Emulating computationally expensive dynamical simulators using Gaussian processes

Hossein Mohammadi, Peter Challenor, and Marc Goodfellow

College of Engineering, Mathematics and Physical Sciences, University of Exeter, UK

Abstract

A Gaussian process (GP)-based methodology is proposed to emulate computationally expensive dynamical computer models or simulators. The method relies on emulating the short-time numerical flow map of the model. The flow map returns the solution of a dynamic system at an arbitrary time for a given initial condition. The prediction of the flow map is performed via a GP whose kernel is estimated using random Fourier features. This gives a distribution over the flow map such that each realisation serves as an approximation to the flow map. A realisation is then employed in an iterative manner to perform one-step ahead predictions and forecast the whole time series. Repeating this procedure with multiple draws from the emulated flow map provides a probability distribution over the time series. The mean and variance of that distribution are used as the model output prediction and a measure of the associated uncertainty, respectively. The proposed method is used to emulate several dynamic non-linear simulators including the well-known Lorenz attractor and van der Pol oscillator. The results show that our approach has a high prediction performance in emulating such systems with an accurate representation of the prediction uncertainty.

Keywords: Dynamic system; Emulation; Gaussian process; Random Fourier features

1 Introduction

The problem of predicting the output of complex computer codes occurs frequently in many applications. Such simulators are computationally intensive, and hence the number of simulation runs is limited by our budget for computation. One way to overcome this problem is to create a surrogate of
the complex computer code. Surrogate models are statistical representation of the true model relying on a limited number of simulation runs and are cheap-to-evaluate. Among broad types of surrogate models, Gaussian process (GP) emulators [38] have become the gold standard for the design and analysis of computer experiments [12, 39, 40]. This is due to their statistical properties such as the flexibility and computational tractability. GPs offer a probabilistic paradigm for modeling unknown functions such that their prediction is equipped with an estimation of the associated uncertainty, see Section 2 for further detail.

An important class of computer codes are those with a time series output known as dynamical simulators. They simulate the evolution of a physical process over time and are commonly used in many applications e.g. in cardiology to study the dynamics of electrical activity of the cardiac membrane [33], and in climate science to model the Atlantic thermohaline circulation [48]. Another example is the Hindmarsh-Rose (HR) model [24] which simulates the dynamics of a single neuron. More details about the HR model and its emulation are given in Section 5.3. Emulating dynamic computer codes is an active field of research and has been tackled via different statistical and machine learning approaches, see e.g. [7, 10, 11, 34, 37]. The focus of this paper is on emulating deterministic dynamical simulators relying on a set of ordinary differential equations (ODE) using GPs.

One may consider the problem of emulating dynamical simulators as a special case of the multi-output GPs [1, 14, 20] with a temporal dependency between the outputs. However, the size of the output dimension in dynamical models is usually too high to be treated by multi-output GPs. To tackle this issue one can first apply techniques such as principal component analysis [23, 26] or wavelet decomposition [4, 17] to reduce the output dimensionality. The pitfall is that we lose some information by not keeping all the components. Another approach is proposed in [27] to account for time using an extra input parameter. This method increases the computational complexity and is reported to be inefficient [13, 32]. The idea of forecasting the time series through iterative one-step ahead predictions is developed in [5, 13]. This method relies on emulating the transition function from one time point to the next, under the assumption that the model output at time \( t \) depends only on the output at time \( t - 1 \), i.e. the Markov property. The work is continued in [34] by considering the input uncertainty in each step of the one-step ahead prediction paradigm. The method needs to estimate the correlation between emulators of state variables, and hence is not efficient if the problem dimensions, i.e. the number of variables in the ODE, are large.

This paper presents a novel approach for emulating non-linear dynamic simulators that are computationally intensive. The method is based on approximating the numerical flow map over a short period of time by a GP whose kernel is approximated via random Fourier features (RFF), see Section 3.2. The flow map can be regarded as the solution trajectory of a
A dynamical system represents the evolution of a phenomenon over time according to a fixed mathematical rule. It is expressed in terms of differential equations whose (time-varying) variables are called state variables and describe the state of the system at any given time. The space that consists of all possible values of the state variables is called the state (phase) space. The associated vector field is given by the time derivative of the state variables as follows

\[ v: \mathcal{X} \rightarrow \mathcal{X}, \quad \frac{d}{dt}x(t) = v(x(t)), \]  \hspace{1cm} (1)

in which \( x(t) = (x_1(t), \ldots, x_d(t))^\top \in \mathcal{X} \subset \mathbb{R}^d \) is the state vector at time \( t \in \mathbb{R} \) [43]. We assume that the system is autonomous which means that the vector field is not explicitly time-dependent.

Given an initial condition \( x_0 = x(0) \), the flow map \( F^t \) is a function that maps \( x_0 \) to \( x(t) \), i.e. the solution of the system obtained after the interval \([0, t]\) [16]. The flow map function is defined as

\[ F^t: \mathcal{X} \times \mathbb{R} \rightarrow \mathcal{X}, \quad (x_0, t) \mapsto F^t(x_0) = x(t), \]  \hspace{1cm} (2)

and can be interpreted as a trajectory traced out in the state space from \( x_0 \) to \( x(t) \). The flow map satisfies the following properties

\[ \bullet \quad F^0(x) = x \quad (\text{the identity property}) \]
\[ \bullet \quad F^{t+t'}(x) = F^t \left( F^{t'}(x) \right) = F^{t'} \left( F^t(x) \right) \quad (\text{the group property}), \]

for any \( x \in \mathcal{X} \).

2 Gaussian process emulators

Our aim is to build statistical representations of the flow map based on GPs to enable efficient construction of trajectories with quantified uncertainty.
The potential benefits of this approach are shown in [34]. Before explaining
the advances in the current manuscript, we introduce GP emulators. Let
\( f(\mathbf{x}), \mathbf{x} \in \mathcal{X} \), be the function we wish to emulate. In the Gaussian process
paradigm, \( f(\mathbf{x}) \) is modelled as a realisation of the stochastic process \( Y(\mathbf{x}) \) expressed by
\[
Y(\mathbf{x}) = \mu(\mathbf{x}) + Z(\mathbf{x}),
\]
where \( \mu : \mathcal{X} \mapsto \mathbb{R} \) is called the trend function. It is deterministic and can be
any function. In this work, a linear trend function is used
\[
\mu(\mathbf{x}) = q(\mathbf{x})^\top \mathbf{\beta},
\]
(4)
such that \( q(\mathbf{x}) = [q_1(\mathbf{x}), \ldots, q_r(\mathbf{x})]^\top \) is a vector of basis functions (regression
functions) and \( \mathbf{\beta} = [\beta_1, \ldots, \beta_r]^\top \) comprises the corresponding coefficients.

The second component in Equation (3), \( Z(\mathbf{x}) \), is a centred (or zero mean) GP. The stochastic process \( Y(\mathbf{x}) \) can be regarded from two perspectives
which are described below.

2.1 Function space perspective

In this framework, \( Y(\mathbf{x}) \) is deemed as a random function such that for any
finite set of inputs, the corresponding outputs have a multivariate Gaussian
distribution [38]. Properties of \( Y(\mathbf{x}) \) such as the level of smoothness is
encoded via its kernel or correlation function \( k \) defined as
\[
k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}, k(\mathbf{x}, \mathbf{x}') = \text{Corr} \left( Y(\mathbf{x}), Y(\mathbf{x}') \right).
\]
(5)
A kernel is called stationary (shift invariant) if it depends only on the dif-
fERENCE between its inputs, i.e. \( k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x} - \mathbf{x}') \). We only consider
stationary kernels throughout this paper. One of the most common sta-
tionary correlation functions is the squared exponential (SE) kernel given
by
\[
k_{SE}(\mathbf{x}, \mathbf{x}') = \exp \left( -0.5 (\mathbf{x} - \mathbf{x}')^\top \Delta^{-1} (\mathbf{x} - \mathbf{x}') \right).
\]
(6)
In the above equation, \( \Delta \) is a diagonal matrix with diagonal elements \( \delta_i \) which represent the correlation length-scale in dimension \( i, i = 1, \ldots, d \).
More details about kernels are given in Section 3.

The predictive distribution of \( Y(\mathbf{x}) \) is obtained by conditioning it on the
observations. Let \( \mathcal{D} = \{ \mathbf{X}, \mathbf{y} \} \) be the training data set in which \( \mathbf{X} = (\mathbf{x}^1, \ldots, \mathbf{x}^n)^\top \) denote a set of \( n \) inputs with the corresponding outputs \( \mathbf{y} = (f(\mathbf{x}^1), \ldots, f(\mathbf{x}^n))^\top \). For our purpose, \( \mathbf{x}^1, \ldots, \mathbf{x}^n \) represent \( n \) sample points
in the \( d \)-dimensional state space and are used to emulate the flow map, see
Section 4. Assuming that all parameters of \( Y(\mathbf{x}) \) are known, the predictive
distribution of \( Y(\mathbf{x}) \) is Gaussian whose mean and variance are
\[
\mathbb{E}[Y(\mathbf{x}) \mid \mathcal{D}] = \mu(\mathbf{x}) + \mathbf{k}(\mathbf{x})^\top \mathbf{K}^{-1} (\mathbf{y} - \mathbf{\mu}),
\]
(7)
\[
\mathbb{V} \text{ar} \left( Y(\mathbf{x}) \mid \mathcal{D} \right) = \sigma^2 \left( 1 - \mathbf{k}(\mathbf{x})^\top \mathbf{K}^{-1} \mathbf{k}(\mathbf{x}) \right),
\]
(8)
in which \( \mu = \mu(\mathbf{X}) \) and \( \sigma^2 \) is called the signal (process) variance. Also, \( \mathbf{k}(\mathbf{x}) = \begin{bmatrix} k(x^1, x) , \ldots, k(x^n, x) \end{bmatrix}^\top \) and \( \mathbf{K} \) is an \( n \times n \) correlation matrix whose \( ij \)-th element is \( K_{ij} = k(x^i, x^j), \forall x^i, x^j \in \mathbf{X} \). It is worth mentioning that the computational complexity of inverting \( \mathbf{K} \) in Equations (7) and (8) is of order \( \mathcal{O}(n^3) \).

### Parameter estimation

The stochastic process \( Y(\mathbf{x}) \) depends on a set of hyperparameters \( \theta \) that are generally unknown and need to be estimated from the data. Suppose that \( \theta = \{ \sigma^2, \mathbf{\beta}, \mathbf{\delta} \} \) is the set of hyperparameters where \( \mathbf{\delta} = [\delta_1, \ldots, \delta_d]^\top \). We use the maximum likelihood (ML) method to estimate them. The logarithm of the likelihood function is

\[
\mathcal{L}(\theta \mid \mathbf{y}) = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{1}{2} \ln(|\mathbf{K}|) - \frac{1}{2\sigma^2} (\mathbf{y} - \mu)^\top \mathbf{K}^{-1} (\mathbf{y} - \mu), \tag{9}
\]

where the correlation matrix \( \mathbf{K} \) depends on \( \mathbf{\delta} \) and \( \mu = \mathbf{Q}\mathbf{\beta} \) in which \( \mathbf{Q} = [\mathbf{q}(x^1), \ldots, \mathbf{q}(x^n)]^\top \). It is an \( n \times r \) matrix called the experimental matrix and comprises the evaluation of the regression functions at the training data. An estimate of \( \mathbf{\beta} \) and \( \sigma^2 \) is obtained by taking the derivatives of \( \mathcal{L}(\theta \mid \mathbf{y}) \) with respect to those parameters and setting the derivatives to zero. The estimated parameters have closed-form expressions given by

\[
\hat{\mathbf{\beta}} = \left( \mathbf{Q}^\top \mathbf{K}^{-1} \mathbf{Q} \right)^{-1} \mathbf{Q}^\top \mathbf{K}^{-1} \mathbf{y}, \tag{10}
\]

\[
\hat{\sigma}^2 = \frac{1}{n} \left( \mathbf{y} - \mathbf{Q}\hat{\mathbf{\beta}} \right)^\top \mathbf{K}^{-1} \left( \mathbf{y} - \mathbf{Q}\hat{\mathbf{\beta}} \right). \tag{11}
\]

If the parameters \( \mathbf{\beta} \) and \( \sigma^2 \) in Equation (9) are substituted with their estimates \( \hat{\mathbf{\beta}} \) and \( \hat{\sigma}^2 \), the profile log-likelihood (after dropping the constants) is achieved as

\[
\mathcal{L}_p(\mathbf{\delta} \mid \mathbf{y}) = -\frac{n}{2} \ln(\hat{\sigma}^2) - \frac{1}{2} \ln(|\mathbf{K}|). \tag{12}
\]

Finally, the length-scales can be estimated by solving the optimisation problem below

\[
\hat{\mathbf{\delta}} = \arg \max_{\mathbf{\delta}} \mathcal{L}_p(\mathbf{\delta} \mid \mathbf{y}). \tag{13}
\]

### 2.2 Weight space perspective

In this view, a GP is represented by a weighted sum of (possibly infinite) basis functions as

\[
Z(\mathbf{x}) = \phi(\mathbf{x})^\top \mathbf{w}, \tag{14}
\]

where \( \mathbf{w} \) is the weight vector and \( \phi(\mathbf{x}) \) represents the vector of the basis functions. The operator \( \phi(\mathbf{x}) \) is called the feature map and transfers the input space \( \mathcal{X} \) to the reproducing kernel Hilbert space \( \mathcal{H} \) [2]. It can be shown that the value of \( k(x, x') \) is equivalent to the inner product between \( x \) and \( x' \) in \( \mathcal{H} \), see Section 3.

5
According to Equation (14), the stochastic process $Y(x)$ can be written as $Y(x) = \mu(x) + \phi(x)^\top w$. Now, if an independent Gaussian distribution is considered for the weights as $w \sim \mathcal{N}(0, \sigma^2 I)$, the stochastic process $Y(x)$ is characterised by

\[
\mathbb{E}[Y(x)] = \mathbb{E}[\mu(x) + \phi(x)^\top w] = \mu(x), \quad (15)
\]
\[
\text{Cov}(Y(x), Y(x')) = \mathbb{E}\left[\left(\phi(x)^\top w\right)\left(\phi(x')^\top w\right)\right] = \phi(x)^\top \mathbb{E}[ww^\top] \phi(x') = \sigma^2 \phi(x)^\top \phi(x'). \quad (16)
\]

The conditional distribution of the weights $p(w \mid D)$ has the following mean and covariance expressions [44]

\[
\mathbb{E}[w \mid D] = \Phi^\top \left(\Phi\Phi^\top\right)^{-1} (y - \mu), \quad (17)
\]
\[
\text{Cov}(w \mid D) = \sigma^2 \left(I - \Phi^\top \left(\Phi\Phi^\top\right)^{-1} \Phi\right), \quad (18)
\]

in which $\Phi = [\phi(x_1), \ldots, \phi(x_n)]^\top$. Accordingly, the predictive (conditional) mean and variance of $Y(x) = \mu(x) + \phi(x)^\top w$ are

\[
\mathbb{E}[Y(x) \mid D] = \mu(x) + \phi(x)^\top \Phi^\top \left(\Phi\Phi^\top\right)^{-1} (y - \mu), \quad (19)
\]
\[
\mathbb{V}[Y(x) \mid D] = \sigma^2 \left(1 - \phi(x)^\top \Phi^\top \left(\Phi\Phi^\top\right)^{-1} \Phi\phi(x)\right). \quad (20)
\]

Note that the computational complexity of $(\Phi\Phi^\top)^{-1}$ is a function of the dimensionality of the feature map and not the number of data points $n$.

### 3 Kernel approximation

#### 3.1 Background

This section provides the material necessary for approximating a kernel with random Fourier features. A GP whose kernel is approximated this way offers a probability distribution over its predictive mean and allows us to draw realisations from the predictive mean distribution. This idea is used in Section 4 to emulate dynamical simulators where multiple realisations from the emulated flow map are employed iteratively to perform one-step ahead predictions. First, we start the discussion by rigorously defining kernels.

**Definition 1 (Kernel)**

Suppose $\mathcal{H}$ is a Hilbert space equipped with the inner product $\langle \cdot, \cdot \rangle_\mathcal{H}$. A symmetric function of two variables $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ is called a kernel if and only if there exists a feature map $\phi : \mathcal{X} \mapsto \mathcal{H}$ such that

\[
k(x, x') = \langle \phi(x), \phi(x') \rangle_\mathcal{H}, \quad \forall x, x' \in \mathcal{X}. \quad (21)
\]
A kernel is a positive semidefinite (PSD) function meaning that for any nonzero coefficient vector \( \mathbf{a} = [a_1, \ldots, a_N]^{\top} \in \mathbb{R}^N \) (\( N \in \mathbb{N} \) is an arbitrary natural number) the following equation holds

\[
\sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0, \quad \forall \mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathcal{X}.
\] (22)

**Definition 2 (Reproducing kernel Hilbert space)**

Let \( \mathcal{H} \) be a Hilbert space of functions defined on \( \mathcal{X} \). The function \( k(\cdot, \cdot) \) is called a reproducing kernel of \( \mathcal{H} \), and \( \mathcal{H} \) is a reproducing kernel Hilbert space (RKHS), if it satisfies

1. \( \forall \mathbf{x} \in \mathcal{X} \rightarrow k(\mathbf{x}, \cdot) \in \mathcal{H} \); and
2. \( \forall \mathbf{x} \in \mathcal{X} \) and \( \forall f \in \mathcal{H}, \langle f, k(\cdot, \mathbf{x}) \rangle_{\mathcal{H}} = f(\mathbf{x}) \) (the reproducing property).

Note that a reproducing kernel is positive semidefinite [2]. For the reproducing kernel \( k \) and the feature map \( \phi : \mathbf{x} \mapsto k(\mathbf{x}, \cdot) \), we have

\[
\langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle_{\mathcal{H}} = \langle k(\mathbf{x}, \cdot), k(\mathbf{x}', \cdot) \rangle_{\mathcal{H}} = k(\mathbf{x}, \mathbf{x}'), \quad \forall \mathbf{x}, \mathbf{x}' \in \mathcal{X},
\] (23)

which follows directly from the reproducing property. Equation (23) suggests that the input space \( \mathcal{X} \) can be projected into a higher (or infinite) dimensional feature space (through \( \phi \)) where the learning procedure can be more successful. For example, in Figure 1 the elements of two data sets (blue triangles and red circles) are not linearly separable in \( \mathbb{R}^2 \) (left picture). However, once the data is transformed into \( \mathbb{R}^3 \) (right picture), the data can be separated into two classes by a linear hyperplane. It is worth mentioning that such projection is never performed in practice since kernels compute the inner product \( \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle_{\mathcal{H}} \) implicitly. This property is known as the kernel trick in machine learning [41, 25].

### 3.2 Random Fourier features

The technique of random Fourier features (RFF) developed by Rahimi and Recht [36] is an effective way for approximating stationary kernels. The RFF method is a consequence of the Bochner’s theorem [6] described below.

**Definition 3 (Bochner’s theorem)**

A stationary kernel is PSD if and only if it is a Fourier transform of a non-negative measure.

According to the Bochner’s theorem, the stationary kernel \( k \) can be expressed as

\[
k(\mathbf{x}, \mathbf{x}') = \int e^{-i\omega^{\top}(\mathbf{x}-\mathbf{x}')} dP(\omega),
\] (24)
where $\mathbb{P}(\omega)$, the Fourier dual of $k$, is equal to the spectral density of the kernel. $\mathbb{P}(\omega)$ has all the properties of a cumulative distribution function except that $\mathbb{P}(\infty) - \mathbb{P}(-\infty) = k(0)$ needs not to be equal to one [30]. Thus, $\mathbb{P}(\omega)$ becomes a proper cumulative distribution function if it is normalised by $k(0)$, i.e. $\mathbb{P}(\omega)/k(0)$. For classic correlation functions such as the SE kernel, no normalisation is required because $k(0) = 1$. In this situation, $p(\omega) = \frac{d\mathbb{P}(\omega)}{d\omega}$ is the density function of $\omega$ and Equation (24) can be rewritten as

$$k(x, x') = \int e^{-i\omega^\top(x-x')} p(\omega) d\omega = \mathbb{E}_{p(\omega)} \left[e^{-i\omega^\top(x-x')}\right]$$

$$= \mathbb{E}_{p(\omega)} \left[\text{Re} \left(e^{-i\omega^\top x} (e^{-i\omega^\top x'})^* \right) \right]$$

$$= \mathbb{E}_{p(\zeta)} \left[\zeta(x)^\top \zeta(x') \right]. \quad (25)$$

Here, the superscript $*$ denotes the complex conjugate and $\zeta(\cdot)$ is a random feature map. Note that there is no imaginary component in Equation (25) because we work with real-valued kernels. A possible choice for $\zeta(\cdot)$ is

$$\zeta(x) = \sqrt{2} \cos \left(\omega^\top x + b \right), \quad (26)$$

in which $b \sim U[0, 2\pi]$ is a uniform random variable [36, 22]. The distribution of $\omega$ depends on the type of correlation function. For example, the spectral density $p(\omega)$ of the Matérn kernel is a $t$-distribution and for the SE kernel is Gaussian specified by [36, 46]

$$\omega_{SE} \sim \mathcal{N} \left(0, \Delta^{-1} \right). \quad (27)$$
The explicit random feature map $\zeta(x)$ given by Equation (26) allows us to approximate the stationary kernel $k$ using a Monte-Carlo approach. To do this, first we need to take $M$ independently and identically distributed (i.i.d.) samples from $p(\omega)$ and $p(b) = U[0, 2\pi]$. Then the vector
\[
\zeta(x) = \sqrt{\frac{2}{M}} \left[ \cos(\omega^{(1)}^T x + b^{(1)}), \ldots, \cos(\omega^{(M)}^T x + b^{(M)}) \right]^T,
\]
(28)
is created which transfers $x \in \mathcal{X}$ into an $M$-dimensional feature space. Finally, the correlation function $k$ is approximated as below
\[
k(x, x') = \mathbb{E}_{p(\zeta)} \left[ \zeta(x)^T \zeta(x') \right] \approx \zeta(x)^T \zeta(x').
\]
(29)

### 3.3 GPs with random Fourier features

In the weight space representation of GPs (Equation (14)), the actual feature map $\phi(x)$ can be approximated via the $M$-dimensional random feature map $\zeta(x)$ given by Equation (28). We note that the feature map $\phi(x)$ is possibly infinite dimensional. In a similar fashion, the predictive mean of $Y(x)$ (Equation (19)) can be approximated as
\[
\mathbb{E}[Y(x) | \mathcal{D}] \approx \mu(x) + \zeta(x)^T \hat{\Phi}^T \left( \hat{\Phi} \hat{\Phi}^T \right)^{-1} (y - \mu),
\]
(30)
wherein $\hat{\Phi} = \left[ \zeta(x^1), \ldots, \zeta(x^n) \right]^T$. The approximation of $\mathbb{E}[Y(x) | \mathcal{D}]$ obtained by (30) is stochastic since the construction of $\zeta(x)$ relies on i.i.d. samples taken from the kernel spectral density $p(\omega)$ and $p(b)$, see Equation (28). As a result, one can generate realisations of the approximated predictive mean by drawing samples from $p(\omega)$ and $p(b)$ repeatedly. This idea is used in Section 4 for emulating dynamic simulators where the short-time numerical flow map is approximated with RFF. Algorithm 1 outlines the steps to draw realisations from the approximated predictive mean. In the algorithm, the notation $\hat{f}^{(s)}(x)$ indicates the $s$-th approximation to the predictive mean because it estimates the true function $f$. These samples can be used to quantify uncertainty of prediction in the absence of a closed-form expression like Equation (8).

It is worth mentioning that increasing the dimensionality of random feature space $M$, yields a better approximation to the predictive mean. For example, Figure 2 visualises the predictive mean of a GP (dashed) based on Equation (7) and three realisations (blue) of its approximation relying on (30). The trend function is $\mu(x) = x$. The correlation function is the squared exponential kernel whose length-scale is equal to one. The dimensionality of random feature space is $M = 5$ and $M = 50$ on the left and right pictures, respectively. As can be seen, the approximations are closer to the actual predictive mean on the right picture with a larger value of $M$. However, increasing $M$ is at the cost of a higher computational complexity.
Figure 2: The predictive mean of a GP (dashed) based on Equation (7) with \( \mu(x) = x \) (dotted) and the squared exponential kernel. The black dots are the training data and the blue lines are realisations of the approximated predictive mean obtained by (30). The dimensionality of random feature space to create \( \zeta(x) \) is \( M = 5 \) (left) and \( M = 50 \) (right). Increasing \( M \) results in a better approximation to the predictive mean.

**Algorithm 1** Drawing realisations from approximated predictive mean

Input: training set \( \mathcal{D} \), kernel \( k \) and its spectral density \( p(\omega) \), \( M \): dimensionality of random feature space

1. Estimate \( \theta \) using the ML method
2. for \( s = 1 \) to \( S \) do
3. Draw \( M \) i.i.d. samples from \( p(\omega) \) and \( p(b) = \mathcal{U}[0, 2\pi] \)
4. Construct \( \zeta(x) \) (Equation (28))
5. Obtain \( \hat{f}^{(s)}(x) \): approximation of \( \mathbb{E}[Y(x) \mid \mathcal{D}] \) (Equation (30))
6. end for

Output: \( S \) approximations of \( \mathbb{E}[Y(x) \mid \mathcal{D}] : \hat{f}^{(1)}(x), \ldots, \hat{f}^{(S)}(x) \)
4 Emulating dynamical simulators

In this section, we introduce our methodology for emulating deterministic dynamical simulators that are computationally intensive. The method relies on emulating the numerical flow map over a short period using RFF. Let $x(t_1) = (x_1(t_1), \ldots, x_d(t_1))^\top$ be the solution of the system at $t_1 = t_0 + \Delta t$ where $\Delta t$ is a fixed “small” time step given the initial condition $x_0$. We assume that $x(t_1)$ is obtained by the (black-box) flow map function defined as

$$F^{t_1}(x_0) = (f_1(x_0), \ldots, f_d(x_0))^\top = (x_1(t_1), \ldots, x_d(t_1))^\top.$$  \hfill {31}

Here, each map $f_i : \mathcal{X} \mapsto \mathbb{R}$ yields the $i$-th component of $x(t_1)$, i.e. $x_i(t_1)$. A forecast of the $i$-th component (denoted by $\hat{x}_i(t)$) is obtained by:

- emulating $f_i$ with RFF
- drawing a realisation from the predictive mean of $f_i$ (denoted by $\hat{f}_i^{(s)}(\cdot)$) using Algorithm 1
- employing $\hat{f}_i^{(s)}(\cdot)$ iteratively to perform one-step ahead predictions

Repeating the above steps leads to a probability distribution over the time series. The mean and variance of that distribution can serve as the model output prediction and the associated uncertainty, respectively. The proposed emulation method is summarised in Algorithm 2.

Algorithm 2 Emulating dynamic non-linear simulators

1: Select $n$ space-filling initial conditions: $X = \{x^1_0, \ldots, x^n_0\}$
2: Run the simulator for each $x^1_0, \ldots, x^n_0$ over $\Delta t$ to obtain $y = \{x^1(t_1), \ldots, x^n(t_1)\}$
3: for $i = 1$ to $d$ do
4:   Use training set $D_i = \{X, y_i\}$ where $y_i = (x^1_i(t_1), \ldots, x^n_i(t_1))^\top$ to emulate $f_i$
5:   Generate $\hat{f}_i^{(1)}(\cdot), \ldots, \hat{f}_i^{(S)}(\cdot)$ using Algorithm 1
6:   for $s = 1$ to $S$ do
7:     Use $\hat{f}_i^{(s)}(\cdot)$ iteratively to perform one-step ahead predictions to forecast the time series
8:     end for
9:   Calculate the mean and variance of the obtained time series
10: end for

The first step of Algorithm 2 is to choose $n$ sample points $X = \{x^1_0, \ldots, x^n_0\}$ from the space of initial conditions. The points are selected in a carefully designed experiment such that they fill the space uniformly. This quality is called “space-filling” and can be achieved using techniques such as Latin hypercube sampling (LHS) [19, 40]. The next step is to run the simulator
at $x_0^1, \ldots, x_0^n$ over the time horizon $\Delta t$ to obtain the corresponding outputs denoted by $y = \{x^1(t_1), \ldots, x^n(t_1)\}$. The training data set $D_i = \{X, y_i\}$ where $y_i = (x^1_i(t_1), \ldots, x^n_i(t_1))^T$ is used to estimate the unknown parameters of the $i$-th emulator, $\theta_i$. Finally, multiple approximations of $f_i(x)$ denoted by $\hat{f}^{(s)}_i(x)$, $s = 1, \ldots, S$, are employed in an iterative manner to perform one-step ahead predictions.

5 Numerical results

The prediction performance of our proposed method is tested on several dynamic systems implemented as computer codes. They are the Lorenz [31], van der Pol [43] and Hindmarsh-Rose models [24]. These systems are further elaborated in the following subsections. The training data to emulate the short-time numerical flow map consists of $n = 15 \times d$ points selected from the space of initial conditions. We use the optimumLHS function implemented in the R package lhs [9] to sample the initial conditions. This function maximises the mean distance between all candidate points, and hence yields a space-filling design. The GP correlation function is the squared exponential kernel which is recommended in [13, 34]. The kernel is approximated using RFF with $M = 250$ as is considered in [22]. The number of realisations drawn from the emulated flow map is $S = 100$. The trend function is a first order polynomial regression whose coefficients are estimated via Equation (10). The length-scales and process variance are estimated by the ML method. The simulation time step is fixed and equal to $\Delta t = 0.01$. The ODEs are solved by the default solver of the package deSolve [42].

In this work, the forecast accuracy is measured via the mean absolute error (MAE) and root mean square error (RMSE) criteria. They are defined as

$$MAE = \frac{\sum_{t=0}^{T} |x(t) - \hat{x}(t)|}{n_{\text{step}}}$$

$$RMSE = \sqrt{\frac{\sum_{t=0}^{T} (x(t) - \hat{x}(t))^2}{n_{\text{step}}}}.$$  

where $T$ is the final time of the simulation and $n_{\text{step}} = T/\Delta t$ is the total number of one-step ahead predictions. A criterion called the “predictability horizon” is suggested in [34] as the time after which the emulator is less reliable. It represents the time at which a change point occurs in the evolution of prediction uncertainties, see below for further details. The predictability horizon (denoted by $t^*$) is computed for all test cases. Finally, we compare the results obtained by our method with the one presented in [34]. We refer to our proposed approach as “Method 1” and the other one as “Method 2” in the rest of the paper.
5.1 Lorenz attractor

The Lorenz model is a system of ODE first derived by Edward Lorenz in 1963 [31]. Although the Lorenz system was originally developed as a model of convection in the earth’s atmosphere, it has applications to other fields, see e.g. [35]. The Lorenz system can produce a famous chaotic attractor whereby trajectories on the attractor are very sensitive to initial conditions. In other words, the evolution of the system starting from two slightly different points will be entirely dissimilar. An attractor is a bounded area in the state space such that the trajectories of all nearby points converge to it [45]. The Lorenz equations are expressed by

\[
\begin{align*}
\frac{dx}{dt} &= a_1 x_1 + x_2 x_3 \\
\frac{dx_2}{dt} &= a_2 (x_2 - x_3) \\
\frac{dx_3}{dt} &= -x_1 x_2 + a_3 x_2 - x_3,
\end{align*}
\]

with the classic parameter values of \( a_1 = -8/3, a_2 = -10 \) and \( a_3 = 28 \). These values result in the well-know “butterfly attractor” as can be seen in Figure 3.

The emulation of the Lorenz model with Methods 1 and 2 is displayed in Figures 3 and 4, respectively. The parameters \( a_1, a_2 \) and \( a_3 \) are set to the values as above and the initial condition in both figures is \( x_0 = (1, 1, 1)^\top \) displayed by a red point in the three-dimensional picture. It demonstrates the evolution of the state variables together. Throughout this paper, the simulation is shown in red and emulation in black; the credible intervals (prediction ± standard deviation of prediction) is shaded. The vertical dashed blue lines indicate the predictability horizon obtained by applying the \texttt{cpt.mean} function implemented in the \texttt{changepoint} R package [28] to the standard deviation (SD) of predictions. For example, Figure 5 illustrates the SD of predictions associated with the variables \( x_1 \) and \( x_2 \) in the Lorenz model. The change points (vertical lines) show the time at which a drastic change happens in the prediction uncertainties.

The two methods have similar performances in emulating the Lorenz system. The prediction accuracy is high up to about \( t \approx 14 \). After that time, the emulator deviates from the true model which has a chaotic behaviour and tends to the average of the process. At the same time, the prediction uncertainties blow which allows the credible intervals to encompass most values of the system. The MAE and RMSE criteria are computed for both approaches and presented together with the predictability horizon in Table 1. For each state variable, the first row is acquired using Method 1 and the second row with Method 2. The table also contains the results from emulating the Lorenz model with another initial condition, i.e. \( x_0 = (-1, -1, -1)^\top \). As can be seen, the values obtained by the two methods are close to each other suggesting that they are comparable.
Figure 3: The prediction (black) and associated uncertainty (shaded) in emulating the Lorenz model (red) based on Method 1. The vertical dashed blue lines represent the predictability horizon which are the change point in the diagram of prediction uncertainties. The initial condition is $x_0 = (1, 1, 1)^\top$ which is shown by a red point in the three-dimensional picture. It shows the evolution of the whole system.
Figure 4: The Lorenz model (red) and its emulation (black) obtained by Method 2. The vertical dashed blue lines show the predictability horizon. The initial condition and the parameter values are the same as Figure 3.
Figure 5: Standard deviation (SD) of prediction associated with the variables $x_1$ (black) and $x_2$ (red) in the Lorenz model. The vertical dashed lines show the change point of each diagram. The y-axis is on logarithmic scale.

Table 1: The MAE and RMSE criteria and predictability horizon of Method 1 (first row) and Method 2 (second row) obtained by two different initial conditions.

| $x_0 = (1, 1, 1)^\top$ | $x_0 = (-1, -1, -1)^\top$ |
|------------------------|------------------------|
| **$x_1$**              | **$x_2$**              | **$x_3$**              |
| MAE        | RMSE       | $t^*$       | MAE        | RMSE       | $t^*$       | MAE        | RMSE       | $t^*$       |
| 3.34       | 5.46       | 13.68       | 2.43       | 5.06       | 20.46       | 2.41       | 4.68       | 20.48       |
| 5.06       | 7.47       | 14.48       | 2.14       | 4.13       | 20.47       | 2.30       | 4.45       | 20.52       |
| 3.99       | 6.34       | 13.71       | 2.14       | 4.13       | 20.47       | 2.30       | 4.45       | 20.52       |
| 3.97       | 6.14       | 14.44       | 2.34       | 4.71       | 20.41       | 2.48       | 4.98       | 20.44       |
5.2 Van der Pol oscillator

The van der Pol equation was first proposed by the Dutch engineer Balthasar van der Pol in 1920 while working on the behaviour of vacuum tube circuits. Since then, this model has been extensively used to study oscillatory phenomena such as biological rhythms [47], heartbeat [18] and circadian rhythms [8]. The evolution of the van der Pol model in time is of the form [43]

\[
\begin{align*}
\frac{dx_1}{dt} &= x_2 \\
\frac{dx_2}{dt} &= a(1 - x_1^2)x_2 - x_1,
\end{align*}
\]

(35)

where the parameter \(a > 0\) controls the frequency of the oscillations. The van der Pol oscillator exhibits a periodic motion, i.e. a limit cycle, see Figure 6. The emulation of the van der Pol oscillator based on Methods 1 and 2 is visualised in Figures 6 and 7, respectively. The truth (red), prediction (black), credible intervals (shaded) and predictability horizon (dashed blue) are illustrated in the figures. The initial condition displayed by a red point in the two-dimensional picture is \(x_0 = (1, 1)\top\) and \(a = 5\) in both cases.

As can be seen, Method 1 outperforms Method 2 in emulating the van der Pol model. Specially, the first state variable is predicted with a high accuracy by Method 1 and the predictability horizon is equal to the total simulation time. There is a frequency miss-match in the predictions obtained by Method 2 after the predictability horizon. Moreover, the amplitude of the predictions (especially \(\hat{x}_2(t)\)) in Method 2 gradually damps. However, this is a less severe problem in Method 1. The two methods are compared according to the MAE, RMSE and predictability horizon criteria. They are presented in Table 2 which also accommodates the measures obtained by an additional initial condition, i.e. \(x_0 = (-1, -1)\top\). The criteria summarised in Table 2 suggest that Method 1 has a better prediction performance (especially in the case of the first state variable) than Method 2.

Table 2: The MAE and RMSE criteria and predictability horizon obtained by Methods 1 and 2. For each variable, the first row consists of the criteria associated with Method 1 and the second row represents those of Method 2.

| \(x_0 = (1, 1)\top\) | \(x_0 = (-1, -1)\top\) | \(x_1\) | \(x_2\) |
|------------------|------------------|--------|--------|
| **MAE** | **RMSE** | **\(t^*\)** | **MAE** | **RMSE** | **\(t^*\)** |
| 0.02 | 0.07 | 75 | 0.03 | 0.08 | 75 |
| 1.18 | 1.82 | 29.31 | 1.16 | 1.80 | 29.27 |
| 0.11 | 0.40 | 36.44 | 0.12 | 0.40 | 36.44 |
| 0.83 | 1.67 | 24.73 | 0.83 | 1.67 | 24.71 |
Figure 6: The van der Pol oscillator (red), the prediction (black) and credible intervals (shaded) using Method 1. The dashed blue lines show the predictability horizon. The initial condition (red point) and parameter value are $x_0 = (1, 1)^T$ and $a = 5$. 
Figure 7: The van der Pol oscillator (red) and its emulation (black) using Method 2 where $x_0 = (1, 1)^T$ and $a = 5$. The dashed blue lines are the predictability horizons. A frequency mismatch happens after the predictability horizon and the amplitude of the predictions gradually damp.
5.3 Hindmarsh-Rose model

The Hindmarsh-Rose (HR) model [24] is widely used in neuroscience to simulate the nonlinear dynamics of a single neuron. Neurons are specialised cells that are responsible for generating electrical signals called action potentials by which information is transmitted throughout the nervous system. The HR model is capable of mimicking spiking and bursting which can be seen to occur in real cells. The mathematical equations of the HR model are

\[
\begin{align*}
\frac{dx_1}{dt} &= x_2 - a_1 x_1^3 + a_2 x_2^2 - x_3 + I \\
\frac{dx_2}{dt} &= a_3 - a_4 x_1^2 - x_2 \\
\frac{dx_3}{dt} &= \varepsilon (a_5 (x_1 - x_{rest}) - x_3),
\end{align*}
\]

wherein \( x_1 \) represents the cell membrane potential. The variables \( x_2 \) and \( x_3 \) describe the ionic currents flowing across the membrane through fast and slow ion channels, respectively. The parameter \( 0 < \varepsilon \ll 1 \) is small, which makes \( x_3 \) a slow variable. \( I \) represents the membrane input current and \( x_{rest} \) is the rest potential of the system. Having studied a limit cycle and a chaotic behaviour, we then choose in the HR model a complex transient trajectory, where the two time scales interplay. The study of transient dynamics is important in many real-world phenomena, see e.g. [21, 29]. To this end, in our experiments the value of the parameters \( \varepsilon, I \) and \( x_{rest} \) are set to 0.01, 2.4 and -1.6, respectively. The constant parameters \( a_1, \ldots, a_5 \) are determined experimentally [15]; their typical values which are considered in the examples below are: \( a_1 = 1, a_2 = 2.7, a_3 = 1, a_4 = 5 \) and \( a_5 = 4 \) [3].

Figures 8 and 9 present the HR model (red) and its emulation (black) based on Methods 1 and 2, respectively. In both figures, the initial condition is \( x_0 = (1, 1, 1)^\top \). As can be seen, Method 1 has a superior performance compared to Method 2 in forecasting all three variables. Particularly, in Method 1 the emulator remains reliable until the end of simulation and the predictability horizon is equal to: \( t^* = 100 \). Methods 1 and 2 are compared according to the MAE, RMSE and predictability horizon criteria in Table 3 with an extra initial condition \( x_0 = (-1, -1, -1)^\top \). The criteria obtained by Method 1 are clearly better than those of Method 2. It can be seen that the values of MAE and RMSE in Methods 1 do not usually exceed 0.07, indicating a high prediction accuracy.

6 Conclusions

This paper proposes a novel approach for emulating computationally expensive dynamic computer codes whose output is a time series. The method is based on approximating the short-time numerical flow map and using the emulated flow map in an iterative manner to perform one-step ahead predictions. The flow map is a function that gives the solution of a dynamic system
Figure 8: Emulating the HR model with the initial condition $x_0 = (1, 1, 1)^\top$ based on Method 1. The proposed approach has a high prediction performance such that the difference between the truth (red) and emulation (black) is negligible. The predictability horizon (dashed blue) occurs at the end of the simulation for all three variables.

Table 3: Comparison of Methods 1 and 2 in emulating the HR model. For each variable, the predictive performance of Methods 1 and 2 is presented in the first and the second row, respectively.

|   | $x_0 = (1, 1, 1)^\top$ |   | $x_0 = (-1, -1, -1)^\top$ |
|---|------------------------|---|--------------------------|
|   | MAE  | RMSE | $t^*$ | MAE  | RMSE | $t^*$ |
| $x_1$ | 0.01 | 0.03 | 100 | 0.02 | 0.04 | 79.53 |
|     | 0.48 | 0.64 | 71.18 | 0.45 | 0.61 | 28.58 |
| $x_2$ | 0.04 | 0.07 | 100 | 0.16 | 0.34 | 81.91 |
|     | 3.45 | 5.10 | 16.19 | 1.46 | 2.02 | 29.08 |
| $x_3$ | 0.001 | 0.002 | 100 | 0.008 | 0.02 | 100 |
|     | 0.54 | 0.71 | 100 | 0.35 | 0.50 | 100 |
Figure 9: Emulating the HR model with the initial condition $x_0 = (1, 1, 1)^\top$ using Method 2. The simulation and emulation are shown in red and black, respectively. This approach has a lower prediction performance compared to Method 1, see Figure 8.
at an arbitrary time for a given initial condition. The main contribution of this paper is that the flow map function is emulated by a GP whose kernel is estimated using RFF. This allows us to draw multiple realisations from the emulated flow map. When the realisations are employed in the one-step ahead prediction paradigm, a distribution over the time series is created. The mean and variance of that distribution serve as the prediction and a measure of the associated uncertainty, respectively. The proposed method is tested on several non-linear dynamic simulators including the Lorenz attractor and van der Pol oscillator. The results suggest that our approach can emulate those models precisely with an accurate representation of the prediction uncertainty.

Acknowledgments

The authors (HM and PC) would like to thank the Alan Turing Institute for funding this work.

References

[1] Mauricio A. Álvarez and Neil D. Lawrence. Computationally efficient convolved multiple output Gaussian processes. *Journal of Machine Learning Research*, 12(41):1459–1500, 2011.

[2] N. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68(3):337–404, 1950.

[3] Roberto Barrio and Andrey Shilnikov. Parameter-sweeping techniques for temporal dynamics of neuronal systems: Case study of Hindmarsh-Rose model. *Journal of mathematical neuroscience*, 1:6, 2011.

[4] M. J. Bayarri, J. O. Berger, J. Cafeo, G. Garcia-Donato, F. Liu, J. Palomo, R. J. Parthasarathy, R. Paulo, J. Sacks, and D. Walsh. Computer model validation with functional output. *The Annals of Statistics*, 35(5):1874–1906, 2007.

[5] Sourabh Bhattacharya. A simulation approach to Bayesian emulation of complex dynamic computer models. *Bayesian Analysis*, 2(4):783–815, 2007.

[6] Salomon Bochner. *Lectures on Fourier Integrals*. Princeton University Press, 1959.

[7] S.L. Brunton and J.N. Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019.
[8] Erika Camacho, Richard Rand, and Howard Howland. Dynamics of two van der Pol oscillators coupled via a bath. *International Journal of Solids and Structures*, 41(8):2133–2143, 2004.

[9] Rob Carnell. *lhs: Latin Hypercube Samples*, 2020. R package version 1.1.1.

[10] A. Castelletti, S. Galelli, M. Ratto, R. Soncini-Sessa, and P.C. Young. A general framework for dynamic emulation modelling in environmental problems. *Environmental Modelling & Software*, 34:5 – 18, 2012.

[11] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems 31*, pages 6571–6583. 2018.

[12] Victoria C. P. Chen, Kwok-Leung Tsui, Russell R. Barton, and Martin Meckesheimer. A review on design, modeling and applications of computer experiments. *IIE Transactions*, 38(4):273–291, 2006.

[13] S. Conti, J. P. Gosling, J. E. Oakley, and A. O’Hagan. Gaussian process emulation of dynamic computer codes. *Biometrika*, 96:663–676, 2009.

[14] Steo Conti and Anthony O’Hagan. Bayesian emulation of complex multi-output and dynamic computer models. *Journal of Statistical Planning and Inference*, 140(3):640–651, 2010.

[15] Nathalie Corson and Moulay Aziz-Alaoui. Asymptotic dynamics of Hindmarsh-Rose neuronal system. *Dynamics of Continuous, Discrete and Impulsive Systemes, Series B: Applications and Algorithms*, (16):535–549, 2009.

[16] P. Cvitanović, R. Artuso, R. Mainieri, G. Tanner, and G. Vattay. *Chaos: Classical and Quantum*. Niels Bohr Institute, Copenhagen, 2016.

[17] F. Dernoncourt, K. Veeramachaneni, and U. M. O’Reilly. Gaussian process-based feature selection for wavelet parameters: Predicting acute hypotensive episodes from physiological signals. In *2015 IEEE 28th International Symposium on Computer-Based Medical Systems*, pages 145–150, 2015.

[18] Angela M. dos Santos, Sergio R. Lopes, and R.L.Ricardo L. Viana. Rhythm synchronization and chaotic modulation of coupled van der Pol oscillators in a model for the heartbeat. *Physica A: Statistical Mechanics and its Applications*, 338(3):335–355, 2004.

[19] Kai-Tai Fang, Dennis K. J. Lin, Peter Winker, and Yong Zhang. Uniform design: Theory and application. *Technometrics*, 42(3):237–248, 2000.
[20] Thomas E. Fricker, Jeremy E. Oakley, and Nathan M. Urban. Multivariate Gaussian process emulators with nonseparable covariance structures. Technometrics, 55(1):47–56, 2013.

[21] Marc Goodfellow, Kaspar Schindler, and Gerold Baier. Self-organised transients in a neural mass model of epileptogenic tissue dynamics. NeuroImage, 59(3):2644–2660, 2012.

[22] José Miguel Hernández-Lobato, Matthew W. Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In Proceedings of the 27th International Conference on Neural Information Processing Systems, NIPS’14, pages 918–926. MIT Press, 2014.

[23] Dave Higdon, James Gattiker, Brian Williams, and Maria Rightley. Computer model calibration using high-dimensional output. Journal of the American Statistical Association, 103(482):570–583, 2008.

[24] J. L. Hindmarsh and R. M. Rose. A model of neuronal bursting using three coupled first order differential equations. Proceedings of the Royal Society of London, 221(1222):87–102, 1984.

[25] Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. Kernel methods in machine learning. Annals of Statistics, 36(3):1171–1220, 2008.

[26] Gaofeng Jia and Alexandros A. Taflanidis. Kriging metamodeling for approximation of high-dimensional wave and surge responses in real-time storm/hurricane risk assessment. Computer Methods in Applied Mechanics and Engineering, 261-262:24–38, 2013.

[27] Marc C. Kennedy and Anthony O’Hagan. Bayesian calibration of computer models. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 63(3):425–464, 2001.

[28] Rebecca Killick and Idris Eckley. changepoint: An R Package for Changepoint Analysis. Journal of Statistical Software, 58(3):1–19, 2014.

[29] Tim Kittel, Jobst Heitzig, Kevin Webster, and Jürgen Kurths. Timing of transients: quantifying reaching times and transient behavior in complex systems. New Journal of Physics, 19(8):083005, 2017.

[30] Georg Lindergen. Lectures on stationary stochastic processes. Lund University Press, 2006.

[31] Edward N. Lorenz. Deterministic nonperiodic flow. Journal of Atmospheric Sciences, 20:130–148, 1963.
[32] David Machac, Peter Reichert, and Carlo Albert. Emulation of dynamic simulators with application to hydrology. *Journal of Computational Physics*, 313:352–366, 2016.

[33] Colleen C. Mitchell and David G. Schaeffer. A two-current model for the dynamics of cardiac membrane. *Bulletin of Mathematical Biology, 65*(5):767–793, 2003.

[34] Hossein Mohammadi, Peter Challenor, and Marc Goodfellow. Emulating dynamic non-linear simulators using Gaussian processes. *Computational Statistics & Data Analysis*, 139:178–196, 2019.

[35] Edgar E. Peters. A chaotic attractor for the s&p 500. *Financial Analysts Journal*, 47(2):55–62, 1991.

[36] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, pages 1177–1184. 2008.

[37] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[38] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning (adaptive computation and machine learning)*. The MIT Press, 2005.

[39] Jerome Sacks, William J. Welch, Toby J. Mitchell, and Henry P. Wynn. Design and analysis of computer experiments. *Statistical Science*, 4(4):409–423, 1989.

[40] T. J. Santner, Williams B., and Notz W. *The design and analysis of computer experiments*. Springer-Verlag, 2003.

[41] Bernhard Scholkopf and Alexander J. Smola. *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT Press, Cambridge, MA, USA, 2001.

[42] Karline Soetaert, Thomas Petzoldt, and R. Woodrow Setzer. Solving differential equations in R: Package deSolve. *Journal of Statistical Software*, 33(9):1–25, 2010.

[43] S.H. Strogatz. *Nonlinear dynamics and chaos*. Studies in nonlinearity. Sarat Book House, 2007.

[44] Mark van der Wilk. *Sparse Gaussian process approximations and applications*. PhD thesis, University of Cambridge, 2019.
[45] Marcelo Viana. What’s new on lorenz strange attractors? *The Mathematical Intelligencer*, 22:6–19, 2000.

[46] Ziyu Wang. *Practical and theoretical advances in Bayesian optimization*. PhD thesis, University of Oxford, 2016.

[47] Arthur T. Winfree. Biological rhythms and the behavior of populations of coupled oscillators. *Journal of Theoretical Biology*, 16(1):15–42, 1967.

[48] Kirsten Zickfeld, Slawig Thomas, and Stefan Rahmstorf. A low-order model for the response of the atlantic thermohaline circulation to climate change. *Ocean Dynamics*, 54:8–26, 2004.