9-2021

A Surrogate-Based Approach to Nonlinear, non-Gaussian Joint State-Parameter Data Assimilation

John Maclean
University of Adelaide

Elaine T. Spiller
Marquette University, elaine.spiller@marquette.edu

Foundations of Data Science, Vol. 3, No. 3 (September 2021): 589-614. DOI.

A Surrogate-Based Approach to Nonlinear, non-Gaussian Joint State-Parameter Data Assimilation

John Maclean
School of Mathematical Sciences, University of Adelaide, SA 5005, Australia
Elaine Spiller
Department of Mathematical and Statistical Sciences, Marquette University, Milwaukee, WI

Abstract

Many recent advances in sequential assimilation of data into nonlinear high-dimensional models are modifications to particle filters which employ efficient searches of a high-dimensional state space. In this work, we present a complementary strategy that combines statistical emulators and particle filters. The emulators are used to learn and offer a computationally cheap approximation to the forward dynamic mapping. This emulator-particle filter (Emu-PF) approach requires a modest number

of forward-model runs, but yields well-resolved posterior distributions even in non-Gaussian cases. We explore several modifications to the Emu-PF that utilize mechanisms for dimension reduction to efficiently fit the statistical emulator, and present a series of simulation experiments on an atypical Lorenz-96 system to demonstrate their performance. We conclude with a discussion on how the Emu-PF can be paired with modern particle filtering algorithms.

Keywords

Data assimilation, uncertainty quantification, data science, statistical surrogates, parameter estimation.

1. Introduction/motivation

Data assimilation (DA) – the process of updating models with data to give state estimates and forecasts complete with attendant uncertainties – has progressed tremendously over the last three decades [9,15,27,5]. Sequential data assimilation techniques, those that update current state estimates and forecasts on the fly as data becomes available, fall into two general categories: Kalman-type filters (KF) and particle-type filters (PF). There are two dominant challenges in sequential DA, namely systems with high-dimensional state spaces and strong non linearity/non Gaussianity. Typically ensemble-based KF techniques, that use relatively few model runs, can address the former while particle-based techniques that require many model runs can address the latter.

Over roughly the same time frame, the field of statistical surrogates of complex computer models emerged [28,35]. Statistical surrogates provide computationally efficient approximations to nonlinear input/output mappings of a computer model; these statistical surrogates are typically based on a modest number of training runs. Further, statistical surrogates offer built-in uncertainty estimates for utilizing the approximation. In this work we develop particle filters that employ statistical surrogates to approximate mappings of system dynamics.

Often dynamic model forecasts are the computational bottleneck in sequential data assimilation. Our approach employs Gaussian process emulators (GPs) to learn the mapping from state and/or parameter values at one observation instance to the next. This mapping provides an effective interpolation between model forecasts and makes available slews of additional approximate model forecasts with negligible additional computational burden. Thus GP emulators are natural to pair with "sample hungry" DA techniques like particle filters. As such, we can produce finely-sampled non-Gaussian posterior estimates with a modest number of model runs typical of ensemble KF techniques. GP enhanced PFs are not an alternative to the assortment of recent innovative PF methodologies, rather this "GP emulator + PF strategy" can naturally be adapted to work with and bolster leading-edge PF algorithms. We demonstrate such a case by combining the 'Optimal Proposal' PF [8] with the GP emulator in section 4.6.

Several papers amount to a recent flurry of activity combining modeling learning and data assimilation, each approach with advantages and drawbacks. [6] combines statistical emulation and data assimilation to aid in model calibration, effectively a smoothing problem. This approach is quite similar to [1], but cleverly utilizes an ensemble Kalman filter (EnKF) sequentially between observations to choose a good design (e.g. a well-chosen set of training runs) to fit the statistical emulator. That

emulator then replaces forward model evaluations in MCMC evaluations in the calibration problem. [3] sets up methodology for combining Neural networks (NN) to learn an approximation to the dynamic forward model from noisy observations in a data assimilation framework. Effectively they construct a posterior distribution as one would in a sequential DA problem and use expectation maximization algorithm (EM) to compute mode posterior estimates of the NN parameters. This approach is quite appealing as it does not rely on physics-based model for forward propagation, yet simulation experiments in [4] indicate the need for a significant amount of data to train the NNs sufficiently. [11] is similar in spirit, but uses a random feature map instead of NNs and they combine their "physics-agnostic" forward model with an EnKF. Like [3,4], this methodology requires a significant amount of training data.

The core idea of this paper is that interpolation between model forecasts, thought of as functions of the parameter and/or previous state values at a fixed time, may be used to produce additional forecasts, and thus provide a cheap means to improve PF performance. If the state is interpreted as a function of the parameter estimates, then the interpolation exploits smoothness of the state with respect to parameter values, that is not used (nor required) in the usual formulations of the PF. Consider the following pedagogical example in terms of parameter dependence (dependence on previous state values or both is similar in spirit.) Suppose we have some computational model that provides forecasts of a single state variable, and which depends on a single parameter. Notionally this model is expensive to run, and we sample the model forecast at eight parameter values, as in Figure 1a. Provided the sampling is space filling in parameter space, one might be able to predict the state values at other choices of the parameters, as in Figure 1b. Maintaining a space-filling design – that is, not just relying on training points for the emulator near the bulk of the mass of the distribution – is key. Note that low probability samples are not used directly in the PF, but are used instead to construct a fast approximation to the mapping. Using a statistical surrogate instead of a deterministic interpolant, we can further capture the *uncertainty* in the state-parameter dependence at parameter values that have not been sampled, as in Figure 1c. These interpolating schemes allow for the output of the computational model to be estimated at many more potential sampling points than the initial eight. It is worth noting that access to more samples through the surrogate does not imply anything about the weight or likelihood of those samples. If these fine samples are used as "model forecasts" in a PF, then the effect is to obtain a dense estimate of the posterior using relatively few model forecasts.

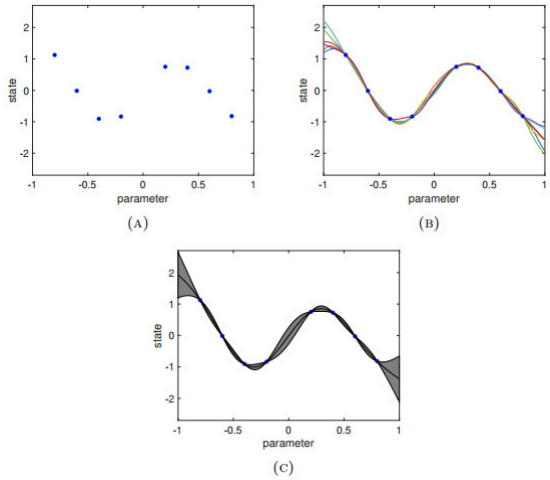


Figure 1. Schematic for state dependence on parameters: we plot the state at eight different samples (1a), then apply a variety of interpolating schemes (1b) and lastly a statistical surrogate (1c). The shaded region in the rightmost plot shows one standard deviation in uncertainty. The second and third plot allow for the state to be estimated at a variety of parameter values

The performance of this method relies on an efficient implementation of the simple interpolate-and-sample concept above. In particular the parameter values at which the model is evaluated should not all be fixed, but updated at each observation time, and the interpolating method should be a statistical surrogate that captures uncertainty. These foundational concepts from Data Assimilation and Uncertainty Quantification are introduced in Section 2. Our novel surrogate DA scheme is described in Section 3, and numerous visualisations of the internal mechanisms and error statistics of the new scheme are contained in Section 4.

2. Background

2.1. Sequential data assimilation

Let us begin by reviewing the setup for sequential data assimilation and two "standard" techniques: particle filters (PF) and ensemble Kalman filters (EnKF). In these approaches data or observations of a system are assimilated into a model describing the dynamics of the underlying system to offer an estimate of the state, and of parameters of interest along with attendant uncertainties. This is often achieved by employing Bayes theorem, $p(\mathbf{x}, \theta \mid \mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{x},\theta)p(\mathbf{x},\theta)}{p(\mathbf{y})}$where $\mathbf{x} \in \mathbb{R}^n$ is the state variable, $\mathbf{y} \in \mathbb{R}^m$ is the data, and $\theta \in \mathbb{R}^p$ are parameters.

In the sequential case where data are available as a time series, we will follow the notation of Doucet *et. al.* [8]. For observations available at times $\mathbf{t} = \{t_j, t_{j+1}, \dots, t_k\}$ we use the shorthand $\mathbf{y}_{j:k} = \{\mathbf{y}_j, \dots, \mathbf{y}_k\}$ and likewise for state variables and parameters at times $\mathbf{t}$, $\mathbf{x}_{j:k} = \{\mathbf{x}_j, \dots, \mathbf{x}_k\}$, $\theta_{j:k} = \{\theta_j, \dots, \theta_k\}$. Ultimately for sequential state-parameter data assimilation, we are interested in describing the posterior distribution

$$p(\mathbf{x}_{0:k}, \theta_{0:k} \mid \mathbf{y}_{1:k}) = \frac{p(\mathbf{y}_{1:k} \mid \mathbf{x}_{0:k}, \theta_{0:k})p(\mathbf{x}_{0:k}, \theta_{0:k})}{p(\mathbf{y}_{1:k})},$$

where the marginal distribution in the denominator is given by $p(\mathbf{y}_{1:k}) = \mathrm{E}_{p(\mathbf{x}_{0:k},\theta)}\left[p(\mathbf{y}_{1:k} \mid \mathbf{x}_{0:k}, \theta_{0:k})\right]$. All of these distributions are updated sequentially as data become available. For both PFs and EnKFs, we can think of the representation of the prior and posterior probability density functions (pdfs) as a collection of N "particles", e.g. state variable and parameter estimates with weights, $\{\mathbf{x}_j^i, \theta_j^i, w_j^i\}_{i=1}^N$. In both cases, the particle states are advanced via system dynamics, e.g. $\dot{\mathbf{x}} = \mathcal{M}(\mathbf{x}, \theta)$, from time t_j to time t_{j+1}, according to some map. For deterministic simulations we write

$$\mathbf{x}_j^i = \varphi\left(\mathbf{x}_{j-1}^i, \theta_{j-1}^i\right) = \mathbf{x}_{j-1}^i + \int_{t_{j-1}}^{t_j} \mathcal{M}\left(\mathbf{x}^i, \theta^i\right) dt. \tag{1}$$

For a stochatic simulation we use the same notation, $\mathbf{x}_j^i = \varphi(\mathbf{x}_{j-1}^i, \theta_{j-1}^i)$, but φ now refers to a *realization* of the stochastic dynamics. }Obtaining state estimates is typically the most computationally intensive part of the particle filter, particularly in the diverse applications in which the dynamical system $\mathcal{M}$ is high-dimensional.

Particle filters.
For generic particle filters, the particle representations of probability density functions (pdfs) are updated by adjusting the weights via the likelihood as current data are incorporated while state values remain unchanged. Often sequential importance resampling (SIR) [17] is employed to overcome inherent filter degeneracy. The idea behind SIR particle filters is to use the posterior distribution from one time step as the prior distribution for the next (along with the state updated by the forward dynamics in eq. (1)) as

$$p(\mathbf{x}_j, \theta_j \mid \mathbf{x}_{j-1}, \theta_{j-1}, \mathbf{y}_{1:j}) = \frac{p(\mathbf{y}_j \mid \mathbf{x}_j) p(\mathbf{x}_j, \theta_j \mid \mathbf{x}_{j-1}, \theta_{j-1}, \mathbf{y}_{1:j-1})}{p(\mathbf{y}_j)}. \tag{2}$$

Each particle is a notionally independent guess $(\mathbf{x}_j^i, \theta_j^i)$ for the states and parameters, where i runs from 1 to the number of particles N_e. A simple implementation of this approach involves updating both particles and weights sequentially in time.

The state component of the particles is updated from time $j-1$ to time j by running the model eq. (1). The parameter component of each particle may formally be assigned the model

$$\theta_j^i = \theta_{j-1}^i$$

associated with the trivial dynamics $\theta = 0$; however, to allow parameter estimates to vary over time, in practice one would employ a non-trivial *parameter model*. We employ the parameter model from [16,e.g.]: at each observation time, parameter estimates are shrunk slightly towards their mean and then some noise is added. Shrinking the parameter estimates mitigates over-dispersion from the repeated application of noise. The parameter model is

$$\theta_j^i = \alpha \theta_{j-1}^i + \frac{1-\alpha}{N_e} \sum_{i=1}^{N_e} \theta_{j-1}^i + \beta \xi_j^i, \tag{3}$$

where $\xi_j^i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ are iid. The parameters α and β determine the strength of the shrinking and noise effects, respectively. Following advice in [16] we generally employ $\alpha = 0.99$, and choose small $\beta = 0.01$.

Once the particles are updated to time j, the weights are updated by computing the likelihood $p(y|x)$ and normalising, with the j^{th} observation yielding

$$w_j^i = \frac{p(\mathbf{y}_j \mid \mathbf{x}_j^i)}{\sum_{i=1}^{N} p(\mathbf{y}_j \mid \mathbf{x}_j^i)} w_{j-1}^i \tag{4}$$

and the posterior approximation

$$p(\mathbf{x}_j, \theta_j \mid \mathbf{y}_{1:j}) \approx \sum_{i=1}^{N} w_j^i \delta(\mathbf{x} - \mathbf{x}_j^i)\delta(\theta - \theta_j^i).$$

Generally for particle filters, resampling will need to be employed when the effective number of particles, $N_{eff} \approx \sum_{i=1}^{N} 1/(w_j^i)^2$, falls beneath some user defined threshold—typically $5 - 10\%$ of N [8]. Note, unless N_{eff} is large, this resulting discrete representation of the posterior is inherently coarse.

Perturbed-obs EnKF.

The ensemble Kalman filter with perturbed observations (summarized here following the work by Evensen [10]) is a sequential data assimilation technique that evolves an ensemble of model states through time and performs Kalman filter style updates as new observations are incorporated.

Given an ensemble of N_e model states at time t_{j-1}, each ensemble member is evolved according to eq. (1). This forecast ensemble is used to generate a Gaussian estimate of the prior distribution at time t_j. We denote the forecast ensemble as $\{\mathbf{x}_{j,f}^i | i = 1, \dots, N_e\}$. The forecast ensemble sample mean $\overline{\mathbf{x}}_{j,f}$ and sample covariance $\mathbf{P}_{j,f}$ can be estimated as follows:

$$\overline{\mathbf{x}}_{j,f} = \frac{1}{N_e}\sum_{i=1}^{N_e} \mathbf{x}_{j,f}^i \tag{5}$$

$$\mathbf{P}_{j,f} = \frac{1}{N_e - 1}\sum_{i=1}^{N_e} (\mathbf{x}_{j,f}^i - \overline{\mathbf{x}}_{j,f})(\mathbf{x}_{j,f}^i - \overline{\mathbf{x}}_{j,f})^T. \tag{6}$$

Observations are assumed to have the form $\mathbf{Y}_j = \mathbf{H}\mathbf{x}_j + \eta_j$, where HH is an observation matrix (typically a linearized observation operator) and observation errors η_j are taken to be iid Gaussian random variables with mean 0 and known covariance $\mathbf{R}$, i.e. $\eta_j \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$. We create an ensemble of N_e perturbed observations with mean equal to $\mathbf{Y}_j$ and covariance $\mathbf{R}$ according to $\mathbf{Y}_j^i = \mathbf{Y}_j + \epsilon_j^i$ where $\epsilon_j^i \sim \mathcal{N}(0, \mathbf{R})$. The covariance of the ensemble of perturbed observations is given by

$$\mathbf{R}_j^e = \frac{1}{N_e - 1}\sum_{i=1}^{N_e} \epsilon_j^i \epsilon_j^{i^T}. \tag{7}$$

The ensemble members are then updated according to

$$\mathbf{x}_{j,a}^i = \mathbf{x}_{j,f}^i + \mathbf{P}_{j,f}\mathbf{H}^T(\mathbf{H}\mathbf{P}_{j,f}\mathbf{H}^T + \mathbf{R}_j^e)^{-1}(\mathbf{Y}_j^i - \mathbf{H}\mathbf{x}_{j,f}^i) \tag{8}$$

and the sample analysis mean and analysis covariance can be calculated as above yielding

$$\overline{\mathbf{x}}_{j,a} = \frac{1}{N_e}\sum_{i=1}^{N_e}\mathbf{x}_{j,a}^{i}, \text{and} \mathbf{P}_{\mathrm{j,a}} = \frac{1}{\mathrm{N_e}-1}\sum_{\mathrm{i=1}}^{\mathrm{N_e}}(\mathbf{x}_{\mathrm{j,a}}^{\mathrm{i}} - \overline{\mathbf{x}}_{\mathrm{j,a}})(\mathbf{x}_{\mathrm{j,a}}^{\mathrm{i}} - \overline{\mathbf{x}}_{\mathrm{j,a}})^{\mathrm{T}}. \quad (9)$$

The analysis ensemble is used to generate a Gaussian approximation of the posterior distribution at time t_j. The analysis ensemble $\{\mathbf{x}_{j,a}^{i}\}$ is then evolved to the next observation time by eq. (1) and used as the forecast ensemble for the next assimilation step.

2.2. Gaussian process emulators

The key approach in this work is, at time t_j, to learn about the mapping from an "input space" (parameter space and/or state space at time t_{j-1}) to an "output space" (state space) through a limited number of particle/ensemble samples. (Note, the weights of the particles do not inform us about this mapping directly.) To this end, we will employ a weakly stationary Gaussian process (GP) to model such an unknown relationship, $\mathbf{x}_j \approx \hat{f}_j(\mathbf{x}_{j-1}, \theta)$ or $\mathbf{x}_j \approx \hat{f}_j(\theta)$. In the statistical computer models community, such modeling is typically referred to as statistical surrogates or GP emulators – effectively statistical models of physical models. [30,26,35] provide excellent and broad overviews of this approach, but for the unfamiliar reader, we summarize the salient points here. It is worth emphasizing that "Gaussian" in GP emulators refers to the class of non-parametric random functions used for interpolation, and does not impose Normal approximations on either the input or output random variables. To frame it another way, the values the particles take on at times t_{j-1} and t_j inform the mapping to be learned in equation 1, but the weights of the particles do not inform this mapping.

Consider n_D *training* or *design* input values, $\mathbf{q}^D = \{\mathbf{q}_1, \dots, \mathbf{q}_{n_D}\}$, with each $\mathbf{q}_k \in \mathbb{R}^r$, and a scalar output y_k (e.g. "output" may be one of the state variable values at time t_j, $y_k = x_j^k$) at each of these n_D inputs, $\mathbf{y}^D = (y_1, \dots, y_{n_D})^T$. We can model $\hat{y} \sim \mathrm{MVN}(m(\mathbf{q}^D), \Sigma)$, a multivariate normal with $m(\cdot)$ a known mean trend and $\Sigma = \sigma^2\hat{\mathbf{R}}$, with variance σ^2. Here the correlation matrix $\hat{\mathbf{R}}$ is computed by evaluating a chosen correlation function $c(\cdot,\cdot)$, e.g. each element is given by $(\hat{\mathbf{R}})_{ij} = c(\mathbf{q}_i, \mathbf{q}_j)$. A Gaussian process emulator provides a prediction $\hat{y}(\mathbf{q}^*)$ at an untried value of the input space $\mathbf{q}^*$ as

$$\begin{aligned}\hat{y}(\mathbf{q}^*) = &\quad h(\mathbf{q}^*)\beta + \mathbf{r}^T(\mathbf{q}^*)\hat{\mathbf{R}}^{-1}(\mathbf{y}^D - h(\mathbf{q}^D)\beta) + \delta \\ = &\quad \hat{f}(\mathbf{q}^*).\end{aligned} \quad (10)$$

where $\mathbf{r}(\mathbf{q}^*) = (c(\mathbf{q}^*, \mathbf{q}_1), \dots, c(\mathbf{q}^*, \mathbf{q}_{n_D}))^T$. In other words, this gives the mean value of a Gaussian process at input $\mathbf{q}^*$, where the process is conditioned to take on values of $\mathbf{q}^D$ at inputs $\mathbf{q}^D$ if the uncorrelated noise term, δ, is zero. Here h is a set of basis functions (typically taken to be constant or linear), so $m(\mathbf{q}) = h(\mathbf{q})\beta$ gives the overall trend based on the data, and the coefficient(s) are given by

$$\beta = (h^T(\mathbf{q}^D)\hat{\mathbf{R}}^{-1}h(\mathbf{q}^D))^{-1}h^T(\mathbf{q}^D)\hat{\mathbf{R}}^{-1}\mathbf{y}^D.$$

In these formulae, $\hat{\mathbf{R}}$ is the $n_D \times n_D$ correlation matrix of the input design; often a power exponential or Matérn correlation kernel is assumed and "fitting" an emulator amounts to finding the trend coefficients and correlation length scales that best represent the design pairs $\{\mathbf{q}^D, \mathbf{y}^D\}$. We can also gain a sense of uncertainty induced by using the GP instead of the computer model simulation directly at $\mathbf{q}^*$ by considering the standard prediction error

$$s^2(\mathbf{q}^*) = \sigma^2\left(1 - \mathbf{r}^T\hat{\mathbf{R}}^{-1}\mathbf{r} + \frac{(1 - \mathbf{1}^T\hat{\mathbf{R}}^{-1}\mathbf{r})^2}{\mathbf{1}^T\hat{\mathbf{R}}^{-1}\mathbf{1}}\right), \quad (11)$$

where 1 is an n-vector of ones and σ^2 is the variance scaling of the process and found during the "fitting" of the GP. Implementations of GP emulators are available: in Matlab one can use the function fitrgp()[1] or the Robustgasp() package [13] (also available in R).

[1]Introduced in version R2015b, see https://mathworks.com/help/stats/fitrgp.html

In the computer model community, space filling designs—typically Latin Hypercubes (LHC)—are the standard approach for training emulators [30]. LHC designs spread out samples and ensure that the maximum distance between each neighboring pair of design points is (approximately) minimized [14]. In using emulators in conjunction with DA methods, it is important to keep in mind that we are *not* assigning any probability to events in the design used for the construction of emulators. That said, we are interested in the GP being a "good" mapping, e.g. the GP having small sample variance in regions of design space where the prior distribution has significant mass. To this end for constructing emulators, we chose some design points to be space filling and some to target the mass of the prior. We describe details of the our proposed emulator design for the Emu-PF in section 4.1.

The Parallel Partial Emulator (PPE) generalizes the standard emulator construction presented in eq. (10) for scalar outputs, to an emulator for vector-valued outputs [12]. Consider then a set of n_D model design inputs and n-dimensional responses $\{\mathbf{q}^D, \boldsymbol{Y}^D\}$. $\boldsymbol{Y}^D$ is now an $n_D \times n$ matrix. PPE allows each output component to have a unique mean $m_j(\mathbf{q}) = h(\mathbf{q})\psi_j$ and variance $j = 1, \dots, n$, but assumes a shared correlation structure and correlation parameters among all locations. Equations for predictive mean and standard error are nearly identical to eq. (10), but are n −dimensional. We mention that the means and variances of the individual Gaussian processes inherit some measure of (spatial) correlation that is present in the physical system, even though no explicit assumption is made about spatial relationships.

2.3. A pedagogical example

Before diving into details of the algorithm and application, we will again focus on the simple pedagogical example of the mapping from parameter space to state space illustrated in Figure 1. Now let us consider a bi-modal prior pdf on the parameter and walk through one PF update step given an observation in state space. We visualize this setup in the left panel of Figure 2. Although we only have eight points to learn it, the GP allows us to sample the prior as much as we like and push those samples

through the emulator-based mapping in order to compare them to the observation. For demonstration, we consider three cases in this update step: a large sample (10^3) using the emulator mapping (EmuPF), the same large sample using the true mapping (for comparison), and a case with eight samples from the prior that is analogous to a standard sample-limited PF implementation. The resulting posterior histograms are displayed in the right panel of Figure 2. (Note the large sample cases are normalized and the small sample posterior is visualized as a stem plot with the height of the stems reflecting relative weights.) From this figure we can see that error introduced by using the emulator-based mapping instead of the true mapping is small – the histograms are nearly identical – and does not impose Gaussianity on the posterior pdf.

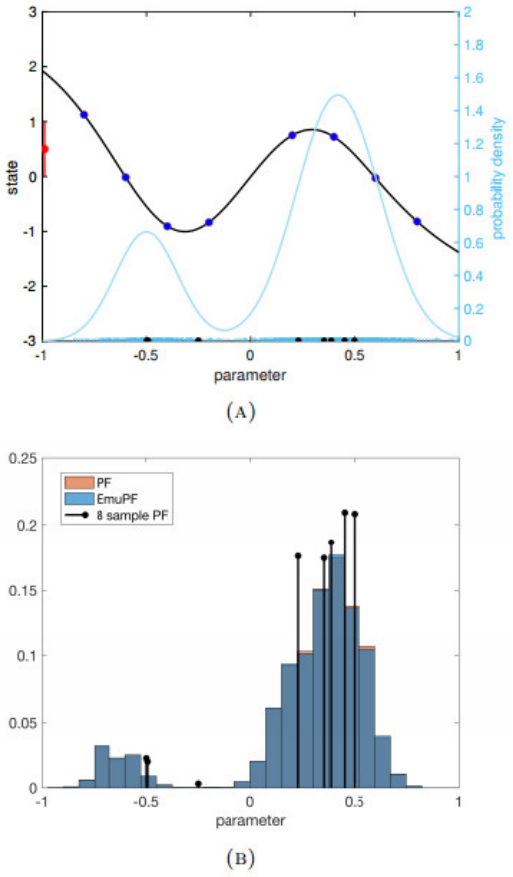


Figure 2. Here we demonstrate how the GP example mapping from parameter space to state space (as in Figure 1) can be used in a particle filter update step. (A) The same GP mapping from parameter to state space is plotted (black line) along with the design points (blue dots along black line) used to fit that mapping and an observation (red dot and line) in state space along the left axis. A bi-modal prior distribution is plotted (light blue) along with samples from that distribution (10^3 in light blue and 8 in black) along the horizontal axis. (B) Stem plots of the eight-sample PF posterior along with 10^3-sample normalized posterior histogram of parameters taking into account the likelihood of GP mapped prior samples given the observation in (A). Plotted behind the emuPF histogram is the equivalent (and nearly identical) histogram using the true mapping instead of the GP mapping for each of the 10^3 samples.

The error introduced by utilizing emulator mapping in the Particle Filter shows up in the posterior sample weights: consider assimilating an observation of a single variable. Define the emulator mapping error as $e = h - \tilde{h}$ where h is the true mapping and $\tilde{h}$ is the emulator-based mapping. Assuming an

observation, y, is normally distributed with mean h and variance σ^2, then we can relate the weight calculation of the particle filter w^{PF} to the weight of the EmuPF, w^{EmuPF}, by

$$-\log\,(w^{PF})^{1/2} = \frac{y-h}{\sqrt{2}\sigma} = \frac{y-\tilde{h}-e}{\sqrt{2}\sigma} = \frac{y-\tilde{h}}{\sqrt{2}\sigma} - \frac{e}{\sqrt{2}\sigma} \tag{12}$$

$$-\log\,(w^{PF}) = (\frac{y-\tilde{h}}{\sqrt{2}\sigma})^2 - \frac{2e(y-\tilde{h})}{2\sigma^2} + (\frac{e}{\sqrt{2}\sigma})^2 \tag{13}$$

$$= (\frac{y-\tilde{h}}{\sqrt{2}\sigma})^2 - \frac{2e(y-h+e)}{2\sigma^2} + (\frac{e}{\sqrt{2}\sigma})^2 \tag{14}$$

$$= \underbrace{(\frac{y-\tilde{h}}{\sqrt{2}\sigma})^2}_{-\log\,(\breve{w}^{EmuPF})} - \underbrace{(\frac{e(y-h)}{\sigma^2} + \frac{e^2}{2\sigma^2})}_{\log\,(\breve{w}^{ERR})}$$

The error in the weighting, w^{ERR} depends both on the mapping error, e, and on the difference between observation and mapped prior sample. Note that e is not constant, but its magnitude is estimated by the predictive standard deviation of the GP, $s(\cdot)$ from Equation 11, for any input sample. As long as e is small relative to σ, the weighting error introduced by utilizing emulators – which is controllable by adding targeted training/design points in regions of input space with large predictive variance – will be close to one as will the ratio w^{PF}/w^{EmuPF}.

3. Methodology

This section constructs approximations of the Particle Filter that employ only a relatively small number of model runs. The model runs are used as design-response pairs in a Gaussian Process emulator; a large number of samples from the GP emulator are then treated as particles in a PF. Several algorithms are presented here, as the practical options for emulator design and response variables depend on the parameter and state dimension.

> The following section 3.1 introduces a naive but straightforward blending of the GP emulator and PF, which is then employed as a springboard to introduce multiple refinements.
>
> We employ subscripts for time indices and superscripts for particle indices, and employ bold font for vectors.

3.1. The Emulator Particle Filter: Emu-PF

We construct an emulator for the map eq. (1) from time $t_{j-1} \to t_j$. Then, we use the emulator output in a PF as if it were samples from the prior distribution in eq. (2).

At time t_{j-1} suppose we have evenly weighted parameter estimates and state estimates $\theta_{j-1}^i \in \mathbb{R}^p$ and $\mathbf{x}_{j-1}^i \in \mathbb{R}^n$, i from 1 to n_D. Suppose also we have a large ensemble of parameter estimates Θ_{j-1}^i and corresponding state estimates $\mathbf{X}_{j-1}^i$ with weights w_{j-1}^i, i from 1 to N_F. Then follow the following sequence:

Forecast: Employ the numerical model eq. (1),

$$\mathbf{x}_j^i = \varphi(\mathbf{x}_{j-1}^i, \theta_{j-1}^i),$$

Label the lth component of $\mathbf{x}_j^i$ by $\mathbf{x}_j^{i,l}$.,l. Set $\theta_j^i = \theta_{j-1}^i$ and $\Theta_j^i = \Theta_{j-1}^i$.

Emulate: For each l from 1 to the state dimension n,

E1. Set the emulator design variables (inputs) to be the state and parameter estimates at which we employed the model,

$$\mathbf{q}^D = \{\mathbf{x}_{j-1}^i, \theta_{j-1}^i\}_{i=1}^{n_D}.$$

E2. Set the response variables (outputs) to be the llth component of the state variable from the model output,

$$\mathbf{y}^D = \{x_j^{i,l}\}_{i=1}^{n_D},$$

and fit the emulator with the design-response pairs.

E3. Evaluate the emulator at each of the fine state and parameter values: set

$$\mathbf{q}^* = \{\mathbf{X}_{j-1}^i, \Theta_{j-1}^i\}_{i=1}^{n_D},$$

obtain $\hat{y}(\mathbf{q}^*) = \hat{f}(\mathbf{q}^*)$ from eq. (10), and save each scalar emulator output as the lth component of $\mathbf{X}_j^i$, $X_j^{i,l} = \{\hat{y}(\mathbf{q}^*)\}_i$, for each i from 1 to N_F.

Assimilate: Treat $\{\Theta_j^i, \mathbf{X}_j^i\}$ as samples from the prior and perform a Data Assimilation scheme in the high-dimensional N_F space. For example, in a Particle Filter, employ eq. (4) to judge the emulator outputs,

$$w_j^i = \frac{\exp\left(-\frac{1}{2}(\mathbf{y}_j - h(\mathbf{X}_j^i))^T \mathbf{R}^{-1}(\mathbf{y}_j - h(\mathbf{X}_j^i)\right)}{\sum_{k=1}^{N} \exp\left(-\frac{1}{2}(\mathbf{y}_j - h(\mathbf{X}_j^k))^T \mathbf{R}^{-1}(\mathbf{y}_j - h(\mathbf{X}_j^k)\right)},$$

assuming the observation errors are Gaussian with covariance $\mathbf{R}$, and the observation operator is $h()$. Calculate the effective sample size N_{eff}, defined below eq. (4), and resample $\{\Theta_j^i, \mathbf{X}_j^i\}$ if needed.

Subsample: Use a resampling algorithm to sample n_D times from the weighted pairs $\left(\{\Theta_j^i, \mathbf{X}_j^i\}, w_j^i\right)$.

In the above we use $\{.\}_{i=1}^N$ to indicate when an operation can be vectorized by concatenating together ensemble members as columns in a matrix.

Figure 3 shows a schematic for this class of surrogate DA methods. The key steps in this schematic are displayed for an Emu-PF (with implementation details delayed until section 4) in fig. 4. The long-time

error statistics for this Emu-PF are compared to Particle Filters employing n_D particles and N_F particles in fig. 5.

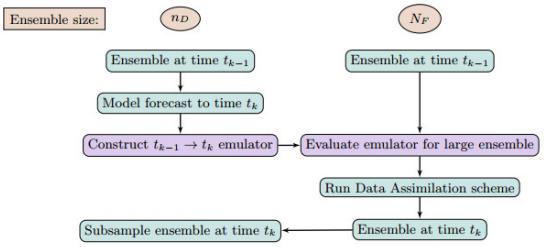
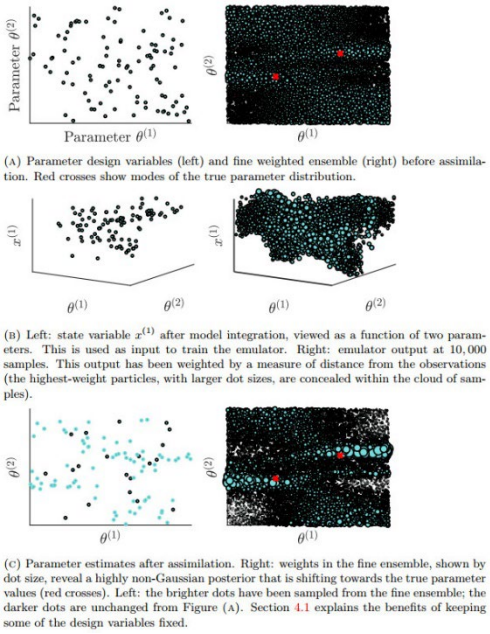


Figure 3. Overview of the novel synthesis of Gaussian process emulators with Data Assimilation methods



(A) Parameter design variables (left) and fine weighted ensemble (right) before assimilation. Red crosses show modes of the true parameter distribution.

(B) Left: state variable $x^{(1)}$ after model integration, viewed as a function of two parameters. This is used as input to train the emulator. Right: emulator output at 10,000 samples. This output has been weighted by a measure of distance from the observations (the highest-weight particles, with larger dot sizes, are concealed within the cloud of samples).

(C) Parameter estimates after assimilation. Right: weights in the fine ensemble, shown by dot size, reveal a highly non-Gaussian posterior that is shifting towards the true parameter values (red crosses). Left: the brighter dots have been sampled from the fine ensemble; the darker dots are unchanged from Figure (A). Section 4.1 explains the benefits of keeping some of the design variables fixed.

Figure 4. Visualisation of the internal Emu-PF mechanisms over one assimilation step. Left column shows components of dimension $n_D = 100$. Right column shows components of dimension $N_F = 10{,}000$. (a): parameter ensembles at time t_j. (b): distribution of one state variable as a function of parameters. (c): parameter ensembles at time t_{j+1}. Full details for this 8 state, 2 parameter experiment are given in section 4

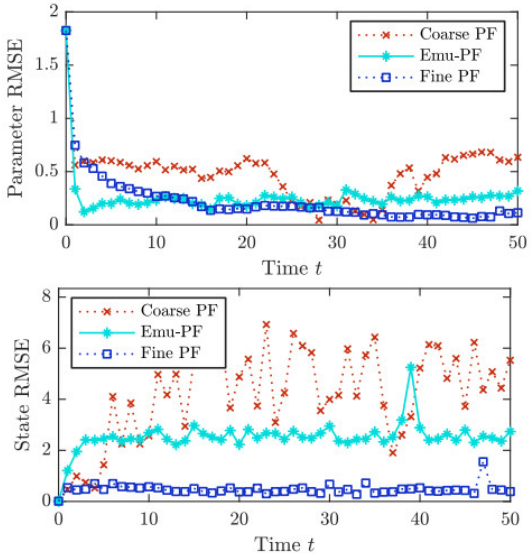


Figure 5. Long term error statistics for the implementation of Emu-PF from fig. 4, compared to: a "coarse" PF that employs $n_D = 100$ model runs (as in the Emu-PF), and a "fine" PF that employs $N_F = 10{,}000$ model runs, equal to the number of samples in the Emu-PF emulator. $n_D = 100$ of Emu-PF is markedly better than the coarse PF

Two shortcomings of GP emulators motivate improvements in the above algorithm. First, it is notoriously challenging to fit emulators with a high-dimensional input space. Yet the surrogate DA method employs high-dimensional inputs to the emulator, as the parameter and state vectors are combined and used as design variables. Some recent works [2,18] offer approaches for dimension reduction for statistical emulators that require either significant prior knowledge of the variability of the input space or a significant amount of data to characterize that variability well. For sequential DA, as a matter of course we have this prior knowledge available, but the flavor of appropriate dimension reduction will be problem specific. In particular, DA schemes that employ localization may favor a dimension reduction approach that is local as opposed to a global dimension reduction. We will explore variations of each. Secondly high-dimensional response variables are avoided by looping through the entire state vector, one dimension at a time; but this is a potentially slow and expensive procedure. There are multiple recent approaches to emulating high-dimensional output and we will explore a variation of our algorithm that utilizes one of those approaches, namely partial parallel emulation [12].

We now introduce practical variations of the Emu-PF. Each either reduces the dimension of the design variables or improves the efficiency of sampling from the emulator.

3.2. Variant: Include only some state values in the emulator input

This modification implements a straightforward localization for the emulator inputs. Modify the emulation step to include only state values near the response variable. For each l from 1 to the state dimension n,

Es1. Choose some integer Γ. Set the emulator design inputs to be the parameter estimates at which we employed the model, and a slice of the state inputs,

$$\mathbf{q}^D = \{(\theta_{j-1}^i, \mathbf{x}_{j-1}^{i,(l-\Gamma:l+\Gamma)})\}_{i=1}^{n_D}.$$

Es2. As item E2.

Es3. Evaluate the emulator at each of the fine parameter values and corresponding state estimates: set

$$\mathbf{q}^* = \{(\Theta_{j-1}^i, \mathbf{X}^{i,(l-\Gamma:l+\Gamma)})\}_{i=1}^{n_D},$$

otherwise as item E3.

In 2-d or 3-d space, instead choose a localization distance parameter $\Gamma \geq 0$ and include every grid point within radius Γ of $\mathbf{x}_j^{\prime,(l)}$ in the design input step item Es1.

One extremely simple implementation of this variation on the Emu-PF is to set $\Gamma = -1$; that is, to include no state variables at all in the emulation. This implementation is justified if the distribution $\mathbf{x}|\theta \approx g(\theta) +$ noise, for a smooth function g. Equivalently, the distribution $\mathbf{x}|\theta$ should be roughly unimodal. This condition is frequently satisfied in practice [21], and the resulting algorithm is fast but still readily capable of filtering nonGaussian marginal distributions for θ.

If this variant of the Emu-PF is employed, we refer to it by the value of Γ chosen; DA methods are benchmarked against the Emu-PF with $\Gamma = -1$ in figs. 6 to 8.

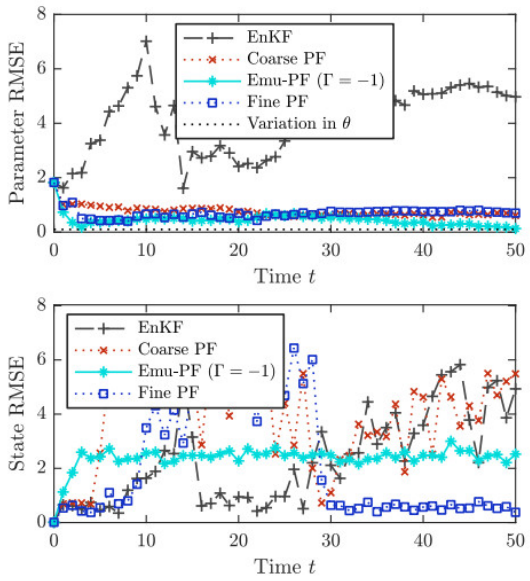


Figure 6. Error statistics for Experiment One, $m = 8$ observations at each observation time, of accuracy $\sigma_0 = 1$. In this (and every) plot, only every 20th data point is shown. For this mildly difficult filtering problem, we observe that the $\Gamma = -1$ implementation of section 3.2, that uses no state variables at all as emulator inputs, is stable and reasonably accurate

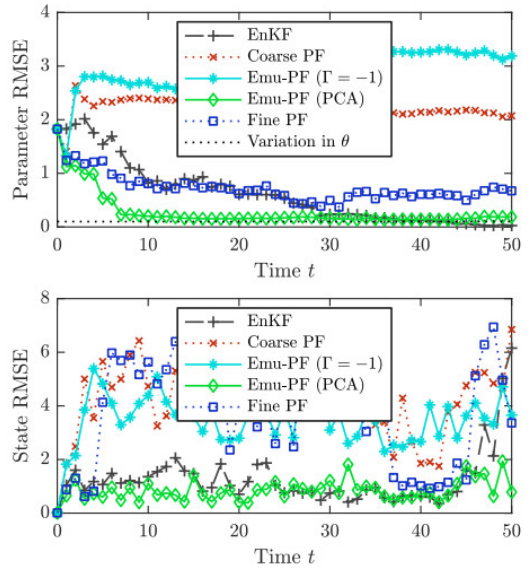


Figure 7. Error statistics for Experiment Two, $m = 2$ observations at each observation time, of accuracy $\sigma_0 = 1$. The $\Gamma = -1$ Emu-PF and fine PF both under-perform compared to their mean behaviour; the Emu-PF employing PCA is stable and accurate

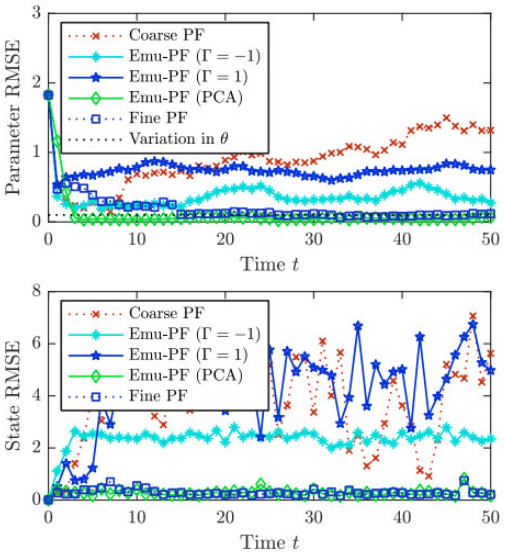


Figure 8. Error statistics for Experiment Three, $m = 4$ observations at each observation time, of accuracy $\sigma_0 = 0.5$. In this case the $\Gamma = 1$ Emu-PF performs only as well as the coarse PF. However the Emu-PF employing PCA is still competitive with the, much more expensive, fine PF

3.3. Variant: Compute emulator outputs in parallel with ppgasp

Use Partial Parallel Estimation (described in section 2.2) to compute all states at once.

Ep1.Set the emulator design inputs to be the parameter estimates at which we employed the model,

$$\mathbf{q}^D = \{\theta_{j-1}^i\}_{i=1}^{n_D}.$$

Ep2. Set the response variables to be the model output,

$$\mathbf{y}^D = \{\mathbf{x}_j^i\}_{i=1}^{n_D}.$$

Ep3. Evaluate the emulator at each of the fine parameter values: set

$$\mathbf{q}^* = \{\Theta_{j-1}^i\}_{i=1}^{n_D},$$

obtain $\hat{y}(\mathbf{q}^*) = \hat{f}(\mathbf{q}^*)$ from eq. (10), and save each column of emulator output as $\mathbf{X}_j^i = \{\hat{y}(\mathbf{q}^*)\}_i$ for each ii from 1 to N_F.

The above implementation avoids the for-loop present in sections 3.1 and 3.2, but as written requires $\Gamma = -1$ (no state variables as design inputs) discussed in section 3.2. We discuss simultaneous parallelization and localization in section 5. A more radically localized Emu-PF for state estimation is discussed in section 3.5.

3.4. Variant: Perform a global dimension reduction before using emulator inputs

Employ a data-based dimension reduction algorithm (e.g. PCA, DMD, diffusion maps, UMAP, ...) on the state variables going into the emulation mapping of $\mathbf{x}_{j+1} = \varphi(\mathbf{x}_j; \theta_j)$. This approach is not generically

used to emulate high dimensional parameter inputs because it's often unclear how to represent the variability of parameters, but in the sequential DA case there is an obvious candidate—the vector of state variables.

As a clear example, in the remainder of the section and in numerical examples we employ PCA. That is, we have an approximation from the fine sampled posterior at the j th time step, $(\{\Theta_j^i, \mathbf{X}_j^i\}, w_j^i)_{i=1}^{N_F}$. Let $X = X_{data} - \overline{X}_{data}\mathbf{1}_{N_F}$ be the $n \times N_F$ matrix where the ith column of X_{data} is $\mathbf{X}_j^i$, $\overline{X}_{data} = \frac{1}{N_F}\sum_{i=1}^{N_F}\mathbf{X}_j^i$, and $\mathbf{1}_{N_F}$ is a row vector consisting of N_F ones. Then $A = XX^T$ is a covariance matrix representative of the variance in X. A singular value decomposition of A produces $A = V\Lambda V^T$ where Λ is a unitary matrix containing ordered singular values, the columns of V contain the corresponding singular vectors, and $V^T = V^{-1}$ as A is symmetric. Truncate Λ and V to keep only the largest $r < n$ singular values; label the truncated matrices $\tilde{\Lambda}$, now $r \times r$, and $\tilde{V}$ now $J \times r$. Note $A \approx \tilde{V}\tilde{\Lambda}\tilde{V}^T$. Now let $Y = \tilde{V}^T X$. Effectively Y is a matrix of weights to multiply the principal components vectors (columns of $\tilde{V}$) to recover the original data X.

In Emu-PF schemes employing PCA, we use the weights Y as input variables for emulation in item E1. The response variables are unchanged in item E2, but when evaluating the emulator at fine samples in item E3 we replace $\mathbf{X}_{j-1}^i$ with $\tilde{V}^T\mathbf{X}_{j-1}^i$.

We discover a fast, flexible and powerful Emu-PF algorithm by combining global dimension reduction of inputs (by PCA in our experiments) and fast emulator outputs (with PPE, described in section 3.3); this algorithm is employed in Experiments Two and Four of section 4.

3.5. Variation: Localize the emulator by "slicing and stacking" the emulator inputs

This variation on the Emu-PF involves a radical rethinking of the emulator state inputs; for that reason we suppress parameter dependence and consider state estimation only. Assume that the physical law governing state evolution is the same for each component of the state vector; then a single model run, initialized at $\mathbf{x}_{j-1}^i$ and producing $\mathbf{x}_j^i \in \mathbb{R}^n$, provides nn samples of that physical law. The following algorithm exploits this rich data by configuring the emulator design inputs as $n \times n_D$ samples, rather than n_D samples.

We suppose that some localized slice of state variables at time $j-1$, within distance Γ of state variable l, is sufficient to predict the lth state variable at time j. The following procedure learns a $\mathbb{R}^{2\Gamma+1} \to \mathbb{R}$ map for the state update.

Er1. Choose some integer $\Gamma \geq 0$. The design inputs $\mathbf{q}^D$ are to be a $(2\Gamma+1) \times (n \times n_D)$ array, with the q-th row of that array given by

$$\{\mathbf{x}_{j-1}^{i,l-\Gamma:l+\Gamma}\}_{i=1}^{n_D},$$

where $i = \text{ceil}(q/n)$ and $l = \text{mod}(q, n)$..

Er2. Set the response variables $\boldsymbol{Y}^D$ to be the corresponding $n \times n_D$-vector of state variables, with the qqth entry

$$\{\mathbf{x}_j^{i,l}\}_{i=1}^{n_D},$$

Er3. Evaluate the emulator at each of the state estimates: set

$$\mathbf{q}^* = \{\mathbf{X}^{i,l-\Gamma:l+\Gamma}\}_{i=1}^{n_D},$$

otherwise as item E3.

This approach entails a radical reduction in the dimension of emulator inputs *and* outputs. Due to the unusual "slicing" of the emulator input to obtain rich training data, we refer to it as the "sliced Emu-PF." We test it on a state estimation problem in fig. 9.

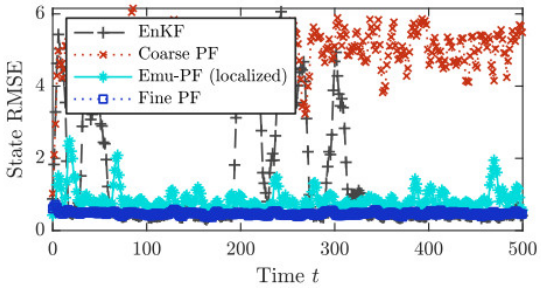


Figure 9. Summary statistics for Experiment Four, long-time state estimation with $m = 4$ observations of accuracy $\sigma_o = 1$. The median RMSE for EnKF and fine PF are similar; however the EnKF error occasionally spikes. The sliced Emu-PF of section 3.2 is stable, with no large error spikes, and performs close to the fine PF in accuracy

4. Numerical experiments and results

We consider a joint state-parameter estimation problem from [29]. The state $\mathbf{x}_j$ is generated by integrating from time t_{j-1} to t_j the system of ordinary differential equations introduced in [19],

$$\dot{x}^{(l)} = (x^{(l+1)} - x^{(l-2)})x^{(l-1)} - x^{(l)} + F^{(l)}, \tag{15}$$

commonly called the Lorenz-96 system. Superscripts in parentheses denote components of a vector, l ranges from 1 to n, and (as introduced in [29]) the forcing depends on two parameters

$$F^{(l)} = 8 + \theta^{(1)} \sin\left(\frac{2\pi l}{n\theta^{(2)}}\right). \tag{16}$$

We will compare the surrogate DA algorithms to Particle and Ensemble Kalman Filters. Our goal is to obtain performance similar to that of a Particle Filter that employs a large number of particles, N_F, but only allowing $n_D \ll N_F$ model runs in our scheme. In order to quantify the benefits, and drawbacks, of our approach, we will include the following algorithms for comparison:

- A "fine PF" employing $N_F = 10{,}000$ particles,

- A "coarse PF" employing $n_D = 100$ particles,
- An EnKF employing $n_D = 100$ particles.

While several of our results feature implementations of the Emu-PF that compete with, or exceed the performance of, the fine PF, it is important to remember that our original goal was to attain performance somewhere between the coarse and fine PF. Exact implementation details for all DA methods are given in section 4.1. We will also briefly discuss better implementations of the Particle Filter.

It is standard in the atmospheric forecasting community to eq. (14) with dimension $n = 40$, and to compute and subsequently discard a "burn-in" period of at least a thousand assimilation steps. Our benchmark fine PF is incapable of resolving the $n = 40$ case without extensive modifications that, if also implemented in an Emu-PF, can make it difficult to be sure what the contributions of the emulator are. Additionally, good filter performance during the first twenty assimilation steps are crucial for parameter estimation (assuming an initially uninformative prior on the parameters). For these reasons we choose model dimension $n = 8$ (analysed in [23]) for the initial experiments, and include the filter performance over the initial assimilation steps. The numerics section concludes by studying Emu-PF performance applied to joint state-parameter estimation in the full 40-dimensional state case.

Over all experiments, a vast quantity of information is computed. We will summarize this information with the Root Mean-Squared-Error (RMSE) and the sample variance. For parameter estimates the posterior distribution is multimodal (see section 4.2); when calculating RMSE or variances of parameter estimates, we first apply absolute values to reduce the number of modes.

4.1. Implementation details

Particle Filters all employ the merging particle filter of [22]. This filter {constructs new particles from weighted averages of extant samples when resampling; we employ the recommended weights $a_1 = 3/4$; $a_2 = (\sqrt{13} + 1)/8$; $a_3 = -(\sqrt{13} - 1)/8$. Additionally PFs employ the parameter model from [16] which, at each observation time, draws all particles slightly towards the particle mean, preserving $\alpha = 0.99$ of the variation among particles, then jitters all particles randomly by adding noise generated with standard deviation $\beta = 0.01$.

The EnKF employs multiplicative covariance inflation of 1.02. That is, when the sample forecast covariance is calculated in eq. (6), it is multiplied by 1.02 before it is used in eq. (8). Covariance inflation is a common remedy to the problem of a slightly under-dispersive ensemble in the EnKF.

The Emu-PF algorithms divide the n_D particles that are used in model runs into two groups. The first group is sampled from the fine posterior after every assimilation step, as described earlier in the paper (fig. 3, for example). The second group is not sampled from the posterior, and remains fixed over the assimilation steps. We fix this second group, comprising 20 of the 100 design variables, so that the emulator can evaluate inputs at a wide range of θ even if the subsampled group has narrow support. Figure 4c (left) shows the first group (80 bright dots) and second group (20 dark dots). The GP emulators packages fitrgp() and ppgasp() estimate correlation parameters of a GP. In this work, we choose the correlation function, $c(\cdot,\cdot)$, to be a Matérn kernel with smoothness parameter 5/2.

A modern implementation of the particle filter to a high-dimensional filtering problem should involve intensive modifications to mitigate the curse of dimensionality. Successful innovations include proposal densities [34], mixtures [7], and dimension reduction strategies including the classic Rao-Blackwellized PF or recent localized PFs [24,25]. We conclude the numerics section by displaying the compatibility of Emu-PFs with a proposal density based PF, the Optimal Proposal PF, in section 4.7. In that section we show that the Emu-PF can in fact improve on the Optimal Proposal PF in high-dimensional filtering problems. Further modifications along these lines are discussed in section 5.

4.2. Experiment details

Unless specified otherwise, we assimilate data at 1000 observation times with time step of 0.05 between them. Model and truth are integrated between these observation times with five steps of a fourth order Runge-Kutta scheme. At each of these integration steps the true value of $\theta^{(1)}, \theta^{(2)}$ is drawn from a Gaussian with mean $(2,1)^T$ and variance $0.01\mathbf{I}_2$. All DA schemes use fixed parameter estimates between assimilation steps. The discrepancy between fixed parameters in model updates for DA schemes, and varying parameters by drawing them from a distribution for the true dynamics, introduces a simple form of model error. We initialize state ensembles at $t = 0$ with a tight spread of variance $0.01\mathbf{I}_8$ around the true initial condition, which is generated randomly. By contrast the parameter ensembles are initially uninformative, being drawn from a uniform distribution on the square $(-5,5) \times (-5,5)$. The symmetry of eq. (15) ensures that the posterior distribution in the parameters is always at least bimodal, as the forcing $F^{(l)}$ is identical at $+\theta, -\theta$; but, also by symmetry, we can calculate reasonable RMSE and variance statistics for parameters by taking the absolute value of parameter estimates. All schemes employ $n_D = 100$ and $N_F = 10{,}000$.

Interpret plots of the DA schemes with the following: if two different initial conditions for eqs. (14) and (15) are integrated for a long time, the mean distance between the two trajectories will be around 5. Any DA method attaining a state RMSE value near 5 is no different to employing no assimilation. However smaller RMSE is not necessarily optimal; each DA scheme is trying to estimate the posterior, which is unknown. Generally we will compare methods to results from the fine PF.

We also present tables with summary statistics for each experiment. These tables present the mean RMSE and the median sample variance over the final 50% of assimilation steps, recorded separately for parameters and for states. We compute median variance as the mean variance is dominated by large variance terms in a few of the state variables. Generally the sample variances will appear to suggest methods are under-dispersive; but the EnKF performs better estimating the bimodal parameter distribution if it is under-dispersive than otherwise (explained further in the discussion of Experiment One).

We vary two quantities between experiments; the dimension mm of the observations, and the accuracy of the observations. We will consider $m = 2,4,8$ evenly spaced observations. The observation accuracy is measured by the scalar σ_o, which controls the observation error covariance matrix $\mathbf{R}$ from section 2.1 according to $\mathbf{R} = \sigma_o^2 \mathbf{I}_m$. More difficult experiments are obtained by reducing mm and/or σ_o. Fewer observations at each observation time lead to a more uncertain posterior, which is difficult for the Emu-PF algorithms to represent with the low number of design variables n_D. Accurate observations, that is smaller values of σ_o, are difficult for Particle Filters in general.

4.3. Experiment 1

We begin by presenting statistics for a fully observed ($m = 8$) system with observation accuracy $\sigma_o = 1.$. in fig. 6 and table 1. The Emu-PF with $\Gamma = -1$ outperforms the, equivalent in number of model runs, coarse PF. The fine PF does not appear to estimate the state variables well in fig. 6; fig. 5 is another run with the same setup in which the fine PF is clearly distinct from the coarse. Table 1 shows that the Emu-PF with $\Gamma = -1$ reliably estimates parameters about as well as the fine PF (albeit with even smaller variance) and estimates states about as well as the coarse PF (albeit with large variance).

Table 1. Summary statistics for twenty repetitions of experiment One. The 'Resampling' column counts how many resampling steps, out of a thousand, were performed by each algorithm.

	RMSE (θ)	Var (θ)	RMSE ($\mathbf{x}$)	Var ($\mathbf{x}$)	Resampling
Fine PF	0.066	0.0035	0.34	0.15	226
Coarse PF	0.79	0.0015	2.1	0.16	663
EnKF	0.048	0.0018	0.32	0.12	-
Emu-PF ($\Gamma = -1$)	0.13	0.00026	2.4	5.1	483

The EnKF performs poorly in fig. 6, with large errors compared to other schemes in both the parameters and state variables, but the median performance in table 1 is excellent; we now discuss why. Poor performance is expected, as the distribution of the parameters is bimodal and the EnKF relies on unimodal approximations. In practice we do observe that the EnKF RMSE reliably remains high for the first thirty to fifty assimilation steps of this experiment, long after the PFs have converged; however the EnKF parameter ensemble also tends to shrink over those assimilation steps. Once the parameter ensemble has shrunk sufficiently, the peaks of the posterior—visible in fig. 4c, right—are no longer both contained in the span of the ensemble, and the ensemble will move close to one or another peak in subsequent assimilation steps. That is, the EnKF is successfully locating one of the two peaks of the bimodal posterior; since the RMSE only records distance from either peak, the EnKF performance appears good in the tables.

4.4. Experiment 2: Sparse observations

Consider the more challenging setup of $m = 2$ evenly spaced observations with the same accuracy $\sigma_o = 1.$. from the previous experiment. An implementation of the Emu-PF employing both, the PCA dimension reduction from section 3.4 (to four variables), and PPE from section 3.3 to compute all emulator outputs simultaneously, is tested in fig. 7. The Emu-PF implementation with PCA not only stably estimates states and parameters under a difficult filtering problem, but out-competes both the EnKF and the fine PF (which employs $100\times$ as many model runs). This good performance from the Emu-PF with PCA far outstrips our original goal, that was just to *replicate* the performance of the fine PF. However, on repeating this experiment, we discovered that the Emu-PF employing PCA suffers issues with instability: the support of the design inputs may shrink, and then the emulator can output NaNs. To address this issue, we stabilised the design variables by adding noise with variance 0.01 to the parameters when they are subsampled (see section 3.1). This design is stable, but less technically impressive; statistics from 20 runs, showing significant improvement on the Coarse PF, are recorded in table 2. Future work will focus on less intrusive alterations to the Emu-PF design.

Table 2. Summary statistics for twenty repetitions of experiment Two.

	RMSE (θ)	Var (θ)	RMSE ($\mathbf{x}$)	Var ($\mathbf{x}$)	Resampling
Fine PF	0.074	0.0043	0.7	0.61	173
Coarse PF	0.49	0.0016	4.9	0.13	312
EnKF	0.065	0.0027	0.78	0.66	-
Emu-PF ($\Gamma = -1$)	0.38	0.00085	3.8	6.1	526
Emu-PF (PCA)	0.27	0.00051	3.1	0.58	339

Table 3. Summary statistics for twenty repetitions of experiment Three.

	RMSE (θ)	Var (θ)	RMSE ($\mathbf{x}$)	Var ($\mathbf{x}$)	Resampling
Fine PF	0.062	0.0032	0.28	0.13	243
Coarse PF	1	0.0012	3.4	0.11	739
EnKF	0.045	0.0017	0.25	0.1	-
Emu-PF ($\Gamma = -1$)	0.15	0.00034	2.4	5.1	590
Emu-PF (PCA)	0.084	0.00075	1.5	0.085	334

4.5. Experiment 3: Accurate, sparse observations

We preserve $\sigma_o = 0.5$ but reduce to $m = 4$ observations. One drawback to the Γ>0Γ>0 Emu-PF that we have observed is that it can be unstable if the filtering problem is slightly too hard; we infer that the emulator is given insufficient training data for the strongly localized input variables. Figure 8 shows the $\Gamma = 1$ Emu-PF performs significantly worse than the, technically inferior, Emu-PF with $\Gamma = -1$. In this case again the Emu-PF employing both PCA and PPE significantly improves on the Coarse PF performance. Localising strategies like the $\Gamma = 1$ approach are critical in many modern DA applications. The results of Experiment Four demonstrate that the localization strategy we have adopted is insufficient for more difficult filtering problems. We plan for future work to combine such localization strategies with the dimension reduction strategy of section 3.4.

4.6. Experiment 4: State estimation

Supposing the model parameters are known and fixed, we now showcase the localization strategy of section 3.2, the sliced Emu-PF with $\Gamma = 1$. We fix $\theta = (0,1)^T$ in all methods, so that the filtering problem is the standard Lorenz-96 model with $F = 8$, and test DA schemes that estimate the state variables. In this state estimation experiment we assimilate every second variable with $m = 4$ and standard accuracy $\sigma_0 = 1$, at each of $10{,}000$ observation times. Implementation for two algorithms differs in this experiment: the EnKF employs multiplicative inflation of 1.1 (tuned to minimize RMSE) and the PF algorithms jitter particles with white noise of variance 0.01 after each resampling step[2]. The Emu-PF with $\Gamma = 1$ and the Emu-PF employing PCA (not plotted) both attained similar error values to the coarse PF[3]. Results in fig. 9 and table 4 show the sliced Emu-PF outperform the EnKF and attain performance almost on par with the fine PF.

Table 4. Summary statistics for Experiment Four

	RMSE ($\mathbf{x}$)	Var ($\mathbf{x}$)	Resampling
Fine PF	0.47	0.15	1706
Coarse PF	5.1	0.16	9917
EnKF	1	0.096	-

Emu-PF (Localized)	0.83	0.31	3566

[2]this step is necessary here because the model is deterministic; if not jittered, the PF ensemble will collapse and all particles will be identical. Jittering is not strictly necessary for the PF in previous experiments because the parameter model is stochastic, and is not needed for the Emu-PF anywhere because the emulator already translates model uncertainty into noise.

[3]additionally, the Emu-PF with PCA halted due to an error with the PPE code.

4.7. Experiment 5: High dimensional, non-Gaussian joint state-para-meter estimation

We test the emulator-PFs in the 40-dimensional Lorenz-96 system with forcing given by eq. (15). We employ the Optimal Proposal PF (OP-PF) in all Particle Filters: a variation on the Particle Filter in which the particles are nudged in the direction of observations. The OP-PF is well described in [8,32,31] and employed in a similar context for state estimation in [20].

In addition to employing the OP-PF for all PF algorithms, we modify the Emu-PF as well. All Emu-PF variants train the emulator on the proposal of the OP-PF, and employ the weight update appropriate for OP-PF. If the emulator samples converge to the prior distribution as the amount of training data increases, then the Emu-PF will converge to an OP-PF.

We choose challenging experimental parameters: observations of every second variable, $m = 20$, corrupted with measurement errors $\sigma_o = 0.5$. We introduce a model error/noise term: at each observation time the truth, and model forecasts, are altered by additive Gaussian noise with standard deviation 1 in each component.

For this difficult filtering problem, the Fine and Coarse OP-PFs do not produce significantly different error statistics (see table 5). However the Emu-PF is promising: the 'standard' variant with $\Gamma = -1$ estimates model parameters more than twice as well as any standard method, and the localised variant with $\Gamma = 2$ also estimates parameters well while estimating the state about as well as the Particle Filters.

Table 5. Summary statistics for twenty repetitions of experiment Five.

	RMSE (θ)	Var (θ)	RMSE ($\mathbf{x}$)	Var ($\mathbf{x}$)	Resampling
Fine OP-PF	1.2	0.0075	1.9	3.3	226
Coarse OP-PF	1.2	0.004	2.0	2.8	205
EnKF	1.1	0.00042	1.5	1.8	-
Emu-PF ($\Gamma = -1$)	0.5	0.0017	2.6	3.5	243
Emu-PF ($\Gamma = +2$)	0.75	0.0035	2.0	3.7	232
Emu-PF (PCA)	1.1	0.061	2.0	2.8	238

Let us focus on the performance of two filters: the Fine OP-PF and the best-performing filter, the Emu-PF with $\Gamma = 2$. The evolution over time of the RMSE in parameters and states for each of the 20 applications of these filters is shown in fig. 10. The localised Emu-PF clearly improves over the Fine PF—despite the Emu-PF only aiming to mimic the Fine PF at lower computational cost!

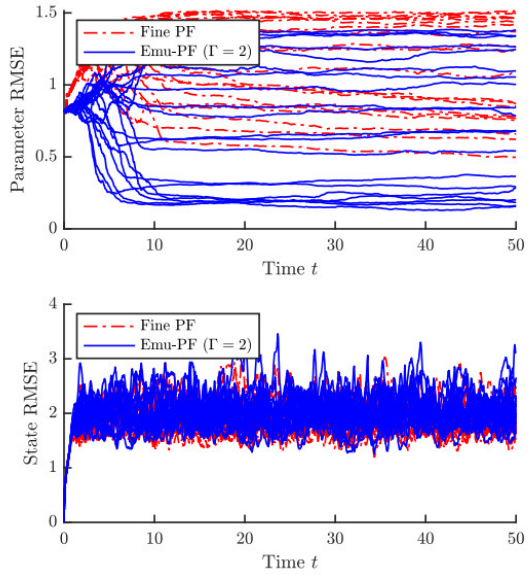


Figure 10. RMSE against time for Experiment Five: dashed red lines plot the Fine PF (formulated under the Optimal Proposal), and solid blue lines plot the best-performing Emu-PF according to table 5. There is a clear improvement in skill in parameter estimation. State estimates are similar in skill (and, importantly, do possess some skill: the state RMSE is well below 5, the approximate long-term or climatic mean RMSE of forecasting with no DA)

5. Discussion and future directions

In this work, we present a straight-for-ward utilization of statistical emulators within sequential data assimilation. We use random function models, specifically Gaussian process emulators (GPs), to learn the mapping from state and/or parameter values at one observation instance to the next. This model-learning technique pairs well with particle filters that typically require $10^3 - 10^5$ forward model runs to assimilate each observation in time. The gist of our methodology is that a GP provides interpolation between model forecasts – thought of as functions of the parameter and/or previous state values at a fixed time – and may be used to produce additional forecasts, and thus provide a cheap means to improve PF performance. Further, statistical emulators provide a built-in estimate of model performance in terms of the predictive variance of the Gaussian process. In our suite of simulation studies, we find that GP emulator-based particle filters with 100 model runs outperform particle filters with the same modest run budget and in some experiments nearly meet the performance levels of perform on par or better when compared to a 10^4 particle "gold-standard" particle filter.

We explore several variations of the basic emu-PF algorithm, both to improve performance and to test various approaches to dimension reduction within the emulator. We introduce these various adaptions to mimic two salient flavors of dimension reduction on inputs to the dynamic forward mapping—namely two forms of localized dimension reduction, and a strategy for global dimension reduction. Localization is a widely-used and effective tool in DA to eliminate the impacts of long-range correlations on estimations and forecasts. The two approaches may be combined in future implementations of Emu-PF: one can imagine utilizing "global" dimension reduction tools within the localization domain of a gridded model. We further utilize the parallel partial emulator in a variation of the Emu-PF appropriate for functional or vector-valued model output. These variations on the Emu-PF, while promising, are not agnostic to the choice of model or observing system. For example, dimension

reduction through PCA is inappropriate in turbulent systems, in which there is no clear scale separation to exploit. The choice of dimension reduction should be informed by the dynamical properties of the forward mapping.

Simulation experiments were performed on an 8-member and a 40-member Lorenz-96 system. We begin by considering a parameterized forcing that induces a bi-modal posterior distribution in parameter space. The emu-PF is able to obtain well-resolved bi-modal posteriors in parameter space with only 100 forward-model runs. We then consider a series of assimilation experiments that present an increased challenge as we lower the dimension of the observational space. We conclude that the success of the computationally cheap emu-PF with various forms of localization bodes well for this tool to be explored more widely.

A very strong asset of this methodology is that it can readily be combined with other modern advances in sequential data assimilation. For example, we combined the Optimal Proposal PF [31] in conjunction with emulators to improve on the existing performance of Particle Filters on the 40-dimensional Lorenz-96 model. Further, the approach could be combined with a Localized PF [24]. In this case, we envision a dimension reduction for the emulator based on the support of the localization(s) utilized within the Localized PF. The emulator-based particle filter also has the potential to work nicely with the Equal Weight PF [33]. One can re-express the equivalent weights problem readily on the probability density functions obtained with emulators. Then one could sample from the resulting distribution. These advanced PF techniques devise approaches to overcome the challenge of searching large sample spaces; our contribution is effectively to accelerate the sampling procedure, so that more samples can be taken.

A challenging future research direction is to include multi-scale modes in the posterior, particularly in combination with high dimensional systems.

References:

[1]	M. J. Bayarri, J. O. Berger, J. Cafeo, G. Garcia-Donato and F. Liu, Computer model validation with functional output, *Ann. Statist.*, **35** (2007), 1874-1906. doi: 10.1214/009053607000000163.
[2]	J. Betancourt, F. Bachoc, T. Klein, D. Idier, R. Pedreros and J. Rohmer, Gaussian process metamodeling of functional-input code for coastal flood hazard assessment, *Reliability Engineering & System Safety*, **198** (2020). doi: 10.1016/j.ress.2020.106870.
[3]	M. Bocquet, J. Brajard, A. Carrassi and L. Bertino, Bayesian inference of chaotic dynamics by merging data assimilation, machine learning and expectation-maximization, *Foundations of Data Science*, **2** (2020), 55-80. doi: 10.3934/fods.2020004.
[4]	J. Brajard, A. Carassi, M. Bocquet and L. Bertino, Combining data assimilation and machine learning to emulate a dynamical model from sparse and noisy observations: A case study with the Lorenz 96 model, *J. Comput. Sci.*, **44** (2020), 11pp. doi: 10.1016/j.jocs.2020.101171.
[5]	A. Carrassi, M. Bocquet, L. Bertino and G. Evensen, Data assimilation in the geosciences: An overview of methods, issues, and perspectives, *Wiley Interdisciplinary Reviews: Climate Change*, **9** (2018). doi: 10.1002/wcc.535.

[6]	E. Cleary, A. Garbuno-Inigo, S. Lan, T. Schneider and A. M. Stuart, Calibrate, emulate, sample, *J. Comput. Phys.*, **424** (2021), 20pp. doi: 10.1016/j.jcp.2020.109716.
[7]	D. Crisan and K. Li, Generalised particle filters with Gaussian mixtures, *Stochastic Process. Appl.*, **125** (2015), 2643-2673. doi: 10.1016/j.spa.2015.01.008.
[8]	A. Doucet, N. de Freitas and N. Gordon, *Sequential Monte Carlo Methods in Practice*, Statistics for Engineering and Information Science, Springer-Verlag, New York, 2001. doi: 10.1007/978-1-4757-3437-9.
[9]	G. Evensen, *Data Assimilation. The Ensemble Kalman Filter*, Springer-Verlag, Berlin, 2009. doi: 10.1007/978-3-642-03711-5.
[10]	G. Evensen, The ensemble Kalman filter: Theoretical formulation and practical implementation, *Ocean Dynamics*, **53** (2003), 343-367. doi: 10.1007/s10236-003-0036-9.
[11]	G. A. Gottwald and S. Reich, Supervised learning from noisy observations: Combining machine-learning techniques with data assimilation, *Phys. D*, **423** (2021), 15pp. doi: 10.1016/j.physd.2021.132911.
[12]	M. Gu and J. O. Berger, Parallel partial Gaussian process emulation for computer models with massive output, *Ann. Appl. Stat.*, **10** (2016), 1317-1347. doi: 10.1214/16-AOAS934.
[13]	M. Gu, J. Palomo and J. O. Berger, RobustGaSP: Robust Gaussian Stochastic Process Emulation in R, *The R Journal*, **11** (2019), 112-136. doi: 10.32614/RJ-2019-011.
[14]	M. E. Johnson, L. M. Moore and D. Ylvisaker, Minimax and maximin distance designs, *J. Statist. Plann. Inference*, **26** (1990), 131-148. doi: 10.1016/0378-3758(90)90122-B.
[15]	K. Law, A. Stuart and K. Zygalakis, *Data Assimilation. A Mathematical Introduction*, Texts in Applied Mathematics, 62, Springer, Cham, 2015. doi: 10.1007/978-3-319-20325-6.
[16]	J. Liu and M. West, Combined parameter and state estimation in simulation-based filtering, in *Sequential Monte Carlo Methods in Practice*, Stat. Eng. Inf. Sci., Springer, New York, 2001,197–223. doi: 10.1007/978-1-4757-3437-9_10.
[17]	J. S. Liu and R. Chen, Sequential Monte Carlo methods for dynamic systems, *J. Amer. Statist. Assoc.*, **93** (1998), 1032-1044. doi: 10.1080/01621459.1998.10473765.
[18]	X. Liu and S. Guillas, Dimension reduction for Gaussian process emulation: An application to the influence of bathymetry on tsunami heights, *SIAM/ASA J. Uncertain. Quantif.*, **5** (2017), 787-812. doi: 10.1137/16M1090648.
[19]	E. N. Lorenz, *Predictability - A problem partly solved*, in Proceedings of Seminar on Predictability, Cambridge University Press, Reading, UK, 1996. doi: 10.1017/CBO9780511617652.004.
[20]	J. Maclean and E. S. V. Vleck, Particle filters for data assimilation based on reduced-order data models, *Q. J. Roy. Meteor. Soc.*, **147** (2021), 1892-1907. doi: 10.1002/qj.4001.
[21]	M. Morzfeld and D. Hodyss, Gaussian approximations in filters and smoothers for data assimilation, *Tellus A*, **71** (2019). doi: 10.1080/16000870.2019.1600344.
[22]	S. Nakano, G. Ueno and T. Higuchi, Merging particle filter for sequential data assimilation, *Nonlin. Processes Geophys.*, **14** (2007), 395-408. doi: 10.5194/npg-14-395-2007.
[23]	D. Orrell and L. A. Smith, Visualizing bifurcations in high dimensional systems: The spectral bifurcation diagram, *Internat. J. Bifur. Chaos Appl. Sci. Engrg.*, **13** (2003), 3015-3027. doi: 10.1142/S0218127403008387.
[24]	J. Poterjoy, A localized particle filter for high-dimensional nonlinear systems, *Monthly Weather Review*, **144** (2016), 59-76. doi: 10.1175/MWR-D-15-0163.1.

[25]	R. Potthast, A. Walter and A. Rhodin, A localized adaptive particle filter within an operational NWP framework, *Monthly Weather Review*, **147** (2019), 345-362. doi: 10.1175/MWR-D-18-0028.1.
[26]	C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*, Adaptative Computation and Machine Learning, MIT Press, Cambridge, MA, 2006. Available from: http://www.gaussianprocess.org/gpml/chapters.
[27]	S. Reich and C. Cotter, *Probabilistic Forecasting and Bayesian Data Assimilation*, Cambridge University Press, New York, 2015. doi: 10.1017/CBO9781107706804.
[28]	J. Sacks, W. J. Welch, T. J. Mitchell and H. P. Wynn, Design and analysis of computer experiments, *Statist. Sci.*, **4** (1989), 409-423. doi: 10.1214/ss/1177012413.
[29]	N. Santitissadeekorn and C. Jones, Two-stage filtering for joint state-parameter estimation, *Monthly Weather Review*, **143** (2015), 2028-2042. doi: 10.1175/MWR-D-14-00176.1.
[30]	T. J. Santner, B. J. Williams and W. I. Notz, *The Design and Analysis of Computer Experiments*, Springer Series in Statistics, Springer, New York, 2018. doi: 10.1007/978-1-4939-8847-1.
[31]	C. Snyder, Particle filters, the "optimal" proposal and high-dimensional systems, in *Proceedings of the ECMWF Seminar on Data Assimilation for Atmosphere and Ocean*, 2011, 1–10. Available from: https://www.ecmwf.int/sites/default/files/elibrary/2012/12354-particle-filters-optimal-proposal-and-high-dimensional-systems.pdf. Google Scholar
[32]	C. Snyder, T. Bengtsson, P. Bickel and J. Anderson, Obstacles to high-dimensional particle filtering, *Monthly Weather Review*, **136** (2008), 4629-4640. doi: 10.1175/2008MWR2529.1.
[33]	P. J. van Leeuwen, Nonlinear data assimilation in geosciences: An extremely efficient particle filter, *Q. J. Roy. Meteor. Soc.*, **136** (2010), 1991-1999. doi: 10.1002/qj.699.
[34]	P. J. van Leeuwen, H. R. Künsch, L. Nerger, R. Potthast and S. Reich, Particle filters for high-dimensional geoscience applications: A review, *Q. J. Roy. Meteor. Soc.*, **145** (2019), 2335-2365. doi: 10.1002/qj.3551.
[35]	W. J. Welch, R. J. Buck, J. Sacks, H. P. Wynn, T. J. Mitchell and M. D. Morris, Screening, predicting, and computer experiments, *Technometrics*, **34** (1992), 15-25. doi: 10.2307/1269548.