TR – 2009 – 03

A Tutorial on Leveraging double-precision GPU Computing for MATLAB Applications

Makarand Datar (*datar@wisc.edu*)
Dan Negrut

June 8th 2009

1 Introduction

Advances in microprocessor performance in recent years have led to wider use of Computational Multi-Body Dynamic Simulations to reduce production costs, decrease product delivery time, and reproduce scenarios difficult or expensive to study experimentally. With the introduction of massively parallel Graphics Processing Units (GPUs), the general public has access to large amounts of computing power. GPUs offer a large number of computing cores and, like super computers, are highly parallel. Unlike super computers, however, GPU's are readily available at low costs.

The GPU is a device originally designed for graphics processing. Its applications generally relate to visualization in video games and other graphically intensive programs. These applications require a device that is capable of rendering, at high rates, hundreds of thousands of polygons in every frame. The computations performed are relatively simple and could easily be done by a CPU; it is the sheer number of calculations that makes CPU rendering impractical. The divide between CPUs and GPUs can be benchmarked by measuring their FLoating-point OPeration rate (FLOP rate). This benchmark also demonstrates how the unique architecture of the GPU gives it an enormous amount of computing power. A NVIDIA Tesla C1060 has a peak FLOP rate of 936 Gigaflops in single precision NVIDIA Corporation, 2008b. Comparatively the current fastest Intel core i7 CPU reaches 69.23 Gigaflops in double precision Laird, 2008.

Unlike a typical CPU, a GPU consists of several relatively slow multiprocessors, each of which is made up of multiple cores on which individual threads are processed. These cores are called Streaming Processors (SPs); SPs are generally organized into independent multiprocessor units called Streaming Multiprocessors (SMs), groups of SMs are contained within texture/processor clusters (TCPs). For example, an NVIDIA Tesla C1060 GPU contains 240 SP cores organized in 30 SMs grouped into 10 TCPs allowing it to execute 23040 threads simultaneously Lindholm et al., 2008. On the GPU each multiprocessor, along with having access to the GPUs global/main memory, has access to three other types of memory NVIDIA Corporation, 2008a. First is constant memory; this memory has extremely fast read times for cached data. It is ideal for values that do not change often. Next, texture memory specializes in fetching (reading) of two dimensional textures; fetches are cached, increasing bandwidth if data is read from the same area. Lastly, shared memory is a smaller 16KB piece of extremely fast memory that is unique for each SM and shared by its SPs, unlike texture and constant memory that are global. This is similar to an L2 cache on a CPU.

This technical report describes the use of GPU to achieve performance gain in the legacy MATLAB code.

2 Uncertainty Quantification framework

The GPU is used to improve performance of a code for establishing an analytically sound and computationally efficient framework for quantifying uncertainty in the dynamics of complex multi-body systems. The motivating question for this effort is as follows: how can one predict an average behavior and produce a confidence interval for the time evolution of a complex multi-body system that is subject to uncertain inputs? Herein, of interest is answering this question for ground vehicle systems whose dynamics are obtained as the solution of a set of differential-algebraic equations Hairer and Wanner, 1996. The differential equations follow from Newton's second law. The algebraic equations represent nonlinear kinematic equations that constrain the evolution of the bodies that make up the system Haug, 1989. The motivating question above is relevant for vehicle Condition-Based Maintenance (CBM), where the goal is to predict durability and fatigue of system components. For instance, the statistics of lower control arm loading in a High-Mobility Multi-Purpose Wheeled Vehicle (HMMWV) obtained through a multi-body dynamics simulation become the input to a durability analysis that can predict, in a stochastic framework, the condition of the part and recommend or postpone system maintenance. A stochastic characterization of system dynamics is also of interest in understanding limit behavior of the system. For instance, providing a confidence interval in real time for certain maneuvers is useful in assessing the control of a vehicle operating on icy road conditions. Vehicle dynamics analysis under uncertain environment conditions, e.g. road profile (elevation, roughness, friction coefficient) and aerodynamic loading, requires approaches that draw on random functions. The methodology is substantially more involved than the one required for handling uncertainty that enters the problem through discrete design parameters associated with the model. For instance, uncertainty in suspension spring stiffness or damping rates can be handled through random variables. In this case, methods such as the polynomial chaos (PC), see, for instance, Xiu and Karniadakis, 2002 are suitable provided the number of random variables is small. This is not the case here, since a discretization of the road leads to a very large number of random variables (the road attributes at each road grid point). Moreover, the PC methodology requires direct access and modification of the computer program used to run the deterministic simulations to produce first and second order moment information. This represents a serious limitation if relying on commercial off-the-shelf (COTS) software, which is most often the case in industry when running complex high-fidelity vehicle dynamics simulations. In conjunction with Monte Carlo analysis, the alternative considered herein relies on random functions to capture uncertainty in system parameters and/or input. Limiting the discussion to three-dimensional road profiles, the methodology samples a posterior distribution that is conditioned on available road profile measurements. Two paths can be followed to implement this methodology; the first draws on a parametric representation of the uncertainty, the second being nonparametric in nature. The latter approach is general yet expensive to implement. It can rely on smoothing techniques (nonparametric regression) that use kernel estimators such as Nadaraya-Watson or variants; see, for instance, Wasserman, 2006. The parametric approach is used in this thesis by considering Gaussian Random Functions as priors for the road profiles. Furthermore, the discussion will be limited to stationary processes although current research is also investigating the nonstationary case. The use of a parametric model raises two legitimate questions: why use a particular parametric model, and why is it fit to capture the statistics of the problem? Gaussian Random Functions (GRF) are completely defined by their correlation function, also known as a variogram Adler, 1990; Cramér and Leadbetter, 1967. Consequently, scrutinizing the choice of a parametric GRF model translates into scrutinizing the choice of correlation function. There are several families of correlation functions, the more common ones being exponential, Matérn, linear, spherical, and cubic (see, for instance Santner et al., 2003). In this context, and in order to demonstrate the proposed framework for uncertainty quantification in multi-body dynamics, a representative problem will be investigated in conjunction with the selection of a GRF-based prior. Specifically, an analysis will be carried out to assess the sensitivity of the response of a vehicle to uncertainty in system input, the uncertainty in this case being in the road profile. The outcome of interest will be the load history for the lower-control arm of an HMMWV, a key quantity in the CBM of the vehicle. The parametric priors considered are (i) a GRF with a squared exponential correlation function, (ii) the Ornstein-Uhlenbeck process and (iii) the Matérn correlation function. Pronounced sensitivity of the statistics of the loads acting on the lower control arm with respect to the choice of parametric model would suggest that serious consideration needs to be given to the nonparametric choice, where the empirical step of variogram selection is avoided at the price of a more complex method and increase in simulation time.

The discussion herein concerns handling uncertainty in spatial data. More specifically, it is assumed that limited information is used to generate road profiles that are subsequently used in the dynamic analysis of a ground vehicle. The handling of uncertainty in aerodynamic loads can be addressed similarly but is not of primary interest in this study and will be omitted.

The problem at hand is concerned with supervised learning, which is the problem of learning input-output mappings from empirical data (the training dataset) Rasmussen and Williams, 2006. Depending on the characteristics of the output, this problem is known as either regression, for continuous outputs, or classification, when outputs are discrete.

The input (or the independent variable) for the observed data will be denoted as x, which in general is a vector variable. The output is denoted as y and there can be multiple input and output variables. Hence we have a dataset of n observations, $\{(x_i, y_i) \mid i = 1,2,.....,n-1,n\}$. In the context of this thesis, the input data for road terrain is a collection of points along the lateral and longitudinal direction of the road and the output data is the elevation of the road at these locations.

With the aid of this input data, the goal is to predict the elevations of the road at a finer grid than the one available with observed data. This new set of inputs where predictions need to be made is denoted by x_*. Thus, from a finite set of observed data, predictions are made over any number of new input locations. The methodology used to achieve this relies on the idea of Bayesian inference and Gaussian processes. Gaussian process is a generalization of the Gaussian probability distribution. Whereas a probability distribution describes random variables which are scalars or vectors (for multivariate distributions), a Gaussian process describes distribution of functions instead of variables. Bayesian inference is used to give a prior probability to every possible function, where higher probabilities are given to functions that are more likely. This choice about likeliness is made after studying the observed data and locations where predictions need to be made. The Bayesian inference is illustrated graphically using a simple regression and classification example. Consider a simple 1-d regression problem, mapping from an input x to an output $f(x)$.

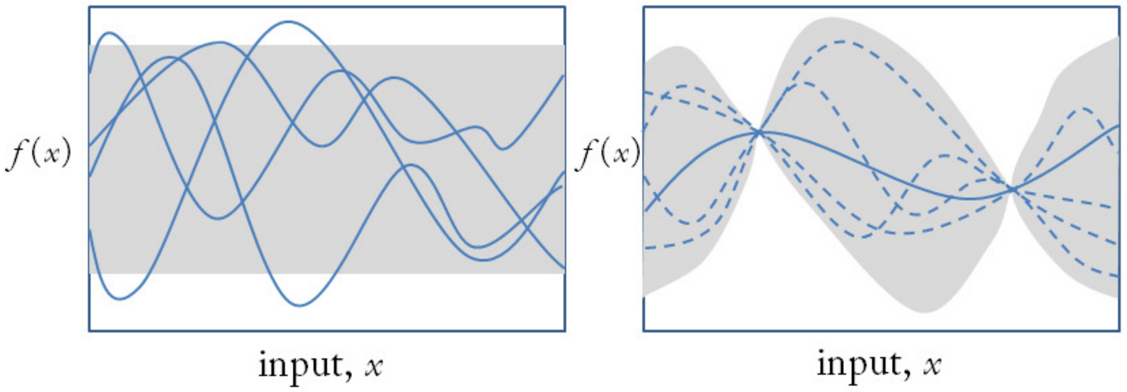

Figure 1: Left: four samples drawn from the prior distribution. Right: situation after two data points have been observed. The mean prediction is shown as the solid line and four samples from the posterior are shown as dashed lines. In both plots the shaded region denotes twice the standard deviation at each input value x Rasmussen and Williams, 2006.

Figure 1 shows number of sample functions drawn at random from a prior distribution based on some Gaussian distribution. This prior is taken to represent prior beliefs over the kinds of functions that are expected before seeing any observed data. The average value of all the infinitely many sample functions at each x is assumed to be zero. At any value of x the variability of the sample functions can be gauged by computing the variance at that point. The shaded region denotes twice the pointwise standard deviation.

With this background, given a dataset $\{(x_1, y_1), (x_2, y_2)\}$ consisting of two observations, only those functions that exactly pass through these points are considered. This situation is illustrated in Figure 1 (right). The dashed lines show sample functions which are consistent with the dataset, and the solid line depicts the mean value of such functions. The uncertainty is reduced closer to the data points and is zero at the data points. The combination of the prior and the data leads to the posterior distribution over functions. If more data points were added, the overall uncertainty at input locations is reduced.

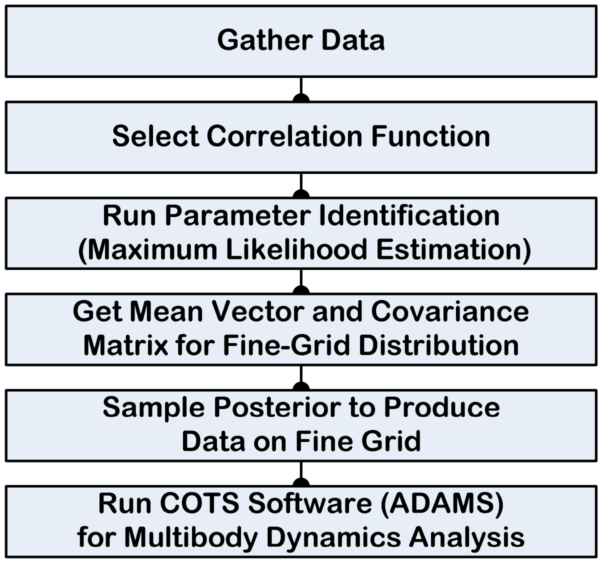
Gather Data
Select Correlation Function
Run Parameter Identification (Maximum Likelihood Estimation)
Get Mean Vector and Covariance Matrix for Fine-Grid Distribution
Sample Posterior to Produce Data on Fine Grid
Run COTS Software (ADAMS) for Multibody Dynamics Analysis

Figure 2: Proposed Uncertainty Quantification Framework

The uncertainty quantification framework proposed is described in Figure 2. An assumption is made that learning data has been made available as the result of field measurements.

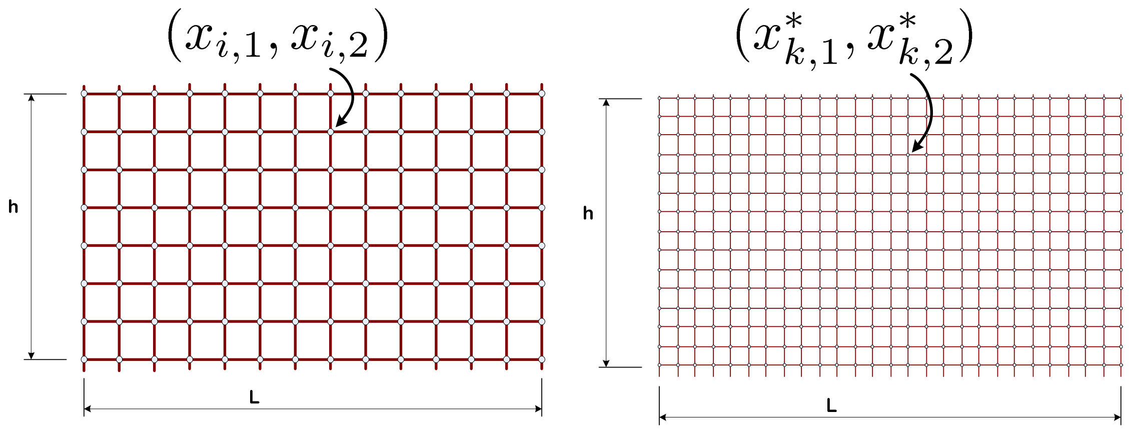

Figure 3: Coarse grid for learning, and fine grid employed in sampling for Monte Carlo analysis. Here $d = 2$.

Referring to Figure 3, the measured data is available on a “coarse” measurement grid. For dynamic analysis, road information is ideally available everywhere on the road as continuous data. Since, this is not possible however, the data is only provided on a fine grid (right image in Figure 3). If working with a parametric model, a correlation function is selected and a learning stage follows. Its outcome, a set of hyper-parameters associated with the correlation function, is instrumental in generating the mean and covariance matrix to be used in generating sample road surfaces on the user specified fine grid.

The learning data available is the road elevation as a discrete function of x-y co-ordinates. An example of such a road can be seen in Figure 3. Note that this represents a two-dimensional problem ($d = 2$).

The use of Gaussian Random Functions (GRF) or processes is a very versatile approach for the simulation of infinite dimensional uncertainty. In general, a spatially distributed random variable $\mathbf{y}(\mathbf{x})$, $\mathbf{x} \in \Re^d$, is a GRF with mean function $m(\mathbf{x};\theta_1)$ and correlation function $k(\mathbf{x},\mathbf{x}';\theta_2)$ if, for any set of space points $\mathbf{X} = \left\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M\right\}$,

$$\mathbf{y}(\mathbf{X}) = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{pmatrix} \sim N\left(\mathbf{m}(\mathbf{X};\theta_1), \mathbf{K}(\mathbf{X},\mathbf{X};\theta_2)\right) \tag{1}$$

Here $\mathbf{m} \in \Re^M$, $\mathbf{K} \in \Re^{M \times M}$, and $N(\mathbf{m},\mathbf{K})$ is the M-variate normal distribution with mean $\mathbf{m}$ and covariance $\mathbf{K}$ given by

$$\mathbf{m}(\mathbf{X};\theta_1) = \begin{pmatrix} m(x_1,\theta_1) \\ m(x_2,\theta_1) \\ \vdots \\ m(x_M,\theta_1) \end{pmatrix} \tag{2}$$

$$\mathbf{K}(\mathbf{X},\mathbf{X}';\theta_2) = \begin{pmatrix} k(x_1,x_1';\theta_2) & \cdots & k(x_1,x_N';\theta_2) \\ k(x_2,x_1';\theta_2) & \cdots & k(x_2,x_N';\theta_2) \\ \vdots & \vdots & \vdots \\ k(x_M,x_1';\theta_2) & \cdots & k(x_M,x_N';\theta_2) \end{pmatrix} \tag{3}$$

where $\mathbf{X}' = \left\{\mathbf{x}_1', \mathbf{x}_2', \ldots, \mathbf{x}_N'\right\}$. The hyper-parameters θ_1 and θ_2 associated with the mean and covariance functions are obtained from a data set $\mathbf{y}(D)$ at nodes $D = \left\{d_1, \ldots, d_M\right\}$. The posterior distribution of the variable $\mathbf{y}(S)$ at node points $S = \left\{s_1, \ldots, s_N\right\}$, consistent with $\mathbf{y}(D)$, is $N(\mathbf{f}^*, \mathbf{K}^*)$ Rasmussen and Williams, 2006, where

$$\begin{aligned}\mathbf{f}^* &= \mathbf{K}(S,D;\theta_2)\mathbf{K}^{-1}(D,D;\theta_2)\big(\mathbf{y}(D) - \mathbf{m}(D;\theta_1)\big) + \mathbf{m}(S;\theta_1)\\ \mathbf{K}^* &= \mathbf{K}(S,S;\theta_2) - \mathbf{K}(S,D;\theta_2)\mathbf{K}^{-1}(D,D;\theta_2)\mathbf{K}(D,S;\theta_2)\end{aligned}$$

The key issues in sampling from this posterior are a) how to obtain the hyper-parameters from data, and b) how to sample from $N(\mathbf{f}^*, \mathbf{K}^*)$ especially in the case where M is very large. The classical way of sampling relies on a Cholesky factorization of $\mathbf{K}^*$, a costly $O(M^3)$ operation. The efficient sampling question is discussed in Anitescu et al., 2008. A brief description of the hyper-parameter calculation follows.

2.1 Parameter Estimation

The method used herein for the estimation of the hyper-parameters from data is maximum likelihood estimation (MLE) Rasmussen and Williams, 2006. The method relies on the maximization of the log-likelihood function. In the multivariate Gaussian with mean $\mathbf{m}(\theta) \in \Re^M$ and covariance matrix $\mathbf{K}(\theta) \in \Re^{M \times M}$ case, the log-likelihood function assumes the form

$$\log p(\mathbf{y} \mid \theta) = -\frac{1}{2}\mathbf{W}^T \mathbf{K}(\theta)^{-1}\mathbf{W} - \frac{1}{2}\log \mid \mathbf{K}(\theta) \mid - \frac{M}{2}\log 2\pi$$

Here, $\mathbf{W} = \mathbf{y} - \mathbf{m}(\theta)$ and $\mathbf{y}$ is the observed data. Note that the dependence on the hyper-parameters θ appears in terms of coordinates $\mathbf{x}$, and $\theta = \{\theta_1, \theta_2\}$. The gradients of the likelihood function can be computed analytically Rasmussen and Williams, 2006:

$$\begin{aligned}\frac{\partial}{\partial \theta_{1j}}\log p(\mathbf{y} \mid \theta) &= -\left(\frac{\partial}{\partial \theta_{1j}}\mathbf{m}(\theta)\right)^T \mathbf{K}(\theta)^{-1}\mathbf{W}\\ \frac{\partial \log p(\mathbf{y} \mid \theta)}{\partial \theta_{2j}} &= \frac{1}{2}\operatorname{tr}\left((\mathbf{K}(\theta)^{-1}\mathbf{W}(\mathbf{K}(\theta)^{-1}\mathbf{W})^T - \mathbf{K}(\theta)^{-1})\frac{\partial \mathbf{K}(\theta)}{\partial \theta_{2j}}\right)\end{aligned} \tag{4}$$

MATLAB's fsolve function, which implements a quasi-Newton approach for nonlinear equations, was used to solve the first order optimality conditions $\frac{\partial \log p(\mathbf{y} \mid \theta)}{\partial \theta_{1j}} = 0$ and

$\frac{\partial \log p(\mathbf{y} \mid \theta)}{\partial \theta_{2j}} = 0$ to determine the hyper-parameters θ_1 and θ_2. The entire approach hinges, at this point, upon the selection of the parametric mean and covariance. It is common to select a zero mean prior $\mathbf{m} \equiv 0$, in which case only the θ_{2j} hyper-parameters associated with the covariance matrix remain to be inferred through MLE.

2.2 Covariance Function Selection

The parametric covariance function adopted determines the expression of the matrix $\mathbf{K}(\theta)$ from the previous subsection, and it requires an understanding of the underlying statistics associated with the data. In what follows, the discussion focuses on three common choices of correlation function: squared exponential (SE), Ornstein-Uhlenbeck (OU) Uhlenbeck and Ornstein, 1930 and Matérn (MTR) Matérn, 1960.

The SE correlation function assumes the form

$$k(x, x'; \theta_2) = \exp\left(-\left[\frac{(x_1 - x_{1'})}{\theta_{21}} \right]^{2/\gamma} - \left[\frac{(x_2 - x_{2'})}{\theta_{22}} \right]^{2/\gamma} \right), \tag{5}$$

where $\gamma = 1$. The hyper-parameters θ_{21} and θ_{22} are called the characteristic lengths associated with the stochastic process. They control the degree of spatial correlation; large values of these coefficients lead to large correlation lengths, while small values reduce the spatial correlation leading in the limit to white noise, that is, completely uncorrelated data. The SE is the only continuously differentiable member of the family of exponential GRF. As such, it is not commonly used for capturing road profiles, which are typically not characterized by this level of smoothness. To this end, Stein Stein, 1999 recommends the Matérn family with the correlation function

$$k(r; \theta_2) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu} r}{l} \right)^{\nu} K_{\nu}\left(\frac{\sqrt{2\nu} r}{l} \right), \tag{6}$$

with positive parameters ν and l, where K_ν is the modified Bessel function. The degree of smoothness of the ensuing GRF can be controlled through the parameter ν: the corresponding GRF is p-times differentiable iff $\nu > p$. Note that selecting $\gamma = 2$ in Eq. (5) leads to the OU random process, which is a nonsmooth process although not as general as the Matérn family.
The three covariance models discussed so far: SE, OU, and Matérn are stationary. Referring to Eq. (1), this means that for any set of points $\mathbf{X} = \left\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M\right\}$, where M is arbitrary, and for any vector $\mathbf{h} \in \Re^{\mathbf{d}}$, $\mathbf{y}(\mathbf{X})$ and $\mathbf{y}(\mathbf{X}+\mathbf{h})$ always have the same mean and covariance matrix. In particular, when $M = 1$, this means that the GRF should have the same mean and variance everywhere. Clearly, the stationary assumption does not hold in many cases. For vehicle simulation, consider the case of a road with a pothole in it, which cannot be captured by stationary processes. A nonstationary neural network covariance function has been proposed by Neal , 1996:

$$k(\mathbf{x}, \mathbf{x}'; \theta_2) = \frac{2}{\pi} \sin^{-1}\left(\frac{2\tilde{\mathbf{x}}^T \Sigma \tilde{\mathbf{x}}'}{\sqrt{\left(1 + 2\tilde{\mathbf{x}}^T \Sigma \tilde{\mathbf{x}}\right)\left(1 + 2\tilde{\mathbf{x}}'^T \Sigma \tilde{\mathbf{x}}'\right)}} \right), \tag{7}$$

where $\tilde{\mathbf{x}} = \left(1, x_1, \ldots, x_d\right)^T$ is an augmented input vector; the symmetric positive definite matrix Σ contains the parameters associated with this GRF and are determined based on the MLE approach described in 2.1. In this context, Rasmussen and Williams , 2006 suggest $\Sigma = diag(\sigma_0, \sigma_1, \ldots, \sigma_d)$. Recall that for the road profile problem $d = 2$.
Using this parameter estimation approach, a mean and co-variance function for the Gaussian process is determined. This is then used to generate new roads which are statistically equivalent to the road used in the learning process.
Uncertainty quantification (UQ) is the quantitative characterization of uncertainty in applications. Three types of uncertainties can be identified. The first type is uncertainty due to variability of input and/or model parameters, where, the characterization of the variability is given (for example, with probably density functions). The second type is similar to the first type except that the corresponding variability characterization is not available, in which case work needs modifications to gain better knowledge. The third type, which is the most challenging, is uncertainty due to an unknown process or mechanism (Sometimes type 1 is referred to as aleatory; type 2 and 3 are referred to as epistemic uncertainties). Type 1 UQ is relatively

straightforward to perform. Techniques such as Monte Carlo are frequently used. To evaluate type 2 uncertainties, many methods such as fuzzy logic or evidence theory have been proposed. An objective of UQ is to work towards reducing type 2 and 3 uncertainties to type 1. It often happens in real life applications that all three types of uncertainties are present in the systems under study. [Tong, 2006, Wojtkiewicz et al., 2001]

It becomes apparent that UQ is essential to the whole model development process from conception to maturity, and that it should probably not be viewed as just a discipline to study the propagation of uncertainty from the inputs to the output of interest. Thus UQ in the generic sense encompasses (not an exhaustive list) uncertainty analysis, sensitivity analysis, design exploration, design optimization, and design validation and calibration.

The reasons responsible to introduce uncertainty in a model may be:

1. The model structure, i.e., how accurately does a mathematical model describe the true system for a real-life situation
2. The numerical approximation, i.e., how appropriately a numerical method is used in approximating the operation of the system
3. The initial / boundary conditions, i.e., how precise are the data / information for initial and / or boundary conditions
4. The data for input and/or model parameters

The previous sections outlined how the time evolution of a system under consideration (HMMWV) is studied using a simulation tool, ADAMS/Car or a combination of simulation tools, PSAT, FTire and ADAMS/Car. Although the simulation tool can be trusted for its high fidelity multibody simulation capabilities, there is an inherent uncertainty in the very model that is being simulated. The vehicle model being simulated is an assembly of a vehicle chassis, a powertrain system, a tire model, a road model and a driver model. There are numerous sources of uncertainty in the model input parameters. For example, the vehicle is simulated assuming some stiffness value for the springs in the suspension system (which is usually the stiffness of the spring when the vehicle is first assembled). After a few years of operation, the changes in material properties, length and loads, will most definitely change the stiffness value of the spring. Not only that, but different vehicles will be in different conditions. Thus, a single simulation cannot be trusted for its results. What we need is a study set of simulations that will put upper and lower bounds on the results with some confidence. A 95% confidence interval is used in this work.

What this means is that one can say with a 95% confidence that the simulation result is going to be within the specified upped and lower bound. The numerical experiments carried out illustrate how the proposed uncertainty quantification framework is used to predict an average behavior and produce a confidence interval in relation to the time evolution of a complex multi-body system. A high-fidelity nonlinear ground vehicle model is considered, and its time evolution is subject to data uncertainty stemming from measurements of the road profile. This setup was chosen due to its relevance in CBM, where the interest is the statistics of the loads acting on the vehicle for durability analysis purposes. A detailed description of the results of this methodology are given by Datar, 2008.

3 Computation bottlenecks and solution

The parameter estimation algorithm described in the previous section can lead to large simulation times. This particularly evident when the size of the training data set (M) is large (see Eq. (3)). This results in having to compute a covariance function $\mathbf{K}$ where $\mathbf{K} \in \Re^{M \times M}$ in Eq. (3). The time taken for this operation scales with $O(M^2)$ making simulation times particularly long for large training datasets.

The most time consuming part of the code stems from the iterative numerical approach used in finding the values of hyper-parameters that satisfy first order optimality conditions $\frac{\partial \log p(\mathbf{y} \mid \theta)}{\partial \theta_{1j}} = 0$ and $\frac{\partial \log p(\mathbf{y} \mid \theta)}{\partial \theta_{2j}} = 0$ from Eq. (4). MATLAB's fsolve function, which implements a quasi-Newton approach for nonlinear equations, was used for this solution. This requires computation of the right side of Eq. (4) at each iteration until it is driven to zero. This function evaluation uses the sensitivity matrix $\frac{\partial \mathbf{K}(\theta)}{\partial \theta_{2j}}$ where $\frac{\partial \mathbf{K}(\theta)}{\partial \theta_{2j}} \in \Re^{M \times M}$. This again is a $O(M^2)$ operation and poses the same difficulties as computation of the covariance function $\mathbf{K}$. This problem of long simulation times can be overcome by exploiting NVIDIA's GPU architecture to implement a parallel algorithm for computation of $\mathbf{K}$ and $\frac{\partial \mathbf{K}(\theta)}{\partial \theta_{2j}}$.

The nature of the algorithm used for the computation of $\mathbf{K}$ and $\frac{\partial \mathbf{K}(\theta)}{\partial \theta_{2j}}$ allows for an embarrassingly parallel implementation using the data decomposition pattern. The data decomposition pattern is particularly useful if (1) the most computationally intensive part of the problem is organized around the manipulation of a large data structure and (2) similar operations are being applied to different parts of the data structure, in such a way that the different parts can be operated on relatively independently Mattson et al., 2004. Both these conditions are in favor for the parameter estimation framework with log-likelihood function explained above. As explained above, the fsolve function requires evaluation of large matrices (covariance function and its sensitivity wrt hyper-parameters) at every iteration and hence becomes the most computationally intensive part of the algorithm. Also, every entry in $\mathbf{K}$ and $\frac{\partial \mathbf{K}(\theta)}{\partial \theta_{2j}}$ is computed using exact same function that only differs in the data it operates on. Further, the computation of every entry in this matrix is completely independent of rest of the entries.

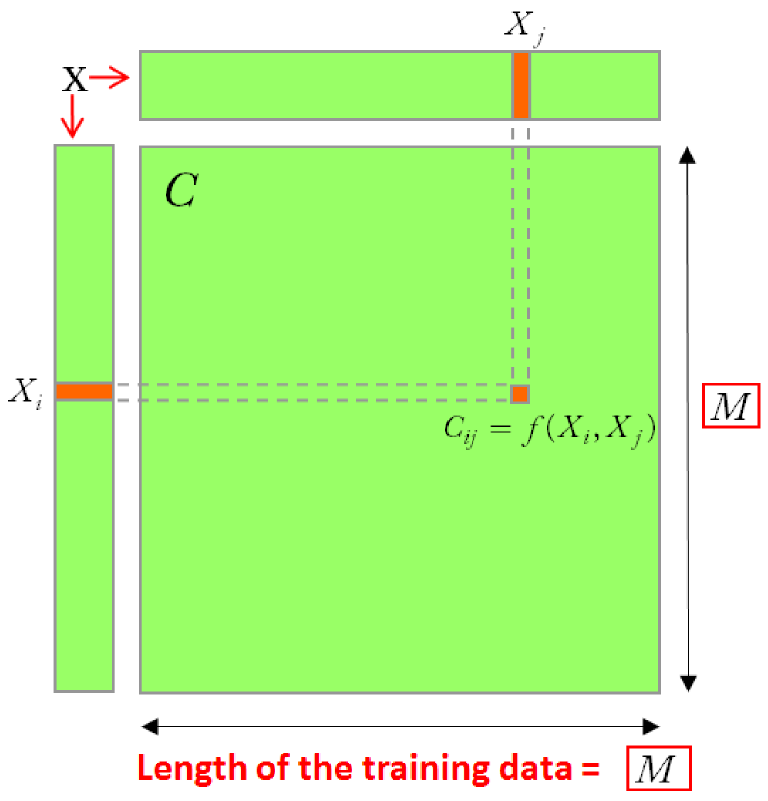

Figure 4: Parallel computation of $\mathbf{K}$ or $\dfrac{\partial \mathbf{K}(\theta)}{\partial \theta_{2j}}$

Referring to Figure 4, the output matrix is denoted by C, where $C \in \Re^{M \times M}$. A parallel implementation to generate this matrix is fairly straight forward. The output matrix C is subdivided into blocks that are updated simultaneously. An element C_{ij} in matrix C, needs access to only two data points, X_i and X_j from the training dataset.
This is implemented on the GPU by subdividing the output matrix is smaller chunks and then simultaneously updating each of them. Every subdivision is mapped on a block which copies the required data onto the shared memory and by appropriately mapping this data to individual threads, C_{ij}s are computed.

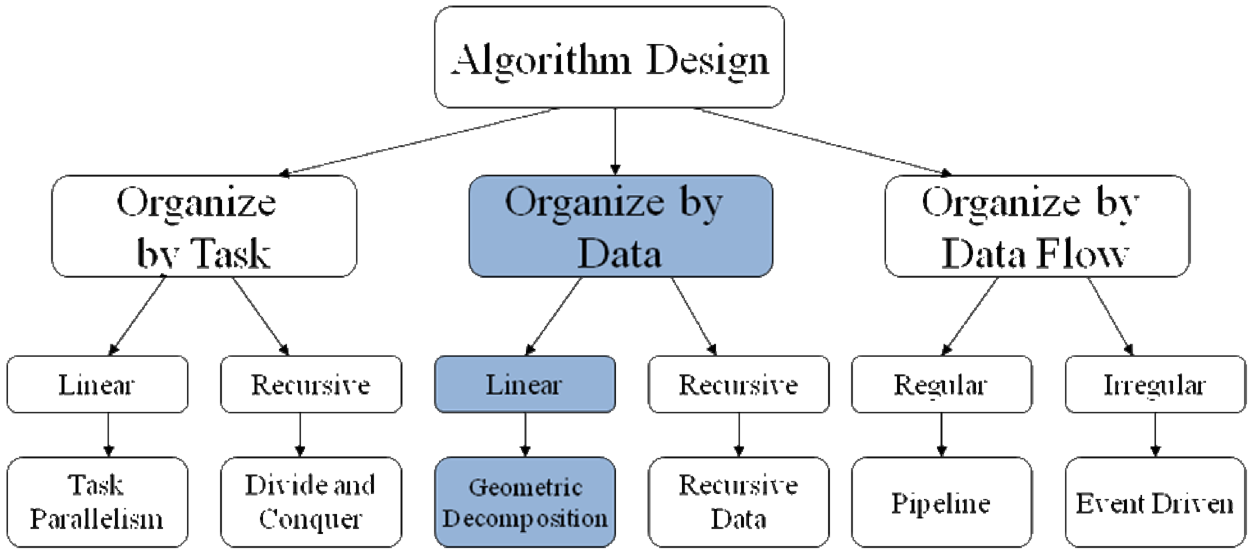

Figure 5: Choice of algorithm

Hence for this application, the parallel algorithm design is organized by data and the data is decomposed to be worked on concurrently. This is shown in the blue colored branch of Figure 5.

4 MATLAB – CUDA interface

Once the concurrency in the algorithm is identified (as explained in the previous section), there are two possible routes that could be followed for implementing the algorithm to run on the GPU. Firstly, the entire code can be rewritten in C to take advantage of the CUDA NVIDIA Corporation, 2008a API. Rewriting the MATLAB code in C can be cumbersome and at times can be extremely hard to do. MATLAB is higher level programming language with highly optimized - ready to use set of commonly encountered functions/techniques. For an example, it would take a lot of effort to implement an optimized fsolve routine, which uses a quasi-Newton approach for solution of nonlinear equations. This takes the focus away from a problem at hand of exploiting the concurrency in the algorithm.

The second approach is to retain the legacy MATLAB code and use MEX files to perform computations on the GPU using CUDA. MATLAB provides a script, called MEX, to compile a MEX file to a shared object or dll that can be loaded and executed inside a MATLAB session. This script is able to parse C, C++ and FORTRAN code. This script can call CUDA code to create greater optimization on GPUs.

MEX stands for MATLAB Executable. MEX-files are dynamically linked subroutines produced from C or Fortran source code that, when compiled, can be run from within MATLAB in the same way as MATLAB M-files or built-in functions. The external interface functions provide functionality to transfer data between MEX-files and MATLAB, and the ability to call MATLAB functions from C or Fortran code The MathWorks, 2009.

The main reasons to write a MEX-file are: 1) The ability to call large existing C or FORTRAN routines directly from MATLAB without having to rewrite them as M-files. 2) Speed; you can rewrite bottleneck computations (like for-loops) as a MEX-file for efficiency The MathWorks, 2009.

Hence, using this approach, the legacy MATLAB code is maintained and only the code involving the computation of $\mathbf{K}$ and $\frac{\partial \mathbf{K}(\theta)}{\partial \theta_{2j}}$ is branched off to be performed on the GPU.

For most general cases, the MEX file will contain CUDA code (kernel functions, configuration) and in order to be processed by "nvcc", the CUDA compiler, the file needs to have a ".cu" suffix. This suffix .cu could not be parsed from the native MATLAB MEX script. To solve this problem, NVIDIA developed new scripts that are able to parse these kind of files NVIDIA Corporation, 2007. In particular cases, it is still possible to write standard C code to process data on the GPU, especially when the operations performed on the data could be expressed in terms of calls to CUFFT or CUBLAS functions and use the regular MEX infrastructure.

The package downloadable from the NVIDIA web site contains a new script (called nvmex) and a configuration option file for the underlying compiler that simplify the deployment of CUDA based MEX files NVIDIA Corporation, 2007.

This plugin for MATLAB can be downloaded here

http://developer.nvidia.com/object/matlab_cuda.html

Follow the steps laid out in the readme file to setup the nvmex capability to work correctly. The current implementation works only for the single precision operations on the NVIDIA hardware, so the results are not in 64-bit precision that the native MATLAB implementation uses without the NVIDIA plug-in. For many applications, a single precision computation might not significantly affect the results and if the results with single precision are acceptable, this plugin can be used for GPU computations unaltered. However, in situations where double precision computations are a necessity for accuracy/stability/convergence of a code, modifications are required in the above mentioned plugin. Having a GPU that supports double precision computation is not enough. Even with a double precision GPU, all the computations are by default single precision. With the modifications in the plugin and a GPU that can perform double precision computations, the GPU computations can be forced to be double precision. Below is a list of GPUs that currently support a double precision computation.

GeForce GTX 295

GeForce GTX 285

GeForce GTX 280

GeForce GTX 260

Tesla S1070

Tesla C1060

Quadro Plex 2200 D2

Quadro FX 5800

Quadro FX 4800

To be able to perform double precision computations, the *nvmexopts.bat* file from the above mentioned plugin needs to be modified. Flags that force the double precision computation need to be permanently added in the COMPFLAGS line of the *nvmexopts.bat* file. A flag -arch sm_13 was added at the end of the COMPFLAGS line. The modified line looks like this:

```
set COMPFLAGS= -c -Xcompiler "/c /Zp8 /GR /W3 /EHs /D_CRT_SECURE_NO_DEPRECATE
/D_SCL_SECURE_NO_DEPRECATE /D_SECURE_SCL=0 /DMATLAB_MEX_FILE /nologo /MD" -arch sm_13
```

Once this change is made, the nvcc compiler knows that the GPU supports double precision computations. Other flags like -arch compute_13, --gpu-name sm_13, --gpu-name compute_13 have been reported to work.

With these modifications and use of MEX files, double precision computations on the GPU can be invoked from within the MATLAB code.

More information on use of MEX files can be found at the MATLAB online guide for using MEX files (The MathWorks, 2009).

5 Results and Conclusion

The use of MEX files to offload a part of MATLAB computation to GPU means that additional time is spent in copying data from MATLAB onto GPU and in reading the results back in MATLAB. However, if the size of the size of the output matrix $\mathbf{K}$ or $\frac{\partial \mathbf{K}(\theta)}{\partial \theta_{2j}}$ is large enough, this loss of time is overshadowed by the performance gain achieved in computing the matrices in parallel.

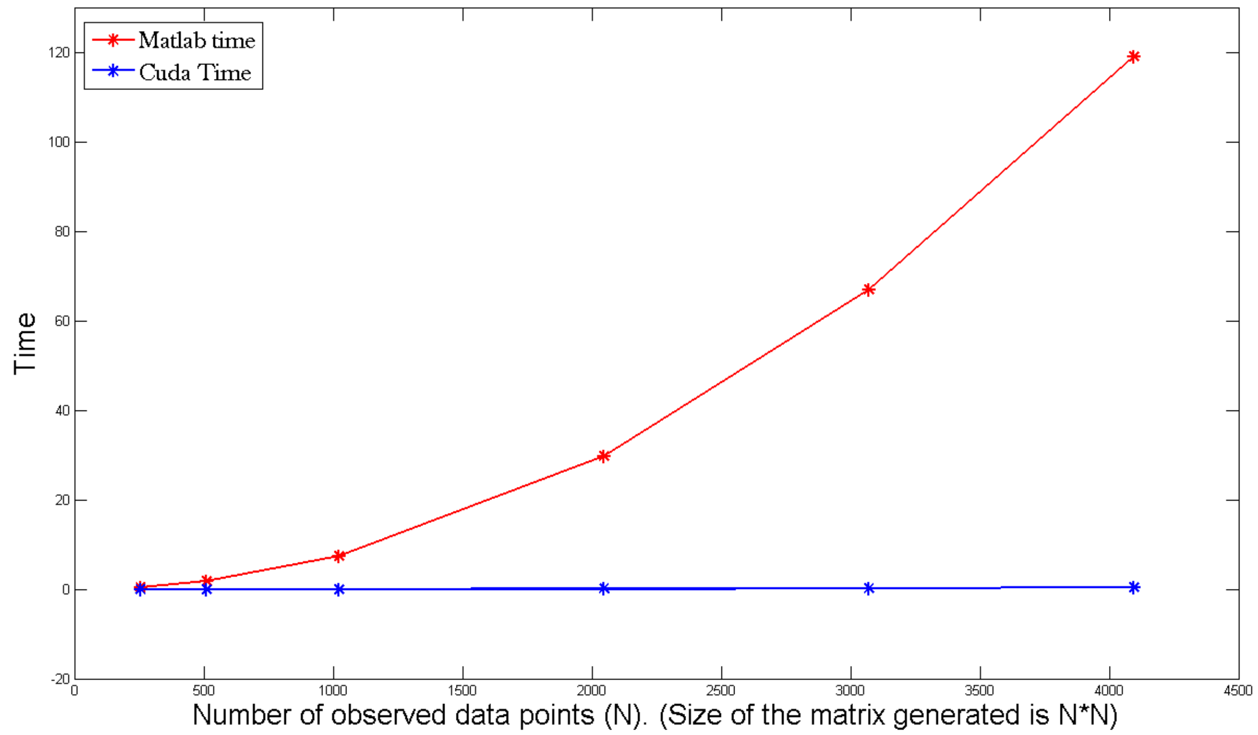

Figure 6: MATLAB and CUDA times for computing the gradient matrix $\frac{\partial \mathbf{K}(\theta)}{\partial \theta_{2j}}$

Figure 6 shows a plot of time taken to compute the gradient matrix $\frac{\partial \mathbf{K}(\theta)}{\partial \theta_{2j}}$ using native MATLAB code (red) and parallelized CUDA code (blue) time. The correlation function used here is Squared exponential (see Eq. (5)). Note that the CUDA execution is called from within the MATLAB code by use of MEX files. Figure 7 shows a plot of speedup in computation time by use of CUDA over the native MATLAB code. For observed data of size 256 (256 data points or 256 points on the road surface where the elevation data is known), the speedup is over 100. For a bigger size of data, for instance 4096 data points, the speedup is almost 250.

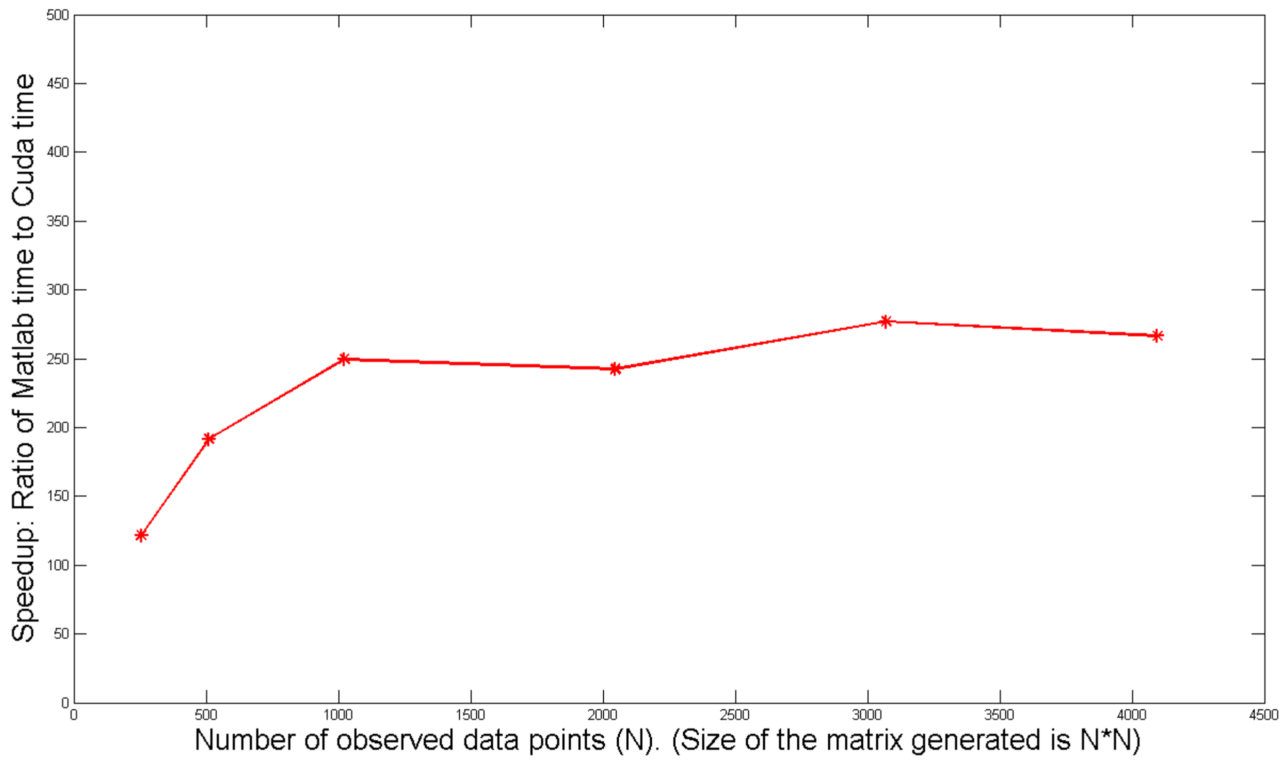

Figure 7: Speedup in computation time

Note that these speedups correspond to the time required for computing a single gradient matrix $\frac{\partial \mathbf{K}(\theta)}{\partial \theta_{2j}}$. Hence, if the size of the data to be computed is large enough and if the algorithm presents possibilities for a parallel implementation, use of GPU can lead to large computational gains.

References

Adler, R. J. (1990): *An Introduction to Continuity, Extrema, and Related Topics for General Gaussian Processes*, Institute of Mathematical Statistics.

Anitescu, M., Schmitt, K. and Negrut, D. (2008): Efficient sampling of dynamical systems with spatial uncertainty. *International Journal for Numerical Methods in Engineering, submitted*.

Cramér, H. and Leadbetter, M. R. (1967): *Stationary and related stochastic processes,* New York, Wiley

Datar, M. (2008): Uncertainty Quantification in Ground Vehicle Simulation, M.S. Thesis. *Mechanical Engineering.* Madison, University of Wisconsin-Madison.

Hairer, E. and Wanner, G. (1996): *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems,* Berlin, Springer.

Haug, E. J. (1989): *Computer-Aided Kinematics and Dynamics of Mechanical Systems. Volume I:Basic Methods,* Boston, MA, Allyn and Bacon.

Laird, J. (2008): Intel Corei7 benchmarks. Bath, Future Publishing Limited.

Lindholm, E., Nickolls, J., Oberman, S. and Montrym, J. (2008): NVIDIA Tesla: A Unified Graphics and Computing Architecture. *Micro, IEEE,* 28, 39-55.

Matérn, B. (1960): Spatial Variation (Lecture NotesStatist. 36). Springer, Berlin.

Mattson, T., Sanders, B. and Massingill, B. (2004): *Patterns for parallel programming*, Addison-Wesley.

Neal, R. (1996): *Bayesian Learning for Neural Networks*, Springer.

NVIDIA Corporation (2007): Accelerating MATLAB with CUDA™ Using MEX Files.

NVIDIA Corporation (2008a): NVIDIA CUDA: Compute Unified Device Architecture, Programming Guide. Santa Clara, NVIDIA Corporation.

NVIDIA Corporation (2008b): Tesla C1060 Computing Processor Board. *Board Specification.*

Rasmussen, C. E. and Williams, C. K. I. (2006): *Gaussian processes for machine learning*, Springer.

Santner, T. J., Williams, B. J. and Notz, W. (2003): *The Design and Analysis of Computer Experiments*, Springer.

Stein, M. (1999): *Interpolation of Spatial Data: Some Theory for Kriging*, Springer.

The MathWorks, I. (2009): MEX-files Guide. The MathWorks, Inc.

Tong, C. (2006): Refinement strategies for stratified sampling methods. *Reliability Engineering and System Safety,* 91, 1257-1265.

Uhlenbeck, G. and Ornstein, L. (1930): On the Theory of the Brownian Motion. *Physical Review,* 36, 823-841.

Wasserman, L. (2006): *All of Nonparametric Statistics*, Springer.

Wojtkiewicz, S., Eldred, M., Field, R., Urbina, A. and Red-Horse, J. (2001): Uncertainty quantification in large computational engineering models. *American Institute of Aeronautics and Astronautics,* 14.

Xiu, D. and Karniadakis, G. E. (2002): The Wiener-Askey Polynomial Chaos for Stochastic Differential Equations. *SIAM Journal on Scientific Computing*, 24, 619-644.