f_current) <
min_f_change*abs(f_previous)))
         break;
      end
   end

   x_min = x_current;
   f_min = f_current;

   if (TR_size < rel_min_TR_size)
      stop_criteria = 'trust region size reached minimum tolerance';
   elseif ((f_previous - f_current) < min_f_change*abs(f_previous))
       stop_criteria = 'change in high-fidelity function reached minimum
tolerance';
   else
       stop_criteria = 'maximum number of iterations reached';
   end

end



% Compute gp_lo + gp_diff.

function f_est = gp_f(x, sample, lo_response, diff_response, parms1,
InvMtrx1,parms2,InvMtrx2)

% Need to transpose the samples and x for the GP code.

   f_est = gp_quick(lo_response, sample', x',parms1, InvMtrx1) +
gp_quick(diff_response, sample', x',parms2, InvMtrx2);

end



% Compute value of the trust-region constraint.

function [C, Ceq] = TR_constraint(x, x_current, TR_size)

   C = sum((x - x_current).^2) - TR_size;
   Ceq = [];

end
```

```
%Compute value of high fidelity response

function hi_response = hi_fi(x)

% Since x is a vector, refer to it with only one index.

    hi_response = (-exp(-1*(x(1)-1).^2) - exp(-.8*(x(1)+1).^2) +
0.05*sin(8*(x(1)+0.1)))*(-exp(-1*(x(2)-1).^2) - exp(-.8*(x(2)+1).^2) +
0.05*sin(8*(x(2)+0.1)));
    hi_response = -1*hi_response;

end



%Compute value of low fidelity response

function low_response = lo_fi(x)

% Since x is a vector, refer to it with only one index.

    low_response = (-exp(-1*(x(1)-1).^2) - exp(-.8*(x(1)+1).^2))*(-exp(-
1*(x(2)-1).^2) - exp(-.8*(x(2)+1).^2));
    low_response = -1*low_response;

end



%Calculate the hyperparameters governing the covariance matrix and the
% covariance matrix itself.

function [cvparms,InvCovMatrix] = libgpfull(Yobs, X)

%libgpfull takes as input a set of observed data Yobs, and
% a set of inputs corresponding to these outputs X.  From this,
% it calculates the optimal values of the hyperparameters governing
% the covariance, parms, as well as the covariance matrix of the
% observations, Cov.

    [num_obs,num_vars]=size(X);
    Y = Yobs-mean(Yobs);

    for i = 1:num_vars
          Xnorm(:,i) = (X(:,i) - mean(X(:,i)))./std(X(:,i));
    end

   %critical assumption - small jitter term
    gpVar = 0.0001;
```

```
    cvparmsInit = 1*ones((num_vars+1),1);
   %cvparmsInit(1)=1;
    lb = 0.001*ones((num_vars+1),1);
    ub = 5*ones((num_vars+1),1);
    cvparms = cvparmsInit;
    thisLik = loglik(cvparms);

    options = optimset('fmincon');
    optnew = optimset(options, 'MaxFunEvals',50);
    cvparms = fmincon(@loglik,cvparmsInit,[],[],[],[],lb,ub,[],optnew)
   %cvparms=[0.1,0.1,1]';

    %Get inverse covariance matrix times the target values
    InvCovMatrix=covmatrix(cvparms,Xnorm,gpVar)\Y;

    function L = loglik(cvparms)
         %[a,b]=size(cvparms);
        CVM = covmatrix(cvparms,Xnorm,gpVar);
        L = log(det(CVM)) + Y'*(CVM\Y);
        %L = log(cvparms(a))+log(det(CVM)) + (1/cvparms(a))*Y'*(CVM\Y);
    end

    function CVM = covmatrix(cvparms,Xnorm,gpVar)
        [m,n] = size(Xnorm);
        CVM = zeros(m);

          i=1:1:m;
          for j = 1:m;
             ssqd = 0;
            for k=1:1:n ;
                ssqd=ssqd + cvparms(k)*(abs(Xnorm(i,k)-Xnorm(j,k)).^2.0);
            end;
            CVM(i,j)= CVM(i,j) + (cvparms(n+1)* exp(-.5*ssqd));
            %CVM(i,j)= CVM(i,j) + exp(-.5*ssqd);
        end;
        CVM = CVM + (gpVar*eye(m));

    end
end

%Calculate the Gaussian process prediction

function [Yest] = gp_quick(Yobs,X,Xnew,cvparms,InvCovMatrix)
% gp_quick takes as input a set of observed data Yobs,
% a set of inputs corresponding to these outputs X,
% and a set of new inputs for which one wants a prediction, Xnew.
% It also takes as input the hyperparameters governing the covariance
% matrix, as well as the Inverse covariance matrix times the Y values.
% Yest are the estimated values corresponding to Xnew, and sigsq
% are the std. deviations of those estimates

    [num_obs,num_vars]=size(X);
    Y = Yobs-mean(Yobs);

    for i = 1:num_vars
```

```
        Xnorm(:,i) = (X(:,i) - mean(X(:,i)))./std(X(:,i));
        Xprednorm(:,i) = (Xnew(:,i) - mean(X(:,i)))./std(X(:,i));
    end

   %critical assumption - small jitter term
    gpVar = 0.0001;
    E3=covvector(cvparms,Xnorm,Xprednorm,gpVar);
    E5=E3';
    E2=E3*InvCovMatrix;
    Yest = E2 + mean(Yobs);

    %predict variance - we are not doing this yet
    %E4=covmatrix(cvparms,Xnorm,gpVar)\E5;
    %GPPredVar = E3*E4;
    %CovNew = covvector(cvparms,Xprednorm,Xprednorm,gpVar);
    %GPPredVar = CovNew-GPPredVar;
    %PredVar = diag(GPPredVar);
   %NormPredVar = PredVar*var(Yobs);
    %BoundLow=Yest-2*sqrt(PredVar);
    %BoundHigh = Yest+2*sqrt(PredVar);
    %sigsq=sqrt(PredVar);

    function CVV = covvector(cvparms,X,T,gpVar)

        [m,n] = size(X);
        [t,u] = size(T);
        CVV = zeros(t,m);
        i=1:1:t;
        for j=1:m;
            ssqd = 0;
            for k=1:1:n;
                ssqd = ssqd + cvparms(k)*(abs(T(i,k)-X(j,k)).^2.0);
            end;
            CVV(i,j)=CVV(i,j)+ (cvparms(n+1)*exp(-0.5*ssqd));
             %CVV(i,j)=CVV(i,j)+ exp(-0.5*ssqd);
        end;

        for i=1:1:t;
             for j=1:m;
             k=1:1:n;
                if all (T(i,k)==X(j,k))
                CVV(i,j)=CVV(i,j)+gpVar;
                end;
            end;
        end;
    end
end
```

Glossary

A *design (or control) variable* is a variable that can actually be controlled in a laboratory experiment or that can be changed by a manufacturing process. These are also called factors in the design of experiments literature.

An *uncertain (or noise) variable* is a variable that cannot actually be controlled in a laboratory experiment or that cannot be changed during a manufacturing process. These are also called covariates in the design of experiment literature. While not controllable in general, covariates can be measured.

A *sample* is a set of (uncertain) variables whose values have been determined from a probability distribution.

A *point* is a set of variables whose values have been fixed in some manner.

A *design point* is a set of design variables whose values have been fixed in some manner.

A *design matrix* (also, an *experiment matrix*) is a set of computer runs (or tests) consisting of a combination of variable values to determine an unknown effect or response.

A *design* is a model built from specified model parameters fixed in some combination.

A *parametric space exploration or parametric study* is an investigative process that generates simulation output that can then be analyzed from systematic changes to the simulation input.

A *Gaussian Process (GP)* is a stochastic process (random function) which has a Multivariate Normal distribution for each associated finite dimensional distribution function. The set of GPs indexed by the process parameters and their prior distributions represent our prior uncertainty regarding possible output surfaces.

Distribution

1	MS1318	Scott Mitchell	01411
1	MS1318	Laura P. Swiler	01411
1	MS1318	Steve W. Thomas	01411
1	MS0370	Timothy G. Trucano	01411
1	MS1318	Suzanne Rountree	01415
1	MS1318	Robert Heaphy	01415
1	MS1318	William E. Hart	01415
1	MS0139	Steve Lott	01812
1	MS	Scott Hutchinson	01437
1	MS0828	Martin Pilch	01533
1	MS0828	Anthony Giunta	01533
1	MS9151	James L. Handrock	08960
1	MS9159	Michael F. Hardwick	08964
1	MS9159	Heidi Ammerlahm	08962
1	MS9154	Ed Talbot	08222
1	MS	Neil F	
1	MS9159	Genetha A. Gray	08962
1	MS9159	Patricia D. Hough	08962
1	MS9159	Monica Martinez-Canales	08962
1	MS9403	Jill Hruby	08700
1	MS9405	Ken Wilson	08770
1	MS9042	Michael L. Chiesa	08774
1	MS9950	Randolph R. Settgast	08774
1	MS9042	Davina M. Kwon	08774
1	MS9153	Greg Thomas	08220
1	MS9034	Al McDonald	08221
1	MS9154	Paul Booker	08222
1	MS0123	LDRD Office	01011
1	MS9018	Central Technical File	08945-1
2	MS0899	Technical Library	09616
1	MS0612	Review & Approval Desk for DOE/OSTI	09612