
# Strathprints Institutional Repository

Management Science
2008/02

# Emulation of Poincaré return maps with Gaussian Kriging models

Alireza Daneshkhah & Tim Bedford

Department of Management Science
Strathclyde Business School
University of Strathclyde
40 George Street
Glasgow
G1 1QE
Email: alireza.daneshkhah@strath.ac.uk
tim.bedford@strath.ac.uk

# Emulation of Poincaré return maps with Gaussian Kriging models

**Alireza Daneshkhah** **Tim Bedford**
Deptartment of Management Science
University of Strathclyde
Graham Hills Building, 40 George St.
Glasgow, G1 1QE, Scotland
alireza.daneshkhah@strath.ac.uk, tim.bedford@strath.ac.uk

February 6, 2008

**Abstract**

In this paper we investigate the use of Gaussian emulators to give an accurate and computationally fast method to approximate return maps, a tool used to study the dynamics of differential equations. One advantage of emulators over other approximation techniques is that they encode deterministic data exactly, so where values of the return map are known these are also outputs of the emulator output, another is that emulators allow us to simultaneously emulate a parameterized family of ODEs giving a tool to assess the behavior of perturbed systems. The methods introduced here are illustrated using two well-known dynamical systems: The Rössler equations, and the Billiard system. We show that the method can be used to look at return maps and discuss the further implications for full computation of differential equation outputs.

KEY WORDS: Billiard system; Chaotic and non-chaotic dynamic systems; Computer code output models; Emulators; Gaussian process; Poincaré return map; Rössler flow.

## 1 Introduction

The dynamical behaviour of many real world systems can be modeled by a set of ordinary differential equations. Numerical simulation of such systems is usually fairly straightforward, but always involves a tradeoff between accuracy and computation time. In many applications we need to carry not both accurate and fast simulations. Therefore it is of interest to find alternative ways to perform computations.

These mathematical models described the complex dynamical systems are typically implemented in computer codes. Often, the mathematical model is highly complex, and the resulting computer code is large and may be expensive in terms of the computer time required for a single run. Nevertheless, running the computer model will be much cheaper

than making direct observations of the process. A range of tools have been developed using Bayesian statistics to tackle many important problems faced by users and developers of complex process models.

These methods can be seen as developments of work on Design and Analysis of Computer Experiments (DACE) in the 1980s, which introduced the fundamental idea of building a statistical *emulator* of a simulator model (see, Sacks et al (1989a, b), Welch et al (1992) and Morris et al (1993)). The Bayesian analysis of Computer Code Outputs (BACCO) extends this approach both in terms of emulation of more complex behaviour and in terms of methodology to employ emulators to address a wide range of practical questions in the development and use of process models (see O'Hagan (2006) and the references therein for a comprehensive tutorial).

An emulator is not only statistical approximation to the model of interest (simulator), but it is built using statistical methods. Given a set of *training runs* of the model, considered as the model inputs, the model outputs can be evaluated/ovserved. We then treat these inputs and outputs to estimate the model. The form of emulator used in the BACCO approach is the Gaussian process, which is an analytically very tractable of stochastic process. A property fitted Gaussian process emulator will satisfy the following two fundamental criteria: 1) at a design training point, the emulator should reflect the fact that the true value of the model output is known, so it should return the corresponding model output with no uncertainty; 2) at the other points, the distribution for the model given a mean value that represents a plausible interpolation or extrapolation of the training data, and the probability distribution around this mean should be a realistic expression of uncertainty about how the model might interpolation/extrapolation. Haylock and O'Hagan (1996), Kennedy and O'Hagan (2001) and Oakley and O'Hagan (2002) use the Gaussian emulators to approximate the deterministic and computationally expensive computer codes output models.

Once the Gaussian process emulator has been built, various analyses can be made without more simulation runs. The simplest such analysis is to predict the outputs at inputs not previously run on the simulator. An application of these processes to select input and prediction in computer models is given in Welch et al (1992). Furthermore, using these emulators is already well established for uncertainty and sensitivity analyses (see Oakley and O'Hagan, 2002, 2004). Kennedy et al (2006) present a number of recent case studies in which an emulator of a computer code is created using a Gaussian process model. They used this emulator to illustrate the sensitivity and uncertainty analysis.

The Gaussian process emulators have recently been used in the context of dynamical systems by Conti and O'Hagan (2007). They developed the multi-output emulator which can form the basis for emulation of dynamic simulators. The dynamic simulator they considered models a system that is evolving over time and operate iteratively over fixed time steps. A single run of such a simulator generally consists of a simulation over many time steps, and it can be thought in terms of a simpler, single-step simulator being run iteratively many times. However, there are a number of technical difficulties that arise from the time parameter in the emulator. The key to the multi-output emulator's improved performance relative to the time-input emulator lies in its more flexible modelling of correlations over time, but this is also a source of extra computational load through the need for larger training samples. Thus, this method could be still slow for ordinary differential equations due to the emulation of a large number of single-step functions.

This paper explores a different approach which largely eliminates the need to include

the time parameter, namely the use of Poincaré return maps, and more generally the use of cross sections. Fast simulation of dynamical systems can then be achieved by the iteration of mappings between cross sections, together with conventional numerical solution methods to get to and from the cross sections at the start and finish of a simulation.

The paper illustrates the construction of this map for two well known examples. In the first example, we emulate the Poinacré return map for Rössler flow (a comprehensive review of construction of this map for this system can be found in Basu, 2007). We also estimate the Poinacré return map for the Lorenz attractor (introduced in 1963 by Lorenz) and calculated by the numerical methods in Henon (1982) and Tucker (2002). We now review some of the required background material for dynamical systems and emulators, in order to provide the context for our work.

## 1.1 Dynamical systems

This section recalls some well-known concepts from dynamical systems theory (see Katok and Hasselblatt (1996), Hirsch et al (2003), Strogatz (1994) and references therein). We start with an ordinary differential equation, whose solution gives a flow, that is, a mapping parameterized by time $t$. We suppose that the differential equation is defined on a space $X$ (a "nice" subset of $\mathbb{R}^n$), and that it has unique solutions given by a flow

$$\Phi_t : X \to X.$$

Furthermore, we suppose that there are parameters, $a$, in the differential equations that we want to study the effect of. So the flow is really a function of these parameters, $\Phi_{t,a}$. Typically we are only interested in parameters in some open neighbourhood of a particular value $a_0$. For most of the discussion we shall drop the parameter.

A point $x \in X$ is *(forward) recurrent* if for any open neighbourhood $U$ of $x$, there is a time $t > 0$ such that $\Phi_t(x) \in U$. Otherwise the point is *transient*.

In practical applications we see a mixture of recurrent and transient behaviour. Recurrent behaviour can be "nice", for example where you have a periodic orbit (a point that returns to itself after some time) or chaotic (where there is a forward invariant set $K$, $\Phi_t(K) \subset K$ in which any point in $K$ returns near any other point in $K$ infinitely often). Transient behaviour is also important in practical applications. However, either transient points go off to infinity or they converge to some recurrent set.

As the numerical solution of the flow takes some computational effort, we are interested in ways to reduce this, or to compactly represent the flow.

A set $Y \subset X$ is a *cross-section* to the flow if whenever $y \in Y$, $\Phi_t(y) \notin Y$ for all small $t > 0$. In other words, the flow moves points off of $Y$. A cross-section should have one dimension less than that of $X$, that is, it is of co-dimension 1. The notion of a cross section can be used for both transient and recurrent situations. Suppose that $Y_0$ and $Y_1$ are cross-sections to the flow. Then we can define an induced mapping $r : Y_0 \to Y_1 \cup \{\infty\}$ and a first-hitting time $h : Y_0 \to \mathbb{R} \cup \{\infty\}$ as follows by

$$h(y) = \begin{cases} \inf\{t : \Phi_t(y) \in Y_1\} & \text{if } \Phi_t(y) \in Y_1 \text{ for some } t > 0, \text{ or} \\ \infty & \text{otherwise} \end{cases}$$

and

$$r(y) = \begin{cases} \Phi_{h(y)}(y) & \text{if } h(y) < \infty, \text{or} \\ \infty & \text{otherwise.} \end{cases}$$

In computational studies one typically considers an initial starting point $y$ and takes $Y_0$ to be a "flat disc" around $y$ transverse (perpendicular) to the flow. We then flow from $y$ for a chosen time to reach a point $y_1$, and define $Y_1$ as a "flat disk" transverse to the flow around $y_1$.

For computational reasons it is good to choose $Y_0$ and $Y_1$ as easily described sets- for example by fixing one coordinate to get a co-dimension 1 subspace. Note that the first hitting time function $h$ will generally be non-linear. In principle we could make $h$ a simple function - a constant - by choosing to flow for a fixed time with every point of $Y_0$. However, this makes $Y_1$ a much more complicated space. Hence either $Y_1$ is easy to describe and $h$ is complicated, or the other way around.

The flow - at least from $Y_0$ and $Y_1$ - is entirely specified by the mappings $r$ and $h$. If we consider a parameterized family of flows, with a vector of parameters $a$, then the mappings $r$ and $h$ are also indexed by these parameters. There are some technical issues here - however, in general the same cross sections can still be used for nearby parameters, and so one can represent the effect of changing parameters purely in the return map and return time function.

When the flow is recurrent then we could take $Y_0 = Y_1$ and the mapping $h$ is called a Poincaré return map. When the flow is transient one might consider a whole sequence of subspaces $Y_0, Y_1, Y_2, \ldots$ which are visited in that order, with corresponding maps $h_i, r_i$.

From a computational point of view, the reason for doing this is that if we have a rapid way of computing the maps, then we could calculate $\Phi_t(x)$ for any $t$ and $x$ quite quickly. The flow is solved from its initial condition $x$ until it first reaches a cross section $Y_i$ say, at time $t_i$ and at point $y_i$.

If $t_i < t < t_i + h_i(y_i)$ then the flow is solved until you get to time $t$. However, if $t > t_i + h_i(y_i)$ then you can jump by the mapping to the next point

$$y_{i+1} = r_i(y_i) = \Phi_{t_i + h_i(y_i)}(x)$$

This process is iterated so that the point jumps from one cross-section to the next, until the cumulative time as measured from the return time function is almost equal to $t$. The flow is computed conventionally from the last cross-section point.

There is clearly a computational balance here. Computing lots of maps along the flow might be a lot of work, however, iterating maps is much faster than stepwise solution of differential equations. Because of the difficulty of computing, approximating and representing return maps, this computational method is not in use. However, our use of Gaussian process emulators makes such a computational strategy a realistic option.

It should be noticed that when we compute the flow in this way, we are effectively iterating the maps $h_i(h_{i-1}(h_{i-2} \ldots h_1(y_1) \ldots))$, which means we are coupling emulators in chains. This raises the possibility that chains of emulators could themselves be emulated in order to provide even faster computations.

## 1.2 Gaussian processes

Large computer codes to implement very complicated mathematical models are comprehensively used in all fields of science and technology to explain and understand complex systems.

This program is called a *simulator*. The simulator that we are dealing with in this paper is assumed to be deterministic in the sense that we will get the same output for running simulator for the same input many times. The size and complexity of a simulator can become a problem when it is necessary to make very many runs at different inputs. The most simulation methods, in particular, standard Monte Carlo based methods require thousands of simulator runs. This can become infeasible even for moderately large computer models which requires just a few seconds per run.

Following Sacks et al. (1989), a two-stage approach bases on *emulation* (see also Haylock and O'Hagan, 1996; Kennedy and O'Hagan, 2001; O'Hagan, 2006) of the simulator's output has been developed which is offering substantial efficiency gains over standard Monte carlo-based methods. The simulator can be represented in the form of a function $\mathbf{y} = \mathbf{f}(\mathbf{x})$, and the process of generating one set of outputs $\mathbf{y}$ for one particular input configuration $x$ is called a *run* of the simulator. A Bayesian formulation assumes a Gaussian process prior distribution for the function $\mathbf{f}(.)$, conditional on various hyperparameters. This prior distribution is updated using as data a preliminary training sample $\{y_1 = f(x_1), \ldots, y_n = f(x_n)\}$ of $n$ selected simulator runs. Formally, the posterior mean of $f(.)$ is regarded as the emulator. The derived posterior distribution is also a Gaussian process conditional on the hyperparameters. The conditioning upon the training set forces realisations from the emulator to interpolate the observed data points and induces posterior distributions for the hyperparameters.

The emulator is built in the first stage of the approach mentioned above, then problems such as uncertainty analysis, sensitivity analysis or calibration are tackled in the next stage using the emulator. It should be noticed that the emulator runs almost instantaneously, therefore the computational cost of the emulator based approach lies in producing the training data. The efficiency gains arise because it is usually possible to emulate the simulator output to a higher degree of precision using only a few hundred runs of the simulator.

We first review the emulation of the multi-output simulator introduced by Conti and O'Hagan (2007). We consider a deterministic simulator that return outputs $\mathbf{y} \in \mathbb{R}^q$ from inputs $\mathbf{x}$ lying in some input space $\mathcal{X} \subseteq \mathbb{R}^p$. The simulator is essentially a function $\mathbf{f} : \mathcal{X} \mapsto \mathbb{R}^q$; that is, $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_q((\mathbf{x}))^T$. Despite $\mathbf{y} = \mathbf{f}(\mathbf{x})$ being in principle known for any $\mathbf{x}$, in practice the complexity of the simulator requires to run the computer code to determine $\mathbf{y}$. From a Bayesian perspective, we thus regard $\mathbf{f}(.)$ as an unknown function, and following O'Hagan et al. (1999) model the uncertainty around this function can be represented by means of the $q$-dimensional Gaussian process as follows

$$\mathbf{f}(.)|B, \Sigma, \mathbf{r} \sim \mathcal{N}_q(\mathbf{m}(.), c(.,.)\Sigma) \tag{1}$$

The notation in (1) meant that for all $\mathbf{x}$,

$$E\{\mathbf{f}(\mathbf{x}) \mid B, \Sigma, \mathbf{r}\} = \mathbf{m}(\mathbf{x})$$

and for all $\mathbf{x}$ and $\mathbf{x}'$,

$$Cov\{\mathbf{f}(\mathbf{x}), \mathbf{f}(\mathbf{x}') \mid B, \Sigma, \mathbf{r}\} = c(\mathbf{x}, \mathbf{x}')\Sigma,$$

where $c(.,.)$ is a positive-definite (correlation) function such that $c(\mathbf{x}, \mathbf{x}) = 1$ for every $\mathbf{x}$. Therefore, we assume a stationary, separable covariance structure, with covariance between the outputs at any single input given by the matrix $\Sigma = [\sigma_{jj'}] \in \mathbb{R}^+_{q \times q}$ and with $c(.,.)$ providing correlation across $\mathcal{X}$. We model the mean and covariance functions in terms of further unknown hyperparameters $B$ and $R$ by

$$\mathbf{m}(\mathbf{x}) = B^T \mathbf{h}(\mathbf{x})$$

$$c(\mathbf{x}, \mathbf{x}') = exp\{-(\mathbf{x} - \mathbf{x}')^T R(\mathbf{x} - \mathbf{x}')\} \tag{2}$$

Here $h : \mathcal{X} \mapsto \mathbb{R}^m$ is an arbitrary vector $m$ regression functions $h_1(\mathbf{x}), \ldots, h_m(\mathbf{x})$ shared by each individual function $f_j(.), j = 1, \ldots, q$; $B = [\beta_1, \ldots, \beta_q] \in \mathbb{R}_{m,q}$ is a matrix of regression coefficients; and $R$ is a diagonal matrix of $p$ positive smoothness parameters $\mathbf{r} = (r_1, \ldots, r_p)$.

The selection of the prior mean structure should be driven by both experience and simplicity; a linear specification $\mathbf{h}(\mathbf{x}) = (1, \mathbf{x})^T$ has been found to be adequate in most applications. The form of $c(.,.)$ presented in Equation (2) is seen to be mathematically convenient choice, and implies that $f_j(.)$ are infinitely differentiable with probability 1 (see Kennedy and O'Hagan, 1996).

We start by running the computer code on a pre-selected design set $\mathcal{S} = \{s_1, \ldots, s_n\} \subset \mathcal{X}$ which yields simulations organised in the output matrix $D = [f_j(s_r)] \in \mathbb{R}_{n,q}$. These are the *training data* from which the emulator is built. The set $\mathcal{S}$ is selected in accordance with some space-filling experimental design criterion (e.g., the maximum Latin-hypercube design of Morris and Mitchell, 1995).

The *multi-output emulator* is formally identified by the posterior distribution of $\mathbf{f}(.)$ given data $D$. To derive this posterior distribution, we first note that, from (1) and (2), the joint distribution of $D$ conditional on hyperparameters $B, \Sigma$ and $R$ is the matrix-normal distribution

$$D \mid B, \Sigma, R \sim \mathcal{N}_{n,q}(HB, \Sigma \otimes A) \tag{3}$$

where $H^T = [\mathbf{h}(\mathbf{s}_1), \ldots, \mathbf{h}(\mathbf{s}_n)] \in \mathbb{R}_{m,n}$, $A = [c(\mathbf{s}_r, \mathbf{s}_l)] \in \mathbb{R}^+_{n,n}$ and $\otimes$ denotes the Kronecker product operator. Standard normal theory in addition to some matrix manipulations lead to the following conditional posterior distribution for the simulator:

$$\mathbf{f}(.) \mid B, \Sigma, R, D \sim \mathcal{N}_q(\mathbf{m}^*(.), c^*(.,.)\Sigma), \tag{4}$$

where for $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$

$$\mathbf{m}^*(\mathbf{x}_1) = B^T\mathbf{h}(\mathbf{x}_1) + (D - HB)^T A^{-1}\mathbf{t}(\mathbf{x_1}),$$

$$c^*(\mathbf{x}_1, \mathbf{x}_2) = c(\mathbf{x}_1, \mathbf{x}_2) - \mathbf{t}^T(\mathbf{x}_1)A^{-1}\mathbf{t}(\mathbf{x}_1),$$

with $\mathbf{t}^T(.) = [c(., \mathbf{s}_1), \ldots, c(., \mathbf{s}_n)] \in \mathbb{R}^n$.

The posterior distribution of $\mathbf{f}(.)$ conditional on $R$ alone is found by integrating of (4) with respect to the posterior distribution of $B$ and $\Sigma$. Since any substantial information about such parameters will hardly ever be elicited from the code developers, the conventional non-informative prior $\pi^*(B, \Sigma \mid R) \propto |\Sigma|^{-\frac{q+1}{2}}$ is selected. Combining the distribution in (4) and $\pi^*$ using Bayes' theorem yields

$$\mathbf{f}(.) \mid \Sigma, R, D \sim \mathcal{N}_q(\mathbf{m}^{**}(.), c^{**}(.,.)\Sigma), \tag{5}$$

with

$$\mathbf{m}^{**}(\mathbf{x}_1) = \hat{B}^T\mathbf{h}(x_1) + (D - H\hat{B})^T A^{-1}\mathbf{t}(\mathbf{x_1})$$

$$c^{**}(\mathbf{x}_1, \mathbf{x}_2) = c^*(\mathbf{x}_1, \mathbf{x}_2) + [\mathbf{h}(\mathbf{x_1}) - H^T A^{-1}\mathbf{t}(\mathbf{x_1})]^T (H^T A^{-1} H)^{-1} [\mathbf{h}(\mathbf{x_2}) - H^T A^{-1}\mathbf{t}(\mathbf{x_2})]$$

where $\hat{B} = (H^T A^{-1} H)^{-1} H^T A^{-1} D$ is the generalised least squares estimator of $B$. Provided now that $n \geq m + q$, so that all ensuing posteriors are proper, the conditional posterior

distribution of $\mathbf{f}(.)$ given $R$ then follows the $q$-variate Student's process with $n-m$ degrees of freedom

$$\mathbf{f}(.) \mid R, D \sim \mathcal{T}_q(\mathbf{m}^{**}(.), c^{**}(.,.)\hat{\Sigma}; n-m) \tag{6}$$

where $\hat{\Sigma} = (n-m)^{-1}(D - H\hat{B})^T A^{-1}(D - H\hat{B})$.

The final step to producing the emulator is to integrate (6) with respect to the posterior distribution of $R$. Unfortunately this cannot be done analytically, and a full MCMC-based marginalisation of (6) with respect to the unknown smoothness parameters in $R$ is computationally expensive. A satisfactory alternative lies in plugging-in posterior estimates of $(r_1, \ldots, r_p)$. Kennedy and O'Hagan (2001) found uncertainty about $R$ to be relatively unimportant.

Given the estimated roughness parameters, the posterior process in (6) is the emulator of the simulator $\mathbf{f}(.)$. The mean function $\mathbf{m}^{**}$ interpolates the training data exactly and provides an approximation to $\mathbf{f}(.)$ that can be used as a fast surrogate for the simulator. It should be stressed, though, that the emulator is more than just a code's approximation, because, the estimated covariance structure $c^{**}(.,.)\hat{\Sigma}$ provides a measure of the accuracy of the approximation. The size $n$ of the design set is ideally selected to be large enough to make the emulator variance small at all points of interest in the input space.

# 2 Emulation of dynamical systems

Dynamic simulators model the evolution of state variables over a number of time steps. If we are interested in the state variables (or some transformation of them) after a fixed number of time-steps, then ordinary emulation techniques will suffice. However, if we want to emulate the behaviour of the simulator over a number of time steps, we need a different formulation. Suppose that the dynamic model produces a vector of outputs $\mathbf{y} = (y_1, \ldots, y_T)$ spanning the simulation time period $t = 1, \ldots, T$ and that a data matrix $D$ as in Subsection 1.2 is obtained from some set of training runs. In 2007, Conti and O'Hagan proposed the following procedures for emulating such a dynamic simulator.

1. The first method consists of using multi-output emulator, developed by Conti and O'Hagan (2007), where the dimension of the output space is $q = T$ and we assume that the outputs of the dynamic model is presented by $\mathbf{y} = (y_1, \ldots, y_T)$.

2. The second approach is required just one single-output emulator, following an idea originally mentioned by Kennedy and O'Hagan (2001) to analysis the spatial outputs. The time is considered as an extra input for this model, and the output $y_t = f_t(\mathbf{x})$ can now be presented as $f^*(\mathbf{x}, t)$, where $t = 1, \ldots, T$. Therefore, emulation of $\mathbf{f}(.)$ can be obtained by emulating $f^*(.,.)$. The training set for construction this emulator consists of $nT$ outputs generated by inputs in the grid $S \times \{1, \ldots, T\}$, where $X = \{x_1, \ldots, x_n\}$ denote to the pre-selected design set to obtain the outputs by running the computer code on this set.

3. The third approach is to emulate the $T$ outputs separately, each via a single-output emulator. Data for the $t$-th emulator would be then provided by the corresponding column of the data set.

Unfortunately, these emulators have some disadvantages either in terms of flexibility or computational efficiency which make the emulation of the dynamic simulators expensive or infeasible in some situations (see Conti and O'Hagan (2007)). Although these methods are theoretically straightforward, but there are two main disadvantages to use these approach directly: first, the dimension of the input space $\mathcal{X}$ becomes very large because it must include the whole time sequence of the (*forcing*[1]) inputs; second, the resulting emulator is specific to a particular $T$, and if we choose to consider runs over a different time span it will be necessary to rebuild the emulator from scratch. Conti et al (2007) proposed a new method based on the theory of emulation of multi-output models to solve the issue of emulating dynamic simulators. They broke the simulator down into its single-step action on the state variables and then emulate the required series by iteratively using an emulator of the single-step function. They also claimed that massive savings in computation time can be made by using a simple approximation. We briefly introduce this method here.

At time step $t$, the simulator may be written in the following form, $\mathbf{Y}_t = \mathbf{f}(\mathbf{z}_t, \mathbf{Y}_{t-1})$, where $\mathbf{Y}_{t-1}$ is the state vector at the start of the time step, $\mathbf{z}_t = (\mathbf{x}, \mathbf{w_t})$ subsumes both the constants $\mathbf{x}$ and the forcing inputs $\mathbf{w}_t$ at time step $t$, and the simulator outputs the new state vector $\mathbf{Y}_t$. The sequence of state vectors is written as $\{\mathbf{Y}_0, \ldots, \mathbf{Y}_T\}$.

A run of the simulator over the time steps 0 to $T$ can then be expressed recursively in terms of the single-step simulator:

$$\mathbf{Y}_T = \mathbf{f}(\mathbf{z}_T, \mathbf{Y}_{T-1}) = \mathbf{f}(\mathbf{z}_T, \mathbf{f}(\mathbf{z}_{T-1}, \mathbf{Y}_{T-2}))$$

$$= \ldots = \mathbf{f}(\mathbf{z}_T, \mathbf{f}\{\mathbf{z}_{T-1}, \ldots, \mathbf{f}(\mathbf{z}_1, \mathbf{Y}_0)\}) = \mathbf{f}^{(T)}(\mathbf{x}, \mathbf{z}, \mathbf{Y}_0) \tag{7}$$

where $\mathbf{f}^T(.)$ represents the $T$ step simulator, which takes as its inputs the constants$\mathbf{x}$, the whole sequence $\mathbf{z} = \{\mathbf{z}_t : t = 1, \ldots, T\}$ of forcing inputs and the initial state vector $\mathbf{Y}_0$. We then emulate the single-step simulator $\mathbf{f}(.)$ and use this to emulate $\mathbf{f}^{(T)}(.)$. It is claimed that the dimension of the input space, unlike the methods proposed in Conti and O'Hagan (2007), is more manageable, and the emulator can be used for simulator runs of any length. The details of this approach can be found in Conti et al. (2007).

This method could be still slow due to emulating large number of single-step functions. In the next section, we present an alternative approach which largely eliminates the need to include the time parameter namely the use of Poincaré return maps, and more generally the use of cross sections. Fast simulation of dynamical systems can then be achieved by the iteration of mappings between cross sections, together with conventional solution methods to get to and from the cross sections at the start and finish of a simulation.

# 3 Emulation of the Poincaré return map

We present a new method to study the behaviour of the dynamic simulator over a number of time steps by emulating the Poincaré return maps. The method described above to emulate the dynamic system depends on the time parameter, while our method based on emulating the Poincaré return map largely eliminates the need to include time parameter. Furthermore, in many situation, the computing the Poincaré map is an infeasible task or required expensive numerical methods.

[1]Forcing inputs vary from one time step to the next and represent external influences on the system (see Conti et al, 2007).

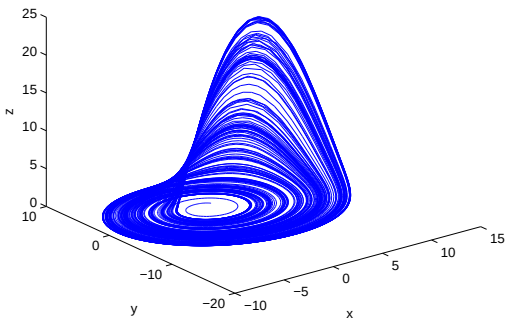


Figure 1: The strange attractor for the Rossler flow.

In this section, we use two well known examples to exhibit our method. Our method is very simple and to approximate the Poincaré map just requires few evaluations of this map at some selected initial points of the dynamic system under study. This method is also able to provide us an uncertainty bound around the approximation.

We first consider the *Rossler flow* dynamic system defined in terms of following set of differential equations:

$$\begin{aligned} \dot{x} &= -y - z \\ \dot{y} &= x + ay \\ \dot{z} &= b + z(x - c) \end{aligned} \tag{8}$$

Here, $(x, y, z) \in \mathbb{R}^3$ are dynamical variables defining the phase space and $(a, b, c) \in \mathbb{R}^3$ are parameters. We first use Rossler's parameter values $a = b = 0.2, c = 5.7$. Consider Figure 1 which shows a typical trajectory of the 3-dimensional Rossler flow. It wraps around the $z$ axis, so a good choice for a Poincaré section is a plane passing through the $z$ axis. We suggest the $y$ axis as a Poincaré section which is conveniently defined as

$$\mathcal{P} = \{(x_n, z_n) \in \mathbb{R}^2 |\ y_n = 0\}$$

Once a particular Poincaré section is picked, we could also exhibit the Poincaré return map. The use of Poincaré maps reduces the study of flows to the study of maps and also reduces the dimension of the system by 1. We therefore may restrict our attention to points exclusively lying in the Poincaré section $\mathcal{P}$. Unfortunately, except under the most trivial circumstances, the Poincaré map cannot be expressed by explicit equations. Obtaining information about the Poincaré map thus requires both solving the system of differential equations, and detecting when a point has returned to the Poincaré section. The system of differential equations could be solved by using the numerical integration methods, such as the Euler method, the midpoint Euler and the Runge-Kutta. The return map obtained from these methods required too many integration steps that could be infeasible for some complex

dynamic systems. Furthermore, the return map obtained from these methods is depended on the initial value and we cannot yield a return map corresponding to a set of initial values representing the whole system.

The emulation-based method to approximate the Poincaré return map and the first hitting time is much easier to use and does not require too many evaluations of the maps (or integration steps). We can also approximate the Poincaré return map and the first hitting map in terms of the selected initial values which covered the whole system space. These initial values can be selected by using maxmin[2] Latin hypercube design method. We denote the selected initial values by $\mathbf{X}$, and then detect their first return points to the hyperplane $\mathcal{P}$, denoted by $\mathbf{Y}$. Therefore, the data to train the Gaussian process to emulate the Poincaré return map is considered as $\mathcal{D} = \{(\mathbf{X}, \mathbf{Y})\}$.

We start with a one-dimensional return map. This map is derived by taking norm of the first return points to $\mathcal{P}$. To fit a Gaussian process to the model, we detect 20 return points lying in the Poincaré section. We detect these points in terms of the same initial value and parameter values mentioned above. We use these points to train the Gaussian process and approximate the hyperparameters of the fitted Gaussian process. The Poincaré return map is then the posterior mean of this Gaussian process. Figure 2 illustrates the emulation of the Rössler flow's return map and the 95% uncertainty bound around the posterior mean of the fitted Gaussian. It should be noticed that the method approximate the the training points with no uncertainty. Figure 3 shows the same map by using the Runge-Kutta numerical method with 10000 integration runs.

We are also able to emulate the first-hitting-time map (see Section 1.1). We first generate 8 data points on the Poincaré and then flow the system from these points to calculate their first hitting time's to the Poincaré section and their corresponding first return points. Figure 4 displays the emulation of the first-hitting-time map for the system introduced above, where $t_n$ is the first hitting time of the chosen points crossing the Poincaré section and $R_n$ is the norm of the first return points to $\mathcal{P}$. In the middle of this figure, there is considerable of uncertainty that can be reduced by choosing larger number of training points.

We now use the theory of emulation of multi-output model described in Subsection 1.2 to emulate a two-dimensional return map. The Gaussian process is trained by evaluating the Poincaré return map at five data points generated by GEM-SA software. These return points are calculated according to different initial values. Figure 5 (a) shows the emulation of the first return map and the %95 confidence bound around the posterior mean of the fitted Gaussian process. The bound around each point in the figure is very tight. A clear image of the uncertainty bound of the marked point in Figure 5 (a) is given in Figure 5 (b).

## 3.1 Bifurcation diagram

A bifurcation diagram provides a nice summary for the transition between different types of motion that can occur as one parameter of the system is varied. A bifurcation diagram plots a system parameter on the horizontal axis and a representation of an attractor on the vertical axis. In other words, given the set of differential equations that models dynamic system with one parameter:

$$\frac{dx}{dt} = f(\mathbf{x}, \alpha), \qquad \mathbf{x} \in \mathbb{R}^n, \ \alpha \in \mathbb{R}$$

[2]The GEM-SA software is able to generate these initial points according to this design.

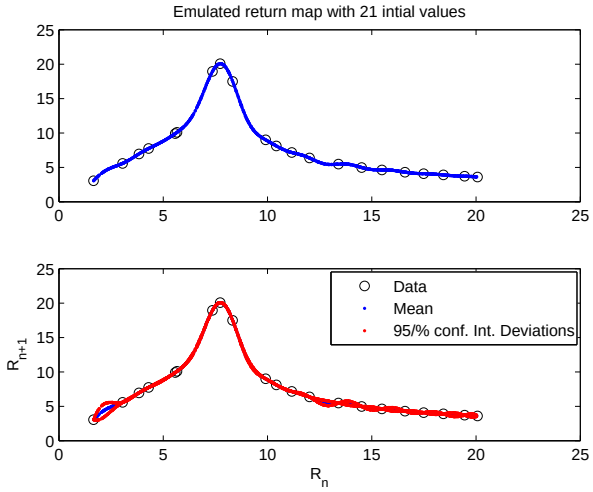


Figure 2: The emulation of the Rössler flow's return map with 21 training points and a sample of size n=401 to verify our approximation where the initial points are [0, 1, 0] and the parameters are a=b=0.2, c=5.7.

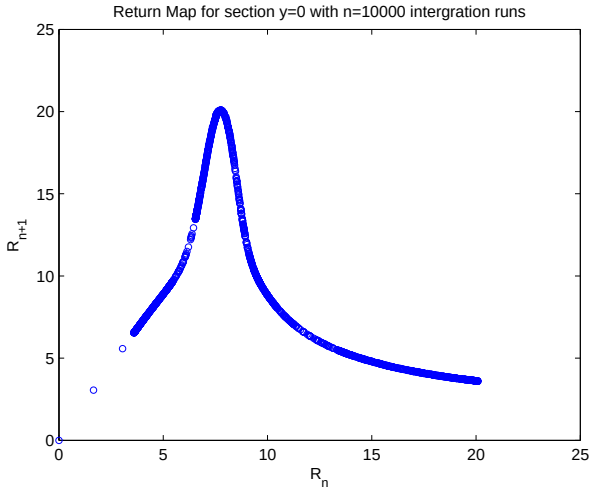


Figure 3: The first return map of the Rössler flow for section $y = 0$ by using $n = 10000$ with the same initial points and parameter values as mentioned in Figure 2.

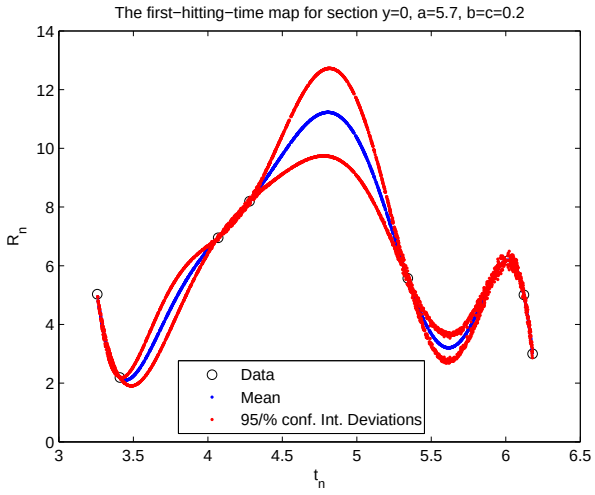


Figure 4: The emulation of the first-hitting-time map of the system shown in Figure 1 for section $y = 0$ with only 8 training points.

the generation of the bifurcation diagram consists of finding one stable fixed point and then using a continuation method to find the next fixed point based on the following condition

$$f(\mathbf{x}, \alpha) = 0 \tag{9}$$

which defines a smooth one-dimensional curve in $\mathbb{R}^n$.

In this section, we use the emulator to plot the bifurcation diagram of the Rössler flow system. Given a particular value of a parameter and a staring point, $R_0$ on the Poincaré map of the system, emulated by Gaussian process, we can determine the next point crossing the Poincaré section. We iterate this process to get the fixed point(s). We then repeat this process for all values of the parameter of interest.

Figure 6 shows the emulation of the Bifurcation diagram versus parameter $c$ of the Rössler system when we let $c$ changes in its domain and fix $a = b = 0.1$. The horizonal axis shows $c$'s values. changing from 0.5 t 40. and the vertical axis is $R_n = norm(x_n, z_n)$ where $(x_n, z_n)$ are the points on the Poincaré section $y = 0$.

# 4 Computing Derivatives

In many situations we are interested in not only the propagation of a single initial point $\mathbf{x}$, but also a whole neighbourhood of $\mathbf{x}$ in the Poincaré section $\mathcal{P}$. Such information allows us to make predictions about the future of $\mathbf{x}$, incorporating a limited amount of uncertainty in its measurement. In almost any physical application, this is very desirable, if not an absolute necessity. This prompts us to compute the matrix of derivatives (in particular, partial derivatives) $D\mathbf{P}$ of the Poincaré map $\mathbf{P}$. Once we know $D\mathbf{P}$, we can compute its eigenvalues and eigenvectors which can be used to classify periodic orbits.

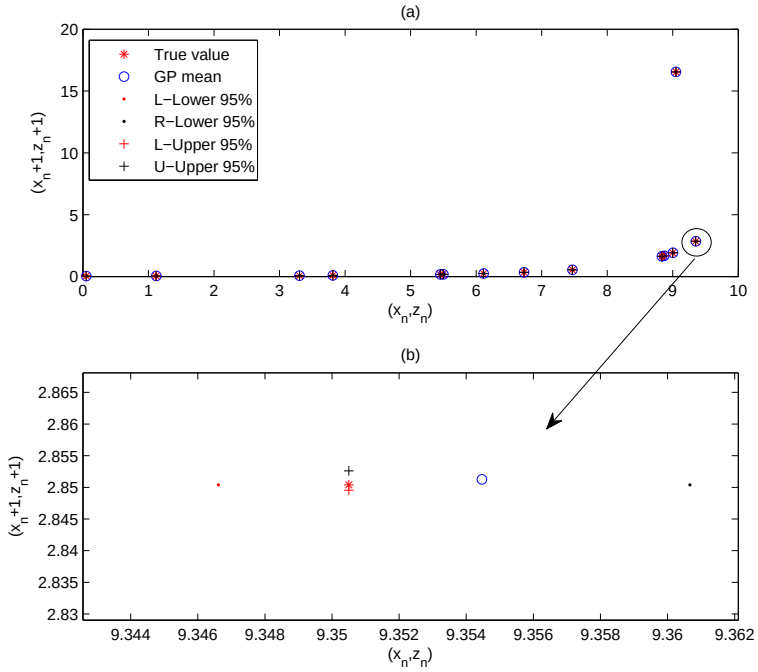


Figure 5: The emulation of two-dimensional return map for the Rössler flow where a=b=0.2, c=5.7 with 5 training data points.

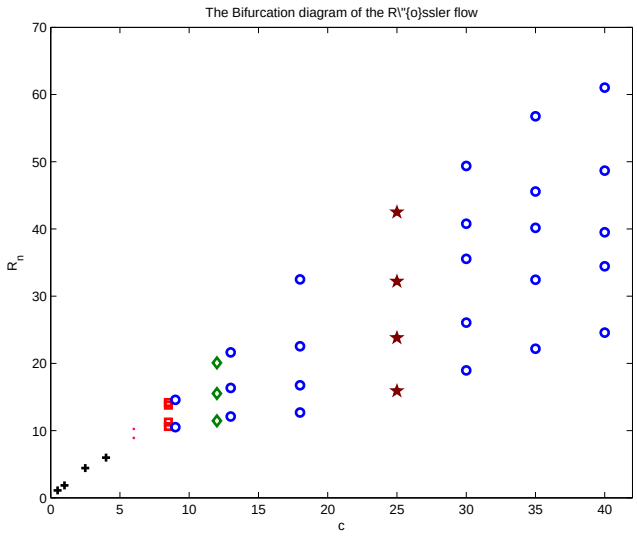


Figure 6: The emulated Bifurcation diagram versus parameter $c$ of the Rössler system.

In this section, we show how properties of the Gaussian process model allow us to include information about derivatives of the Poincaré map. The derivatives of a function $\mathbf{f}(.)$ are modelled using Gaussian processes in O'Hagan (1992). If my uncertainty about $\mathbf{f}(\mathbf{x})$ is modelled with a Gaussian distribution, then my uncertainty about any linear combination of $\mathbf{f}(.)$ is also modelled by a Gaussian distribution. My uncertainty about the derivative of $\mathbf{f}(\mathbf{x})$, defined as

$$\mathbf{f}'(\mathbf{x}) = \lim_{\mathbf{u}\to\mathbf{0}} \{\frac{1}{\mathbf{u}}\mathbf{f}(\mathbf{x}+\mathbf{u}) - \mathbf{f}(\mathbf{x})\}, \tag{10}$$

also then is modelled by a Gaussian distribution provided limit exists. In particular, if

$$E(\mathbf{f}(\mathbf{x}) \mid \mathbf{B}^T) = \mathbf{B}^T\mathbf{h}(\mathbf{x}) \quad \text{and} \quad Cov(\mathbf{f}(\mathbf{x}), \mathbf{f}(\mathbf{x}') \mid \sigma^2) = c(\mathbf{x}, \mathbf{x}')\Sigma,$$

then

$$E(\mathbf{f}^j(\mathbf{x}) \mid \mathbf{B}) = \mathbf{B}^T\mathbf{h}^j(\mathbf{x}) \tag{11}$$

$$Cov(\mathbf{f}^j(\mathbf{x}), \mathbf{f}^i(\mathbf{x}') \mid \mathbf{B}, \Sigma, \mathbf{r}) = c^{ji}(\mathbf{x}, \mathbf{x}')\Sigma \tag{12}$$

where

$$\mathbf{f}^j(\mathbf{x}) = \frac{d^j\mathbf{f}(\mathbf{x})}{d\mathbf{x}^j}, \quad \mathbf{h}^j(\mathbf{x}) = \frac{d^j\mathbf{h}(\mathbf{x})}{d\mathbf{x}^j}, \quad c^{ji}(\mathbf{x}, \mathbf{x}') = \frac{d^{i+j}c(\mathbf{x}, \mathbf{x}')}{d\mathbf{x}^j d\mathbf{x}'^i}, \tag{13}$$

assuming that these derivatives exist.

The derivative of the vector $\mathbf{h}(x)$ is reached by applying Equation (10) to each individual element. Follow the same reasoning as in Section 1.2 the posterior distribution for $\mathbf{f}^j(\mathbf{x}) \mid D$ is given by

$$\mathbf{f}^j(.) \mid R, D \sim \mathcal{T}_q(\mathbf{m}_j^{**}(.), c_j^{**}(.,.)\hat{\Sigma}; n-m) \tag{14}$$

where

$$\mathbf{m}_j^{**}(x_1) = \hat{B}^T \mathbf{h}^j(x_1) + (D - H\hat{B})A^{-1}t^j(x_1),$$

$$c_j^{**}(x_1, x_2) = c_j^*(\mathbf{x}_1, \mathbf{x}_2) + [\mathbf{h^j}(\mathbf{x_1}) - H^T A^{-1}\mathbf{t^j}(\mathbf{x_1})]^T (H^T A^{-1} H)^{-1} [\mathbf{h^j}(\mathbf{x_2}) - H^T A^{-1}\mathbf{t^j}(\mathbf{x_2})]$$

This structure gives us a means of combining information about the derivative of $f(.)$ at a point with observations directly from $f(.)$. From a technical point of view it means that when a function f is n-times continuously differentiable, we can approximate it and those derivatives simultaneously. In the language of topology we can approximate in the $C^n$ topology.

# 5 Emulation of Non-Chaotic dynamic system

One of the special features of the Rössler equations is that the flow is dissipative. Our second example also shows complex dynamics, but is a conservative system (points are not "squashed together") and hence is a qualitatively different type of example. The method described above can be applied to emulate the Poincaré return map of the non-chaotic dynamic systems. We emulate the first return map of a Billiard system as an example in this section.

A billiard is a dynamical system in which a particle alternates between motion in a straight line and reflections off a boundary. When the particle hits the boundary it reflects from it without loss of speed. Billiard dynamical systems are Hamiltonian idealizations of the game of billiards, but where the region contained by the boundary can have shapes other than rectangular and even be many dimensional. The Billiard systems capture all the complexity of Hamiltonian systems (see Goldstein, 1980), from integrability to chaotic motion, without the difficulties of integrating the equations of motion to determine its Poincar map.

We choose a rectangle table, and the initial position and velocity (angle) of a trajectory are specified through the GEM-SA software. The phase space or Poincaré section consists of two variables: the location on the boundary at which the point particle collides denoted by $t$ and defined by the parametric equations describing the table; and and the incident angle $\phi$ gives the direction of the point particle after the collision and is measured relative to the normal of the boundary at the collision point. The horizontal angle $\theta$ gives the direction of the point particle after the collision with respect to the horizontal.

The dynamic of this system can be simply represented in terms of the Poincaré return map, $P : (t_n, \theta_n) \mapsto (t_{n+1}, \theta_{n+1})$ (or $P : (t_1, \theta_1) \mapsto (t_2, \theta_2)$), from the $n$th collision to the $(n + 1)$st collision. The Poincaré map is usually approximated by the numerical methods based on the simulation of this map. The simulation techniques usually require too many function evaluations which either would be very expensive or infeasible. The emulation based method described in Subsection 1.2 can approximate this map based on few evaluations of this map.

The emulated return map, based on Gaussian process model, for the Billiard dynamic system introduced above is shown in Figure 7. We use 8 training points to estimate the hyperparameters of the Gaussian process, and we then use 15 points to verify our approximation of the Poincaré return map. This approximation is shown in Figure 7. One of the training points that their true values and their emulated values (with no uncertainty) are the same is marked by a rectangle while the uncertainty bound around a verifying point marked by a circle is also shown in this figure.

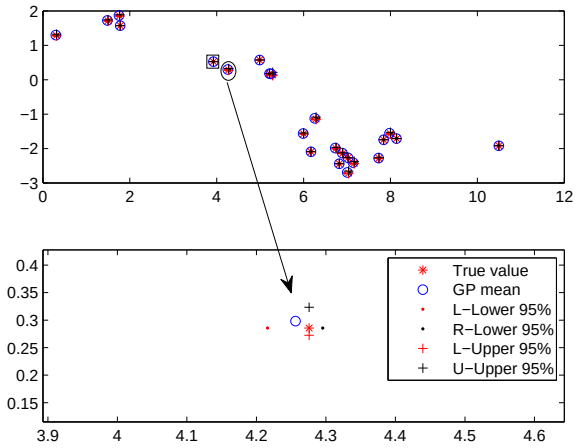


Figure 7: The emulation of the first return map of the Billiard system.

# 6 Conclusions

We present a new method to solve the problem of emulating dynamic simulations. We emulate the Poincaré return map and more generally the use of cross section. Our method gives an accurate and computationally fast method to approximate return maps which is used to study the dynamical behaviour. The need to include the time parameter by using Poincaré return maps are largely eliminated and fast simulation of dynamical systems can be achieved by the iteration of mappings between the cross sections.

This method is faster than the accurate-based computational methods (Tucker, 2002) due to using too many simulation runs, and the emulated-based method presented by Conti et al (2007) due to emulating large number of single-step functions.

The smoothness parameters of the Gussian process fitted to the data plays an important role to improve the flexibility of the multi-output emulator and it would be desirable to achieve this improvement by relaxing its assumption of a set of these parameters common to all outputs; the generality of the many single-output emulators approach in this regard is its principal benefit. Unfortunately, it seems to be very difficult to combine different smoothness parameters for each input with some covariance structure on the output space in such a way as to create a valid positive-definite correlation function. These issues require more investigation and attentions.

This method is easily applicable to emulate a coupled dynamic model. In this case, we first emulate the Poincaré return map of the first model and the evaluation of this map could be considered as input to emulate the return map of the second model. This task is under study.

# References


[1] Basu, A. (2007). Construction of Poincaré return maps for Rössler flow. The paper is available from WWW URL: http://chaosbook.org/projects/Basu/returnmap.pdf.

[2] Conti, S. and O'Hagan, A. (2007). Bayesian emulation of complex dynamic computer models. *Technical report, No 569/07*, Department of Probability and Statistics, University of Sheffield. Submitted to *Journal of Statistical and Planning Inference*.

[3] Conti, S., Gosling, J. P., Oakley, J. E. and O'Hagan, A. (2007). Gaussian process emulation of dynamic computer codes. *Research Report No. 571/07*, Department of Probability and Statistics, University of Sheffield. Submitted to *Biometrika*.

[4] Goldstein, H. (1980). *Classical Mechanics*, 2nd ed. Addison-Wesley, Reading.

[5] Henon, M. (1982). On the numerical computation of Poincaré maps. *Physica* **D**, **5**, 412-414.

[6] Hirsch, M. W., Smale, S. and Devaney, R. (2003). *Differential Equations, dynamical systems, and an introduction to chaos*. Academic Press.

[7] Katok, A. and Hasselblatt, B. (1996). *Introduction to the modern theory of dynamical systems*. Cambridge University Press.

[8] Kennedy, M. C. Anderson, C. W., Conti, S., and and O'Hagan, A. (2006). Case studies in Gaussian process modelling of computer codes. *Reliability Engineering and System Saftey*, **91**, 1301-1309.

[9] Kennedy, M. C. and O'Hagan, A. (1996). Iterative rescaling for Bayesian Quadrature. In *Bayesian Statistics 5*, Ed. J. M. Bernardo, J. O. Berger, A. P. Dawid and A. F. M. Smith, pp. 639-645. Oxford: Oxford University Press.

[10] Kennedy, M. C. and O'Hagan, A. (2001). Bayesian calibration of computer models. *J. R. Stat. Soc. Ser. B. Stat. Methodol.,* **63(3)**, 424-464.

[11] Lorenz, E. N. (1963). Deterministic non-periodic flow. *J. Atoms. Sci.*, **20**, 130-141.

[12] Morris, M. D. and Mitchell, T. J. (1995). Exploratory designs for computational experiments. *Journal of Statistical Planning and Inference*, **43**, 381-402.

[13] Morris, M. D., Mithchell, T. J. and Ylvisaker, D. (1993). Bayesian design and analysis of computer experiments: use of derivatives in surface prediction. *Technometrics*, **35**, 243-255.

[14] Oakley, J. E. and O'Hagan, A. (2002). Bayesian inference for the uncertainty distribution of computer model outputs. *Biometrika*, **89**, 769-784.

[15] Oakley, J. E. and O'Hagan, A. (2004). Probabilistic sensitivity analysis of complex models: a bayesian approach. *Journal of the Royal Statistical Society B*, **66**, 751-769.

[16] O'Hagan, A. (2006). Bayesian analysis of computer code outputs: a tutorial. *Reliability Engineering and System Saftey*, **91**, 1290-1300.

[17] O'Hagan, A, Kennedy, M. C. and Pakley, J. E. (1999). Uncertainty analysis and other inference tools for complex computer codes. In *Bayesian Statitics, 6*, pages 503-524, Oxford University Press, New York.

[18] Sacks, J., Schiller, S. B. and Welch, W. J. (1989a). Designs for computer experiments. *Technometrics*, **31**, 41-47.

[19] Sacks, J., Welch, W. J., Mithchell, T. J. and Wynn, H. P. (1989b). Design and analysis of computer experiments. *Statistical Science*, **4**, 409-435.

[20] Strogatz, S. H. (1994). *Nonlinear dynamics and chaos: with applications to physics, biology chemistry and engineering*. Addison Wesley.

[21] Tucker, W. (2002). Computing acurate Poincaré maps. *Physica* **D**, **171**, 127-137.

[22] Welch, W. J., Buck, R. J., Sacks, J., Wynn, H. P., Mithchell, T. J. and Morris, M. D. (1992). Screening, predicting, and computer experiments. *Technometrics*, **34**, 15-25.