Advanced Quantification of Uncertainties In Fusion modelling at the Exascale with model order Reduction (AQUIFER)
UKAEA NEPTUNE T/AW085/21
Annual report 06/04/2022 - 05/04/2023

Serge Guillas
University College London

29 March 2023

Many activities are joint with the EPSRC SEAVEA project due to synergies and cost efficiencies (e.g. PI time, events, HPC access to ARCHER2, international interactions and dissemination) as mentioned in the AQUIFER project description. The team includes:

- Prof Serge Guillas, PI
- Prof Peter Coveney, coI
- Dr. Matthew Graham, Senior Research Data Scientist
- Dr. Tuomas Koskela, Senior Research Software Engineer
- Yiming Yang, PhD student

1 Events and reporting

Our level of direct interactions is high, as we meet and report

- fortnightly at NEPTUNE Progress Meetings on Thursdays,
- fortnightly in AQUIFER technical meetings on Mondays: Ed Threlfall, Ander Gray and James Buchanan represent UKAEA.

Our events have been joint with EPSRC SEAVEA. These have included:

- Hackathons 23/06/2022 (linked to ICCS 2022), 8-9/12/2022, with another one planned for Spring 2023.
- Hack session with the developers on 2022-05-06 fully capturing the new work in SEAVEAtk (FabNEPTUNE)
- UKAEA workshop, 05-06/09/2022 (Abingdon)
- Applications Meeting, 12/12/2022, UCL
- UCL-UKAEA collaborative workshop, 10/03/2023, UCL

2 Functional Emulation for temperature profile: Reduced Order Modelling

2.1 Study Case: An-isotropic Heat Transportation

The model for time evolution of the temperature field T is thermal diffusion, which in a plasma after Braginskii parametrization gives:

$$\frac{3}{2}N\frac{\partial T}{\partial t} = \nabla \cdot \Big((k_{\|}\mathbf{b}(\mathbf{b}\nabla T) + k_{\perp}(\nabla T - \mathbf{b}[\mathbf{b}\nabla T])\Big)$$

where b indicates the magnetic field and $k_{\|}, k_{\perp}$ are the thermal conductivities with respect to the direction parallel or perpendicular along the b. The homogeneous Dirichlet boundary conditions are set on the left, top, and right boundaries of the spatial domain.

In this case, we are interested in emulating the relationship between direction θ of $\mathbf{b}$ and the temperature profile Tx on the bottom boundary at steady state by emulator such that:

$$f_{emulator} : \theta \to T_x, \theta \in [0, \frac{\pi}{2}], T_x \in C[0, 1]$$

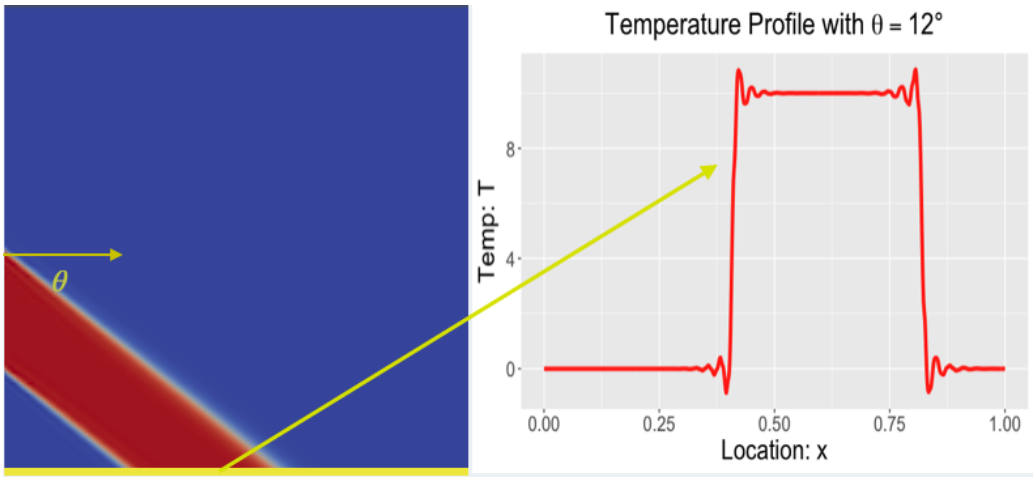


Figure 1: (left) Temperature transfer with incident angle θ (right) Temperature profile on bottom boundaries given θ

2.2 Methods: Outer Product Emulator

The high-resolution physical simulation generates high-dimensional data over discretized spatial domains, such as wave heights in a particular region of the ocean or temperature fields over a heating system. However, the high dimensionality of the data poses significant challenges when building statistical emulators. Traditional emulators often struggle with scalability or accuracy.

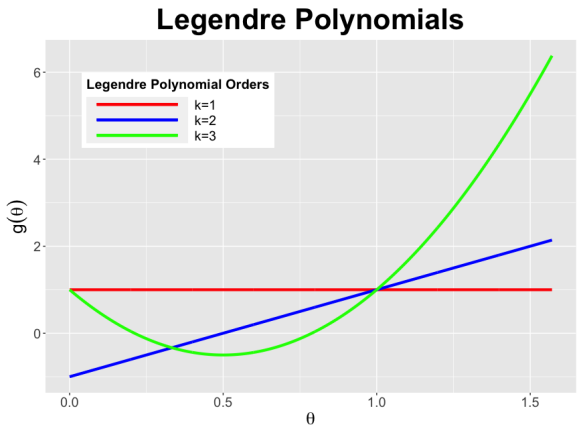
To balance scalability and accuracy, the Outer Product Emulator (OPE) has been introduced. The OPE creates a single emulator for all simulation outputs over the entire domain and simplifies the representation of fitted functions using products of component functions. In general, the OPE takes the form:

$$f(r, s) = \sum_{j=1}^{v} \beta_j g_j(r, s) + \epsilon(r, s)$$

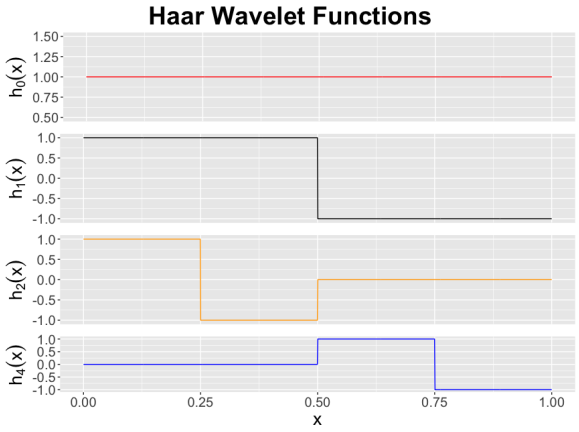
Here, $f(r, s)$ is the simulator output with input r at output location s, g_j are the regressor functions, β_j are unknown coefficients, and ϵ is the residual assumed to be a Gaussian Process (GP) such that

$\epsilon \sim GP(0, k_\lambda(,\cdot,))$.

The OPE has two main characteristics that distinguish it from conventional multivariate emulators. First, the covariance function of the residuals is separated into inputs r and outputs s, that is, $k_\lambda(r, s, r', s) = k^r_\lambda(r, r') \times k^s_\lambda(s, s')$. Second, the regressor functions g_j are given by products $g_j(r, s) = g^r_j(r) \bigotimes g^s_j(s)$ where $\bigotimes$ is the outer product symbol. In this case, the Legendre Polynomials are used as the input regressor, with $g^r_j(\theta) = 1, 6\theta - 1, 6\theta^2 - 6$. However, due to the sharp changes in the temperature profile shown in Figure 1, commonly used smooth basis functions such as Legendre polynomials and Fourier basis are not suitable. Instead, we select the Haar wavelets as the basis functions for our output regressors, see Figures 2a and 2b.



(a) Legendre Polynomials



(b) Haar Wavelet Functions

Figure 2: Choices for Regressor Functions

2.3 Results

We run the simulator over 40 different θs which are evenly chosen to cover the input parameter space. For each run, there are 641 discretised values over spatial field. We implement the OPE in two settings:

- **Emulation for Simulation**: In this case, We use all the 641 discretised values to fit the OPE for emulating the computational model.
- **Emulation for pseudo-experimental "observation"**: We uniformly sampled 80 values from the 641 discretised values and added with random noise for fitting a profile by observed experimental data.

The results demonstrate the benefits and robustness (for pseudo observations) of the reduced order modelling approach using functional emulation, see Figure 3. However, with more input parameters, higher dimensional outputs and coupled models, and for sharper input-output functional forms (e.g. lower angles) more investigation is under consideration: possible use of linked emulations (Ming and Guillas 2021) and output dimension reduction (Zhang, Mak, and Dunson 2022) as well as Deep GPs (Ming, Williamson, and Guillas 2022).

3 Deep GPs and physics-aware emulation

3.1 Deep Gaussian Process (DGP)

A Deep Gaussian Process (DGP) is a probabilistic model that builds on the idea of a Gaussian process (GP) by extending it to multiple layers. The model is comprised of multiple layers of GP models, where each layer is a GP that takes the output of the previous layer as its input (see Figure 4). One of the key advantages of DGPs over traditional GPs is their ability to model complex, highly non-linear functional forms more effectively, which makes them well-suited for emulating non-stationary

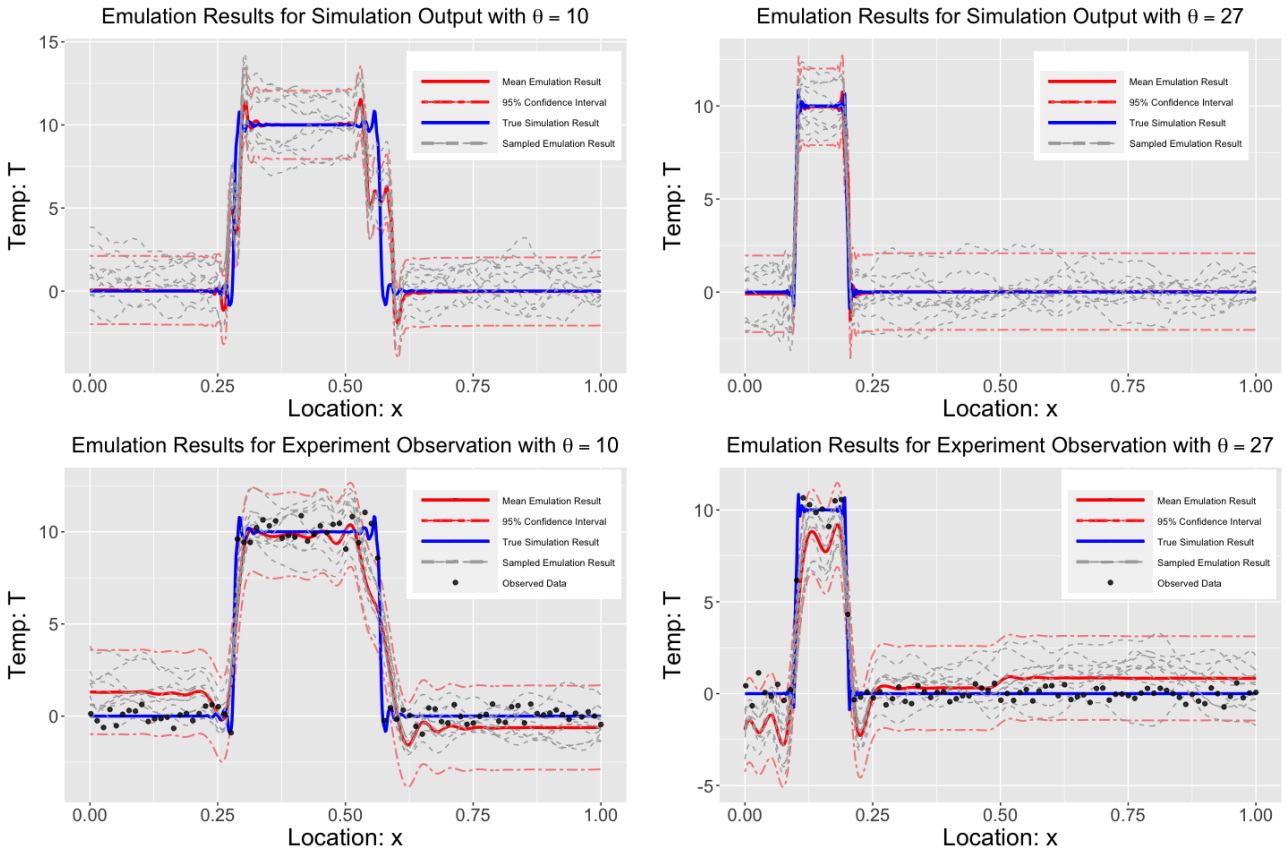


Figure 3: Emulation results for simulation outputs (first row) and pseudo observations with two different θs. The red curves are the mean emulation result, the blue dotted red curves are the 95% Confidence Interval, the blue curves are the true simulation result, and the dotted gray curves are sampled emulation result.

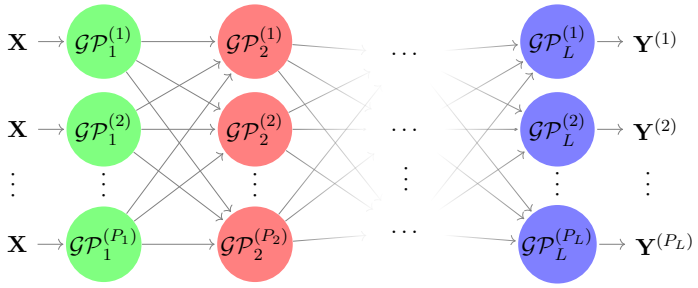


Figure 4: The generic DGP hierarchy

functions, as we will discuss in the following case study. Furthermore, DGPs provide more informative uncertainty estimation in predictions (Ming, Williamson, and Guillas 2022), making them valuable in applications where accurate risk assessment is critical.

3.2 Study Case: Simplified Lorenz Model

The Lorenz Model is derived from a 2D model of Rayleigh-Benard convection:

$$\begin{aligned}\dot{x} &= \sigma(y - x)\\ \dot{y} &= x(\rho - z) - y\\ \dot{z} &= xy - \beta z\end{aligned}$$

where ρ, σ, β are respectively the Rayleigh Number (Ra), the Prandtl Number (Pr), and the coupling strength. The Nusselt number (Nu) for this model, representing the heat flux out is $Nu = 1 + \frac{2z}{\rho}$. The

$(Ra, Pr) - Nu$ surface of the Lorenz model is non-stationary and exists many sub-regimes behaving differently which are shown in Figure 5. In the study, we fixed the coupling strength to 1. The domain for Rayleigh number and Prandtl number is $[0, 100]$.

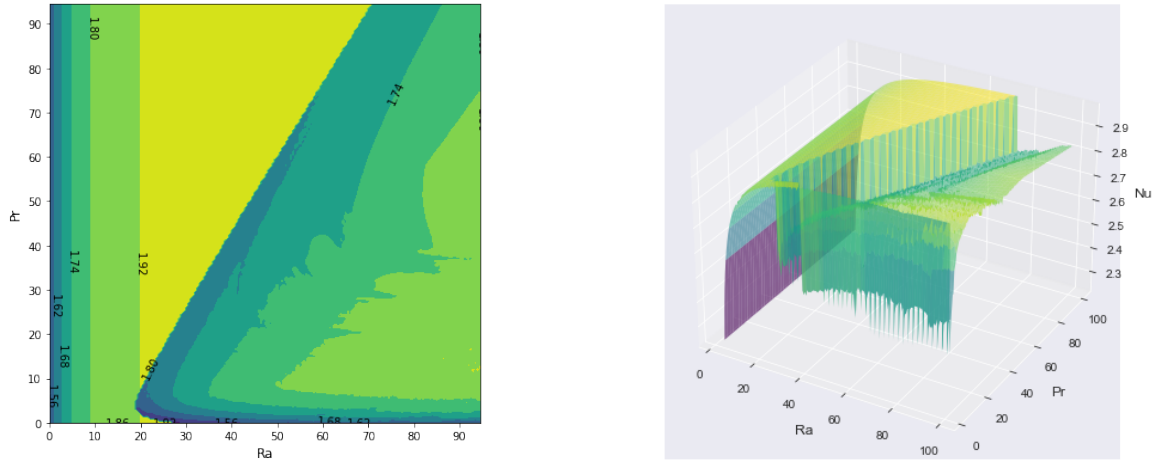


Figure 5: The $(Ra, Pr) - Nu$ surface of Lorenz Model. There are many level sets which show an extreme non-stationarity. (left) 2D contour plot (right) 3D plot.

3.3 DGP Results

In this section, we demonstrate the powerful ability of Deep GP to emulate non-stationary function: the $(Ra, Pr) - Nu$ surface of the Lorenz model. We compare Deep GP with the original GP, and Treed GP which combine the flexibility of decision trees with the smoothness of Gaussian processes to emulate functions with complex structures. The sampling scheme is the Latin hypercube sampling with domain $\mathbf{x} \in [0.1, 100.1]^2$, which is implemented by the MOGP package [1]. The settings for each model are outlined below:

- **Gaussian Process**: The GP model with mean function $m(x) = 0$ and Matérn-2.5 Kernel $K(x, x') = (1 + \frac{\sqrt{5}|x-x'|}{\gamma} + \frac{5(x-x')^2}{3\gamma^2}) \exp(-\frac{\sqrt{5}|x-x'|}{\gamma})$ with length parameter $\gamma = 0.5$.
- **Deep Gaussian Process** The DGP model consists of 2 GPs model above, with length parameters $\gamma_1 = 1.0, \gamma_2 = 0.5$ for stabilizing the numerical computation. The 2 GPs are connected in serial: We use the dgpsi[2] package for implementing the DGP, which are available in both python and R.
- **Treed Gaussian Process** The treed GP method employs a decision tree algorithm to divide a domain into multiple non-overlapping subsets. Let $r \in 1, ..., R$ denote the R non-overlapping regions partitioned by the tree $\mathcal{T}$ drawn from the tree prior. In each region, a GP regression is carried out using a default GP prior, as set in the tgp[3] package. The kernel used is also Matérn-2.5 with length parameter $\gamma = 0.5$, which is the same as the other models compared to ensure fairness. The tree structure and each component GP are jointly optimized by maximizing the likelihood.

The qualitative results are presented in Figure 6. As shown in the figure, both GP and DGP start to capture the discontinuities with only 30 samples, whereas Tree GP requires 100 samples. This could be due to the requirement that splits of Tree GP must be parallel to the axis, making it less flexible with fewer samples. With 100 fitting samples, we observe that DGP outperforms the other models and shows the discontinuous boundaries more clearly. In Figure 7, we compute the Root Mean Square Error (RMSE) to measure the performance improvements with an increasing number of

[1] https://github.com/alan-turing-institute/mogp-emulator
[2] https://github.com/mingdeyu/DGP
[3] https://cran.r-project.org/web/packages/tgp/index.html

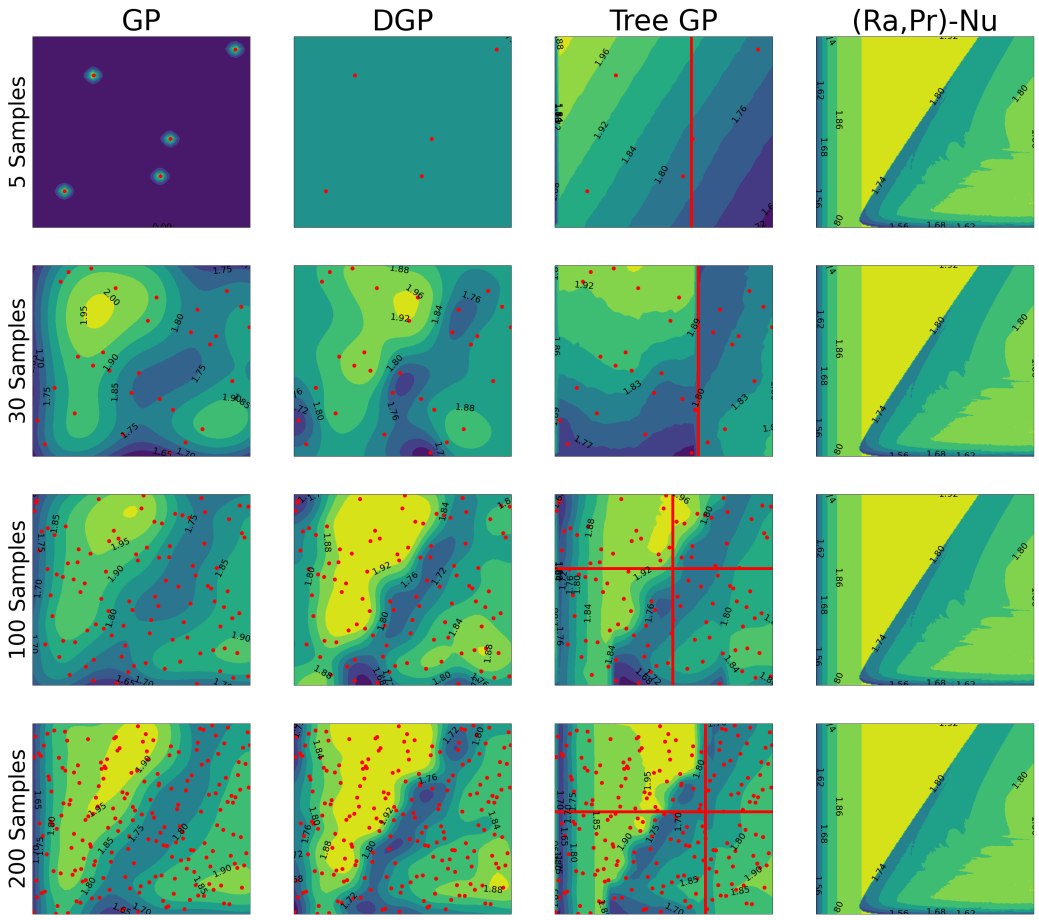


Figure 6: The mean of emulation result: GP, DGP Tree GP and True surface (left to right) with 5, 30, 100, 200 samples (top to bottom)

training samples. Each model is fitted with $n = 5, 10, 15, 30, 50, 100, 200$ samples and evaluated over 2500 test points evenly spaced over the domain $[0.1, 100.1] \times [0.1, 100.1]$. Overall, DGP has the lowest RMSE for all numbers of fitting samples. However, RMSE averages the results over the entire domain, which may obscure the emulation performance of our specific region of interest - the discontinuities. In the future, a more comprehensive metric should be developed to evaluate emulation results for non-stationary functions.

In some downstream tasks, such as uncertainty quantification and parameter calibration, multiple runs of the emulator may be required over the same evaluated point. Therefore, it's important to consider the running time of the emulator. We measured the predicting time over 2500 test points and demonstrated the fitting and predicting time of each model in Figure 8. We observed that the fitting and predicting time of GP and Treed GP remained constant with the increase of the number of fitting samples. On the other hand, the fitting time of DGP increased linearly with the increase of the number of fitting samples, whereas the predicting time increased exponentially. This is a crucial limitation in large-scale and high-resolution emulation. For DGP, the prediction complexity is $\mathcal{O}(mk(n^3 + n))$, where m is the number of prediction samples, k is the number of GP components in DGP structure, and n is the number of fitting samples.
In summary, DGP outperforms standard GP and Treed GP in emulating non-stationary functions in terms of accuracy and boundary detection. However, the heavy computational burden may hinder its applications in large-scale and high-resolution emulations.

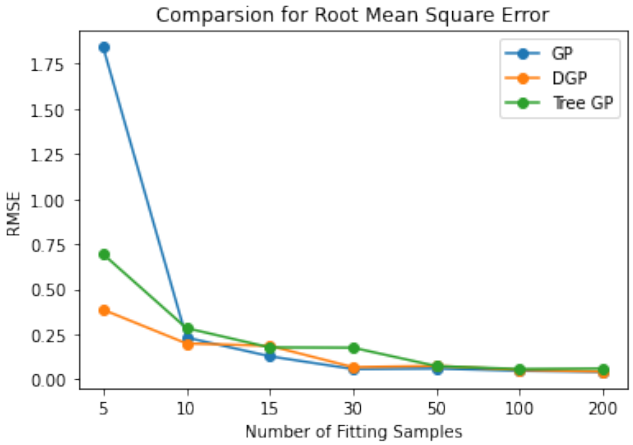


Figure 7: The Root Mean Square Error (RMSE) of emulation means with respect to the number of fitting samples.

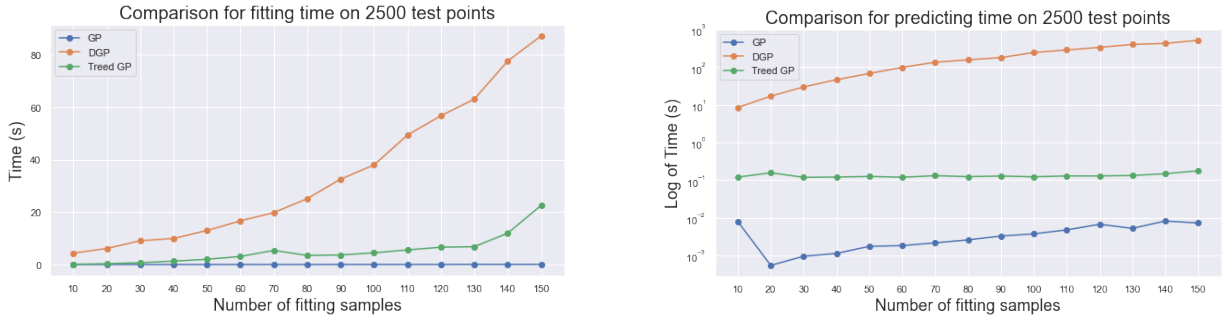


Figure 8: Comparison of Fitting (left) and Prediction (right) Time with respect to the number of fitting samples over 2500 test samples.

3.4 Physics Aware Emulation

In this section, we integrate established principles of physics into our sampling methodologies to assess their potential for emulating non-stationary functions. In the Lorenz model's $(Ra, Pr) - Nu$ surface, there are known asymptotes that dictate the stability of the fixed points in Lorenz dynamics. These asymptotes are represented by the discontinuous boundaries shown in Figure 5, and are characterized by upper and lower limits $Pr_u(Ra)$ and $Pr_l(Ra)$ (see Figure 9), $Pr_u = Ra - 2(b+2)$ and $Pr_l = b+1$.

We created a physics-aware sampling region that covers both asymptotes, with a margin of 15 units (as shown in Figure 9). To cover the non-cube region, we utilized two Latin Hypercube designs (as depicted in Figure 10). In the following investigation, we compared the physics-aware sampling approach with the Latin Hypercube sampling method over the entire domain. In particular, we employed an initial sampling of one-third of the total number of samples over the entire domain for the physics-aware approach, followed by the remainder of the samples being taken from the physics-aware region.

3.5 Physics-aware Emulation Result

The results presented in Figure 11 demonstrate the effectiveness of physics-aware sampling in identifying boundary contours. The figure illustrates that GP with physics-aware sampling requires fewer samples as compared to GP without the sampling scheme. However, DGP requires more samples to detect the boundary contours than both GP with and without physics-aware sampling but DGP outperform others in terms of accuracy with enough samples. Furthermore, Figure 12 shows that both GP and DGP with the physics-aware sampling have lower RMSE values compared to the GP without the sampling scheme. This indicates that the physics-aware sampling scheme significantly enhances the emulation of non-stationarity, especially with a limited number of samples (see RMSE

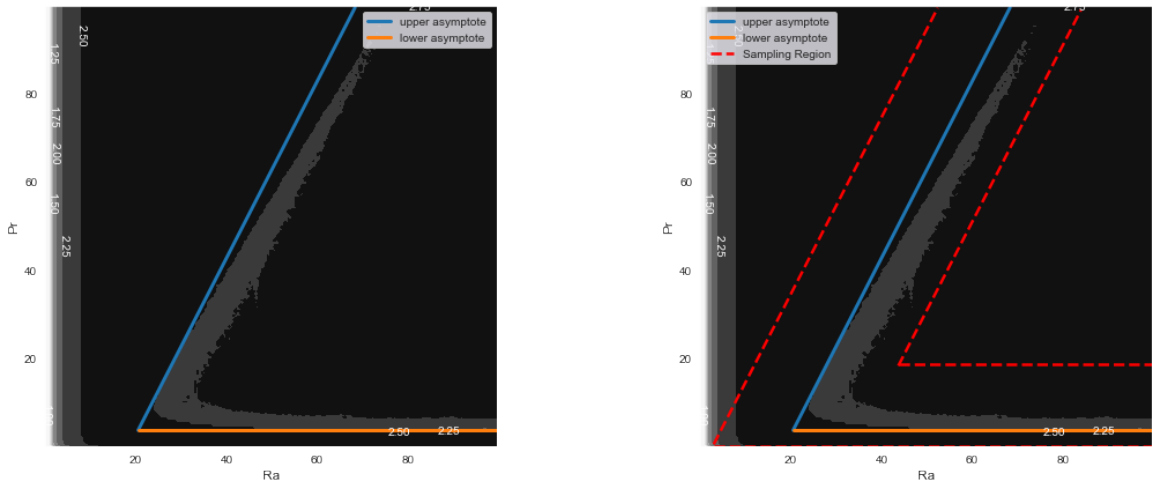


Figure 9: Left: The upper and lower asymptotes $Pr_u(Ra)$ and $Pr_l(Ra)$. Right: Physics-aware sampling region.

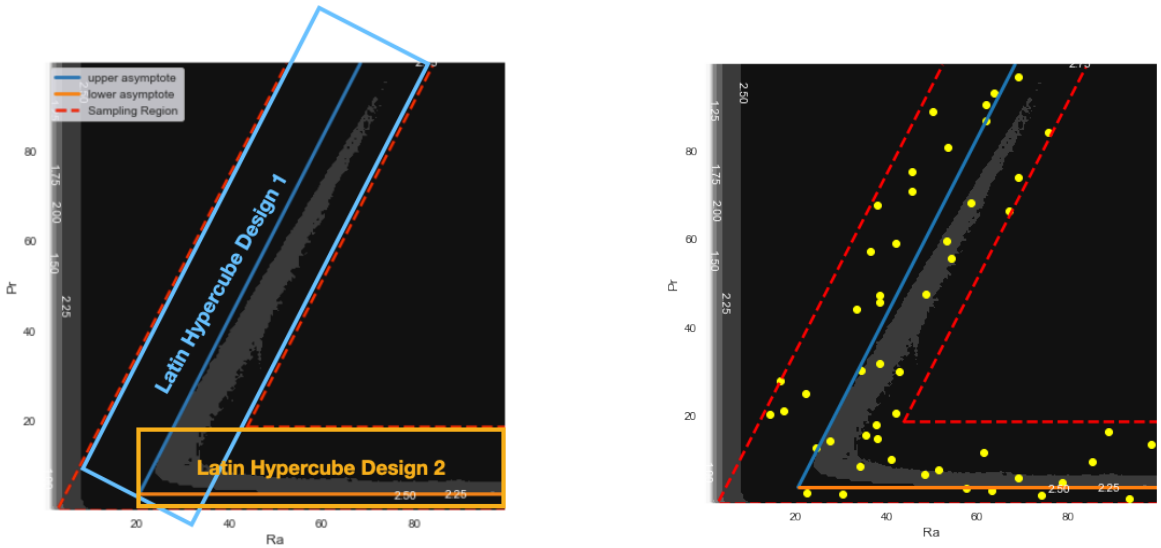


Figure 10: (left) Physics-aware Sampling Scheme (right) An example for the sampling scheme

of 30 samples in Figure 12). Moreover, the RMSEs are also smaller for larger number of samples compared to conventional space-filling schemes.
Although our setup has some evident flaws, such as an overlapping sampling region by two Latin Hypercube designs and a small uncovered area at the left-bottom corner (see Figure 10 (left)), the results show that the sampling scheme improves accuracy and sample efficiency. These findings confirm the effectiveness of incorporating prior knowledge of physics into the emulation approaches. Although this prior knowledge is very strong and can not be available for every problem, we can use such applications under a careful design. In conclusion, the results of this study demonstrate the potential benefits of combining machine learning with physical priors. Moving forward, further research in this area will be critical for advancing our understanding in complex systems and developing more effective tools for solving real-world problems.

4 Treed-GPs with using non-axis aligned splits

We have done some initial exploration of extending the Treed Gaussian (TGP) approach above to using non-axis aligned splits, in anticipation of this being able to better reflect the sort of input-output relationships we expect for the phase-diagrams of both the toy Lorenz 1963 + diffusion model, and the Nektar++ natural vertical convection model. As a simpler starting case we look a model

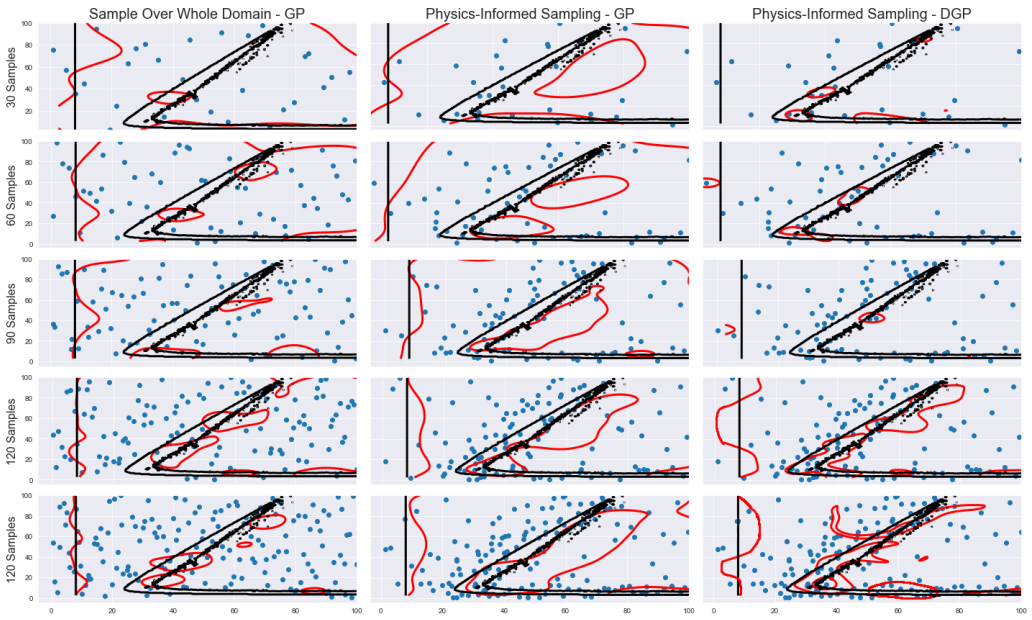


Figure 11: Contour level plots of the predicted means (red curves) and true discontinuous boundaries (black curves). Columns represent methods used, namely GP, GP with physics-aware sampling, and DGP with physics-aware sampling (left to right). Rows correspond to the number of fitting samples used, with 30, 60, 120, and 150 samples (top to bottom).

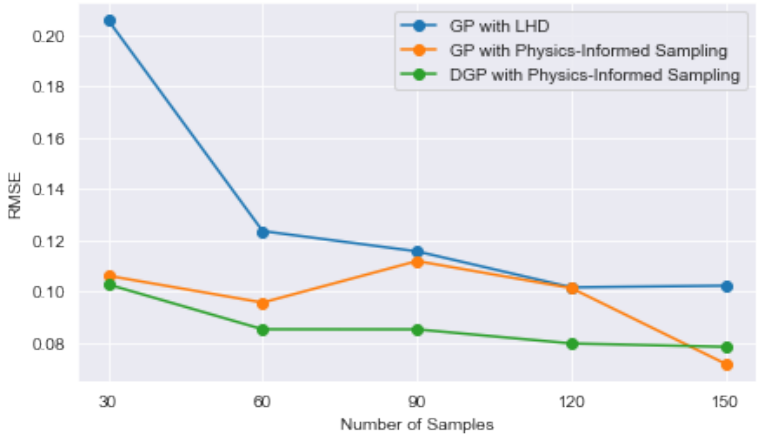


Figure 12: The RMSE for GP, GP with physics-aware sampling, and DGP with physics-aware sampling

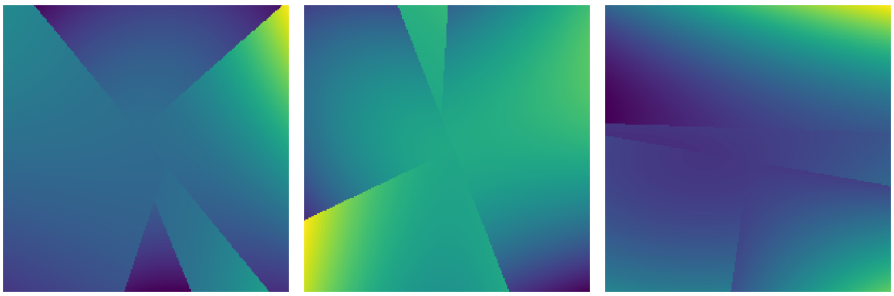


Figure 13: Proof-of-concept (no fit) splitting examples using TGP with non-axis aligned splits

which uses linear inequalities ($a \times x + b > 0$ for parameters a and b and input x) to do the splits at each level of the tree and uses a fixed-degree polynomial in the input as the prediction models at each tree leaf (with the polynomial coefficients parameters to be fitted); it should be tractable to extend this to instead use Gaussian process predictors at the leaf nodes, as in the original Bayesian treed GP

model. For example for a 2D input space generating random parameters for a tree of fixed depth of 4 produces outputs surface like Figure 13 for this simplified model.

This can then be combined with a simple independent Gaussian observation noise model to give a parametric model which can be fitted to simulations using Markov chain Monte Carlo (MCMC). For a fixed tree topology the parameters of the model – the coefficients for the split linear equalities, plus polynomial coefficients of the outputs predictors – are of fixed dimension and we can apply standard MCMC methods to update them. As the forward model is implemented in JAX (https://jax.readthedocs.io/) we can use automatic differentiation to compute the gradient of the posterior density with respect to these parameters to use gradient based MCMC methods such as the Metropolis-adjusted Langevin algorithm and Hamiltonian Monte Carlo.

The posterior density is non-smooth with respect to the split parameters so it is unclear how effective gradient based methods will be at updating these parameters, but we can alternate updates to the predictor parameters with the split parameters fixed (with the posterior density then a smooth function of the predictor parameters) with updates to the split parameters using non-gradient based proposals. The split tree topology can also be updated using a reversible jump MCMC type approach as in the original Bayesian treed GP paper.

5 Polynomial Chaos Expansion (PCE) and Stochastic Collocation (SC) surrogate models: novel Simplex Stochastic Collocation (SSC)

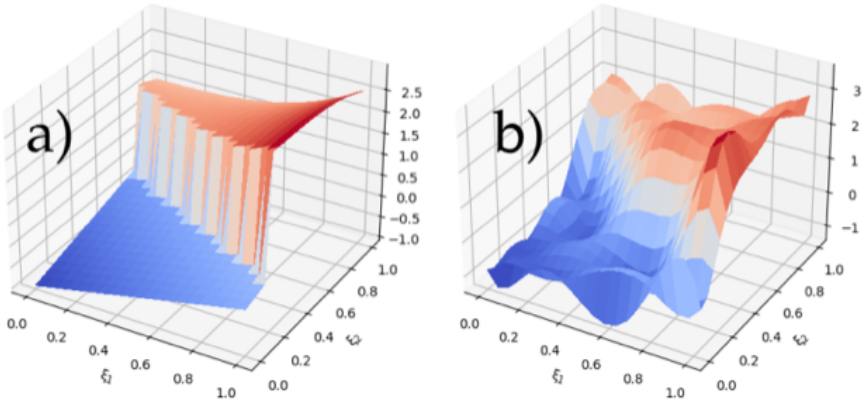


Figure 14: Discontinuous test function and standard SC surrogate.

As stated in the proposal, in D2.1 (Surrogate models, Model Order Reduction and verification) we have extended EasyVVUQ with capability to handle QoIs which display discontinuities or high gradient in the stochastic input space. This is important since surrogate models based on global polynomials (PCE/SC) will display nonphysical oscillations, see Figure 14. In particular we implemented the Simplex Stochastic Collocation (SSC) method (Edeling, Dwight, and Cinnella 2016), which (unlike SC) employs the Delaunay triangulation to discretize the probability space into simplex elements. Using such a multi-element technique has the advantage that local mesh adaptation can be performed, such that only regions of interest are adaptively refined.

The SSC method is 'physics-aware' (related to D2.6) in the sense that it can automatically detect regions in the input space where the function is irregular. It achieves this by enforcing the so-called Local Extremum Conserving (LEC) limiter in all simplex elements. Skipping details for brevity, this limiter flags elements through which a discontinuity runs, increasing the probability that the code is evaluated within that element at the next iteration. Simultaneously, the local polynomial order of the interpolation point stencil of that element is reduced, which avoids nonphysical oscillations as in Figure 14 b. The SSC sampling plan and surrogate are shown in Figure 15, which can be replicated by running the Jupyter notebook tutorial in the EasyVVUQ 'tutorials' folder [4]. Next steps involving

[4] https://github.com/UCL-CCS/EasyVVUQ/blob/dev/tutorials/simplex_stochastic_collocation_tutorial.ipynb

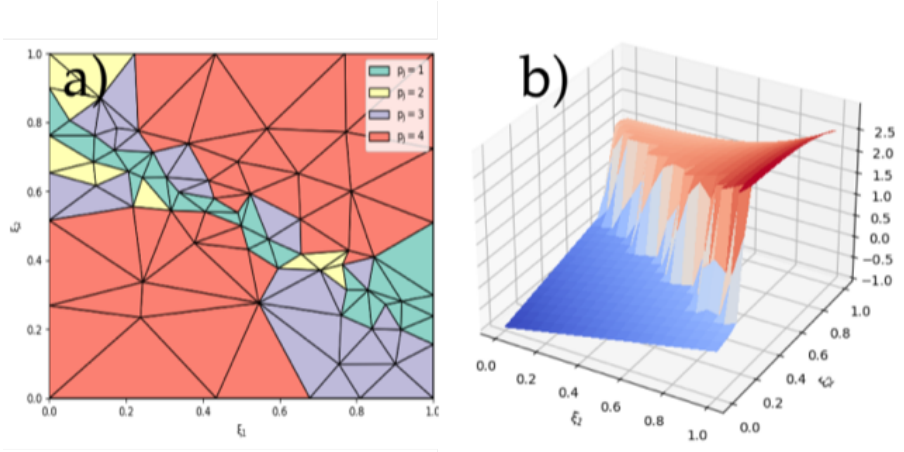


Figure 15: Local mesh and resulting SSC surrogate for the test function in 14 a.

applying the SSC method to a Nektar++ case.

6 Calibration and data assimilation in ANAET model

6.1 Introduction

In this section we consider the problem of inferring the state trajectories, and potentially parameters, of models of dynamical systems in which the evolution of the state over time is governed by a set of differential equations. We consider both cases in which the state evolution is deterministic and described by a system of *ordinary differential equations* (ODEs), and where the dynamics are driven by random noise processes, described by a system of *stochastic differential equations* (SDEs). We describe two computational approaches for estimating the posterior distribution on the model parameters and/or state trajectories given (noisy) observations of the state over time — a gradient-based Markov chain Monte Carlo method and particle filtering — and illustrate these approaches on a test model of magnetohydrodynamic instability. We identify some of the challenges inherent to performing inference and filtering in such models and discuss some alternative computational methods which may be relevant to deal with these issues.

6.2 Notation

Bold symbols, for example $\boldsymbol{x}$ indicate vector quantities. The set of consecutive integers from a to b inclusive is denoted $a{:}b$ and when used to subscript a symbol this indicates an indexed collection, for example $x_{1:N} = (x_n)_{n=1}^N$. The set of real numbers is denoted $\mathbb{R}$, and non-negative reals $\mathbb{R}_{\geq 0}$. A random variable x being distributed to or sampled from a distribution P is denoted $x \sim P$; similarly $x \mid y \sim P(y)$ indicates the conditional distribution of x given y is $P(y)$. We will use *probability distribution* to refer to probability measures, that is $[0, 1]$ valued functions on sets of outcomes (events) in the sample space, which we distinguish from a *density function* for the distribution with respect to an appropriate reference measure, that is a $\mathbb{R}_{\geq 0}$ valued function on the sample space. In all cases here we will consider distributions on real values or real vector spaces and will define densities with respect to the Lebesgue measure, so that for a distribution P with density function p we have $P(A) = \int_A p(x)\,\mathrm{d}x$ for all events A. A Dirac measure (point mass) at $\boldsymbol{x}$ is denoted $\delta_{\boldsymbol{x}}$, a normal distribution with mean m and variance v is $\mathsf{Normal}(m, v)$, a multivariate normal distribution with mean vector $\boldsymbol{m}$ and covariance matrix C is $\mathsf{Normal}(\boldsymbol{m}, C)$ and a half-normal distribution with scale parameter s (that is a normal distribution $\mathsf{Normal}(0, s^2)$ truncated below at zero) is $\mathsf{HalfNormal}(s)$.

6.3 Deterministic evolution models

A wide class of dynamical systems can be modelled by an explicit system of autonomous first order ODEs

$$\frac{\mathrm{d}\boldsymbol{x}(t)}{\mathrm{d}t} = f(\boldsymbol{x}(t), \boldsymbol{\theta}),$$

where $\boldsymbol{x} : \mathbb{R}_{\geq 0} \to \mathbb{R}^X$ is the trajectory of the X-dimensional state of the system over time, $\boldsymbol{\theta} \in \Theta$ is a set of parameters of the model and $f : \mathbb{R}^X \times \Theta \to \mathbb{R}^X$ is a vector-valued function describing the dynamics of the system.

Given an initial state $\boldsymbol{x}(0) = \boldsymbol{x}_0$ for the system we can (numerically) solve the *initial value problem* (IVP) to compute the value of state $\boldsymbol{x}(t)$ at times $t > 0$ for fixed values of the parameters $\boldsymbol{\theta}$. A numerical solver for the IVP implicitly defines a family of maps $F_t : \mathbb{R}^X \times \Theta \to \mathbb{R}^X$ such that

$$\boldsymbol{x}(t) = F_t(\boldsymbol{x}_0, \boldsymbol{\theta}).$$

6.4 Stochastic evolution models

For systems where the dynamics are intrinsically noisy or for which we have only a partial description of the dynamics, an alternative class of models is those described by a system of autonomous SDEs

$$\mathrm{d}\boldsymbol{x}(t) = a(\boldsymbol{x}(t), \boldsymbol{\theta})\mathrm{d}t + B(\boldsymbol{x}(t), \boldsymbol{\theta})\mathrm{d}\boldsymbol{w}(t)$$

where $\boldsymbol{x} : \mathbb{R}_{\geq 0} \to \mathbb{R}^X$ is a vector-valued stochastic process describing the state of the system over time, $\boldsymbol{\theta} \in \Theta$ is a set of (unknown) parameters of the model, $a : \mathbb{R}^X \times \Theta \to \mathbb{R}^X$ is a function describing the (deterministic) drift term in the dynamics, $\boldsymbol{w} : \mathbb{R}_{\geq 0} \to \mathbb{R}^W$ is a W-dimensional vector of standard Wiener processes and $B : \mathbb{R}^X \times \Theta \to \mathbb{R}^{X \times W}$ is a diffusion coefficient defining how the Wiener noise enters the dynamics.

Numerical integrators for SDEs can be used to (approximately) simulate realisations of the process. Specifically most numerical schemes can be formulated as a family of forward operators $F_h : \mathbb{R}^X \times \Theta \times \mathbb{R}^V \to \mathbb{R}^X$ such that for a small time step $h \geq 0$ and a V-dimensional vector of standard normal variates $\boldsymbol{v} \sim \mathsf{Normal}(\mathbf{0}, \mathbb{I})$, then $F_h(\boldsymbol{x}(t), \boldsymbol{\theta}, \boldsymbol{v})$ is an approximate sample from the conditional distribution on $\boldsymbol{x}(t+h)$ given the current state $\boldsymbol{x}(t)$ at time t and parameters $\boldsymbol{\theta}$. The simplest and most commonly used example of such a numerical integrator is the Euler-Maruyama scheme for which

$$F_h(\boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{v}) := \boldsymbol{x} + ha(\boldsymbol{x}, \boldsymbol{\theta}) + \sqrt{h}B(\boldsymbol{x}, \boldsymbol{\theta})\boldsymbol{v}$$

and the vector of normal variates $\boldsymbol{v}$ has dimension $V = W$.

When $W = X$ and $B(\boldsymbol{x}(t), \boldsymbol{\theta})B(\boldsymbol{x}(t), \boldsymbol{\theta})^\mathsf{T}$ is strictly positive definite, then the Euler-Maruyama scheme gives an approximation of the conditional distribution on $\boldsymbol{x}(t+h)$ given $\boldsymbol{x}(t)$ and $\boldsymbol{\theta}$ for small $h > 0$ as

$$\boldsymbol{x}(t+h) \,|\, (\boldsymbol{x}(t), \boldsymbol{\theta}) \sim \mathsf{Normal}\big(\boldsymbol{x}(t) + ha(\boldsymbol{x}(t), \boldsymbol{\theta}), hB(\boldsymbol{x}(t), \boldsymbol{\theta})B(\boldsymbol{x}(t), \boldsymbol{\theta})^\mathsf{T}\big).$$

An alternative approach to approximating the solutions to the SDEs is to decompose the system into an ODE system and zero-drift SDE system

$$\begin{aligned} \frac{\mathrm{d}\boldsymbol{x}(t)}{\mathrm{d}t} &= a(\boldsymbol{x}(t), \boldsymbol{\theta}) \qquad (i); \\ \mathrm{d}\boldsymbol{x}(t) &= B(\boldsymbol{x}(t), \boldsymbol{\theta})\mathrm{d}\boldsymbol{w}(t) \qquad (ii); \end{aligned}$$

and use a Strang-splitting like approach which alternates steps numerical solving (i) and (ii). If we define $A_h(\boldsymbol{x}, \boldsymbol{\theta})$ as a (numerical) solution $\boldsymbol{x}(h)$ to the IVP for the ODE system (i) with initial state $\boldsymbol{x}(0) = \boldsymbol{x}$ then assuming a simple Euler-Maruyama scheme to solve (ii), we have that a Gaussian approximation to the conditional (transition) distribution is

$$\boldsymbol{x}(t+h) \,|\, (\boldsymbol{x}(t), \boldsymbol{\theta}) \sim \mathsf{Normal}\big(A_h(\boldsymbol{x}(t), \boldsymbol{\theta}), hB(\boldsymbol{x}(t), \boldsymbol{\theta})B(\boldsymbol{x}(t), \boldsymbol{\theta})^\mathsf{T}\big).$$

This allows for example using an ODE solver with adaptive time-stepping and/or implicit steps suitable for stiff systems for implementing A_h to maintain numerical stability for larger h.

6.5 Observation model

We assume we noisily observe the system state at N times $0 \leq t_1 < t_2 \ldots t_{N-1} < t_N$. For simplicity we will assume the observations are subject to additive Gaussian noise with

$$\boldsymbol{y}_n \sim \mathsf{Normal}\left(H(\boldsymbol{x}(t_n), \boldsymbol{\theta}), R(\boldsymbol{\theta})\right), \quad n \in 1{:}N,$$

where $H : \mathbb{R}^X \times \Theta \to \mathbb{R}^Y$ is the (possibly non-linear) observation operator and $R : \Theta \to \mathbb{R}^{Y \times Y}$ is a positive-definite valued function defining the observation noise covariance.

6.6 Inference in deterministic evolution models

For deterministic evolution models, the full state trajectories are entirely determined by the initial state of the system $\boldsymbol{x}_0$ and parameters $\boldsymbol{\theta}$. Given a joint prior distribution π_0 on the initial state $\boldsymbol{x}_0$ and parameters $\boldsymbol{\theta}$ then our overall generative model for the observations $\boldsymbol{y}_{1:N}$ is

$$(\boldsymbol{x}_0, \boldsymbol{\theta}) \sim \pi_0; \qquad \boldsymbol{y}_n \sim \mathsf{Normal}(H(F_{t_n}(\boldsymbol{x}_0, \boldsymbol{\theta}), \boldsymbol{\theta}), R(\boldsymbol{\theta})) \quad n \in 1{:}N,$$

and the posterior distribution π on the initial state $\boldsymbol{x}_0$ and parameters $\boldsymbol{\theta}$ is

$$\pi(\mathrm{d}\boldsymbol{x}_0, \mathrm{d}\boldsymbol{\theta} \mid \boldsymbol{y}_{1:N}) \propto$$
$$|R(\boldsymbol{\theta})|^{-\frac{N}{2}} \exp\Big(-\frac{1}{2}\sum_{n=1}^{N}\big(\boldsymbol{y}_n - H(F_{t_n}(\boldsymbol{x}_0, \boldsymbol{\theta}), \boldsymbol{\theta})\big)^{\mathsf{T}} R(\boldsymbol{\theta})^{-1}\big(\boldsymbol{y}_n - H(F_{t_n}(\boldsymbol{x}_0, \boldsymbol{\theta}), \boldsymbol{\theta})\big)\Big)\pi_0(\mathrm{d}\boldsymbol{x}_0, \mathrm{d}\boldsymbol{\theta}).$$

The posterior is specified up to an unknown normalizing term (the model evidence) depending only on the observations $\boldsymbol{y}_{1:N}$. Assuming π_0 has a tractable density function, then we can evaluate an (unnormalized) density function for the posterior. This density function can be used within for example a Markov chain Monte Carlo (MCMC) method to compute a sequence of $(\boldsymbol{x}_0, \boldsymbol{\theta})$ samples which are approximately distributed according to the posterior distribution.

The posterior (predictive) distribution on all states $\boldsymbol{x}(t)$ for $t > 0$ is the pushforward of the posterior distribution on $(\boldsymbol{x}_0, \boldsymbol{\theta})$ under F_t. If we have samples from the posterior distribution on $(\boldsymbol{x}_0, \boldsymbol{\theta})$ given $\boldsymbol{y}_{1:N}$ we can therefore just map these through F_t to get posterior samples of $\boldsymbol{x}(t)$ for any t, both when interpolating between and extrapolating outside the observation times.

While having the full state trajectories determined just by the initial state and parameters both results in a relatively low $(X + \mathsf{dim}(\Theta))$ dimensional inference problem and simplifies making predictions, the rigidity of having the state evolution fully determined by a low-dimensional state can also be problematic in the case of model mismatch - that is a discrepancy between the generative process which gave rise to the observed data and the model being fit to the data. When there is model mismatch there may be no configurations of $(\boldsymbol{x}_0, \boldsymbol{\theta})$ which lead to states at the observed times $(\boldsymbol{x}(t_n))_{n=1}^N$ which are simultaneously consistent with all of the observations $\boldsymbol{y}_{1:N}$. This can potentially lead to a posterior with multiple spurious modes corresponding to configurations consistent with different partial subsets of the observations.

6.7 Filtering in stochastic evolution models

For stochastic evolution models, for now assuming fixed parameters $\boldsymbol{\theta}$, the state trajectories $\boldsymbol{x}(t)$ are no longer deterministic given the initial state $\boldsymbol{x}_0$ of the system. This means we need to infer a posterior distribution on a state trajectory rather than an individual state, resulting in a much higher-dimensional latent space and more challenging inference problem. Conversely, the additional flexibility in the probabilistic description of the dynamics can mean that such models are more robust to mismatch with the underlying generative process that gave rise to the observed data, with the Wiener noise processes able to account for non-modelled aspects in the dynamics.

The transition kernel associated with the exact solution of the SDE defines a conditional distribution on $\boldsymbol{x}(t')$ given $\boldsymbol{x}(t)$ for any $t' \geq t$. In general we cannot sample from nor evaluate a density for the conditional distribution, but we can use numerical schemes to approximate it as described previously.

If we define $\boldsymbol{x}_n = \boldsymbol{x}(t_n)$ for $n \in 1{:}N$ the overall generative model for a stochastic evolution described a system of SDEs with fixed parameters $\boldsymbol{\theta}$ can be formulated as a *state space model* (SSM) defined by

$$\boldsymbol{x}_0 \sim \pi_0; \qquad \boldsymbol{x}_n \sim M_n(\cdot|\boldsymbol{x}_{n-1}), \quad n \in 1{:}N; \qquad \boldsymbol{y}_n \sim G_n(\cdot \mid \boldsymbol{x}_n), \quad n \in 1{:}N;$$

for suitable Markov transition kernels $M_{1:N}$, observation distributions $G_{1:N}$ and initial state (prior) distribution π_0. *Filtering* in SSMs is a process of recursively computing the posterior distribution on a state $\boldsymbol{x}_n$ at time step n given the observations at all time steps up to and including n that is $\boldsymbol{y}_{1:n}$. We define two sequences of distributions: the *filtering distributions* π_n for $n \in 1{:}N$ are the posterior distributions on $\boldsymbol{x}_n$ given $\boldsymbol{y}_{1:n}$ while the *predictive distributions* $\vec{\pi}_n$ for $n \in 1{:}N$ are the posterior distributions on $\boldsymbol{x}_n$ given $\boldsymbol{y}_{1:n-1}$. We then have that two sequences of distributions can be recursively defined by

$$\vec{\pi}_n(\mathrm{d}\boldsymbol{x}_n \mid \boldsymbol{y}_{1:n-1}) = \int_{\mathbb{R}^X} M_n(\mathrm{d}\boldsymbol{x}_n \mid \boldsymbol{x}_{n-1}) \pi_{n-1}(\mathrm{d}\boldsymbol{x}_{n-1} \mid \boldsymbol{y}_{1:n-1}), \quad n \in 1{:}N;$$

with the notational convention that $\vec{\pi}_1(\mathrm{d}\boldsymbol{x}_1 \mid \boldsymbol{y}_{1:0}) := \vec{\pi}_1(\mathrm{d}\boldsymbol{x}_1)$ and $\pi_0(\mathrm{d}\boldsymbol{x}_0 \mid \boldsymbol{y}_{1:0}) := \pi_0(\mathrm{d}\boldsymbol{x}_0)$, and

$$\pi_n(\mathrm{d}\boldsymbol{x}_n \mid \boldsymbol{y}_{1:n}) = \frac{g_n(\boldsymbol{y}_n \mid \boldsymbol{x}_n)\vec{\pi}_n(\mathrm{d}\boldsymbol{x}_n \mid \boldsymbol{y}_{1:n-1})}{\int_{\mathbb{R}^X} g_n(\boldsymbol{y}_n \mid \boldsymbol{x})\vec{\pi}_n(\mathrm{d}\boldsymbol{x} \mid \boldsymbol{y}_{1:n-1})}, \quad n \in 1{:}N;$$

where g_n is a density function for the observation distribution G_n.

In general the integrals in these two recursions will not have analytic solutions and we will need to use methods which approximate them. One such class of algorithms is *particle filtering* (Gordon, Salmond, and Smith 1993) which uses Monte Carlo approximations to the two recursive update steps, and give asymptotically (in the limit of the number of Monte Carlo samples going to infinity) exact estimates of expectations with respect to the filtering distributions.

A particle filter constructs a sequence of *ensembles* of *particles*, $\boldsymbol{x}_n^{(1:P)} := (\boldsymbol{x}_n^{(p)})_{p=1}^P$ for time indices $n \in 0{:}N$ (where P is the *ensemble size*, an integer greater than zero), corresponding to empirical approximations of the filtering distributions,

$$\pi_n(\mathrm{d}\boldsymbol{x}_n \mid \boldsymbol{y}_{1:n}) \approx \frac{1}{P}\sum_{p=1} \delta_{\boldsymbol{x}_n^{(p)}}(\mathrm{d}\boldsymbol{x}_n), \quad n \in 0{:}N.$$

This sequence of *filtering ensembles* is initialised by sampling the particles independently from the initial state distribution π_0, that is

$$\boldsymbol{x}_0^{(p)} \sim \pi_0, \quad p \in 1{:}P.$$

For time indices $n \geq 1$, an ensemble of *particle proposals* $\tilde{\boldsymbol{x}}_n^{(1:P)}$ are computed from the previous filtering ensemble $\boldsymbol{x}_{n-1}^{(1:P)}$, as

$$\tilde{\boldsymbol{x}}_n^{(p)} \sim Q_n(\cdot \mid \boldsymbol{x}_{n-1}^{(p)}, \boldsymbol{y}_n), \quad p \in 1{:}P$$

where Q_n is the *proposal distribution* for time index n, for which various choices are discussed below. These particle proposals can be considered as an importance weighted empirical approximation to the filtering distribution π_n

$$\pi_n(\mathrm{d}\boldsymbol{x}_n \mid \boldsymbol{y}_{1:n}) \approx \frac{\sum_{r=1}^P w_n(\vec{\boldsymbol{x}}_n^{(r)}, \boldsymbol{x}_{n-1}^{(r)}, \boldsymbol{y}_n)\, \delta_{\vec{\boldsymbol{x}}_n^{(r)}}(\mathrm{d}\boldsymbol{x}_n)}{\sum_{s=1}^P w_n(\vec{\boldsymbol{x}}_n^{(s)}, \boldsymbol{x}_{n-1}^{(s)}, \boldsymbol{y}_n)},$$

with importance weighting function

$$w_n(\boldsymbol{x}_n, \boldsymbol{x}_{n-1}, \boldsymbol{y}_n) = \frac{m_n(\boldsymbol{x}_n \mid \boldsymbol{x}_{n-1})\, g_n(\boldsymbol{y}_n \mid \boldsymbol{x}_n)}{q_n(\boldsymbol{x}_n \mid \boldsymbol{x}_{n-1}, \boldsymbol{y}_n)},$$

where m_n is a density function for the Markov transition kernel M_n and q_n is a density function for the proposal distribution Q_n. A new filtering ensemble $\boldsymbol{x}_{n-1}^{(1:P)}$ is computed from the weighted

particle proposal ensemble by *resampling*, that is independently sampling from the weighted empirical approximation to the filtering distribution π_n,

$$\boldsymbol{x}_n^{(p)} \sim \frac{\sum_{r=1}^{P} w_n(\vec{\boldsymbol{x}}_n^{(r)}, \boldsymbol{x}_{n-1}^{(r)}, \boldsymbol{y}_n)\, \delta_{\vec{\boldsymbol{x}}_n^{(r)}}(\cdot)}{\sum_{s=1}^{P} w_n(\vec{\boldsymbol{x}}_n^{(s)}, \boldsymbol{x}_{n-1}^{(s)}, \boldsymbol{y}_n)}, \quad p \in 1{:}P.$$

These proposal and resampling steps can then be iteratively applied for each $n \in 1{:}N$ to compute the full sequence of filtering ensembles.

The choice of proposal distributions $Q_{1:N}$ are a key factor in determining the statistical efficiency of a particle filter. A common choice is the *bootstrap proposal*

$$Q_n(\mathrm{d}\boldsymbol{x}_n \mid \boldsymbol{x}_{n-1}, \boldsymbol{y}_n) = M_n(\mathrm{d}\boldsymbol{x}_n \mid \boldsymbol{x}_{n-1}),$$

that is, sampling the proposals directly from the Markov transition kernels of the SSM. This choice of proposal distributions leads to a particularly simple weighting function

$$w_n(\boldsymbol{x}_n, \boldsymbol{x}_{n-1}, \boldsymbol{y}_n) = g_n(\boldsymbol{y}_n \mid \boldsymbol{x}_n),$$

which depends only on the proposed particle and observation values, and does not require access to an explicit density function m_n for the Markov transition kernel M_n. However, by ignoring the observations when proposing new values for the particles and simply sampling from the prior SSM, the proposals may end up in regions of the state space where $g_n(\boldsymbol{y}_n \mid \boldsymbol{x}_n)$ is small (that is the proposal are inconsistent with the observed data $\boldsymbol{y}_n$), corresponding to being in the tails of the filtering distribution π_n. In such cases, the particle filter will typically suffer from *weight degeneracy* whereby due to the high variance of the importance weights, once normalized most particles have weights very close to zero other than a single particle with weight close to one, resulting in an ensemble of effectively only one sample. Particle filters suffering from weight degeneracy will tend to give poor filtering distribution estimates.

To minimize the tendency towards weight degeneracy, an alternative choice of proposal distribution is to use the conditional distribution on the state at the next time index given *both* the previous state *and* current observations, that is

$$Q_n(\mathrm{d}\boldsymbol{x}_n \mid \boldsymbol{x}_{n-1}, \boldsymbol{y}_n) = \frac{M_n(\mathrm{d}\boldsymbol{x}_n \mid \boldsymbol{x}_{n-1})\, g_n(\boldsymbol{y}_n \mid \boldsymbol{x}_n)}{\int_{\mathbb{R}^X} M_n(\mathrm{d}\boldsymbol{x} \mid \boldsymbol{x}_{n-1})\, g_n(\boldsymbol{y}_n \mid \boldsymbol{x})}.$$

In this case the weighting function simplifies to

$$w_n(\boldsymbol{x}_n, \boldsymbol{x}_{n-1}, \boldsymbol{y}_n) = \int_{\mathbb{R}^X} M_n(\mathrm{d}\boldsymbol{x} \mid \boldsymbol{x}_{n-1})\, g_n(\boldsymbol{y}_n \mid \boldsymbol{x}),$$

which does not depend on the value of the sampled proposals. By eliminating the dependence on the stochastically sampled proposals, this choice of proposal minimises the variance of the importance weights associated with the proposed particles, and is therefore sometimes termed the *locally optimal proposal*, in terms of minimising the variance of the importance weights).

While attractive from a statistical efficiency standpoint, the locally optimal proposal is only applicable to a subset of models for which the proposal distribution and weighting function have closed form solutions. One class of SSMs for which using locally optimal proposal is tractable are those with Gaussian Markov transition kernels $M_{1:N}$ and observation distributions $G_{1:N}$ where additionally the mean of the observation distribution is a linear function of the state (Doucet, Godsill, and Andrieu 2000).

Returning to the specific case of an SDE model as described previously, if we assume the inter-observation intervals $t_n - t_{n-1}, n \in 1{:}N$ are sufficiently small, then the Markov transition kernels for the state updates can be approximated as Gaussian distributions using the splitting numerical scheme described above as

$$M_n(\cdot \mid \boldsymbol{x}_{n-1}) = \mathsf{Normal}(A_{t_n - t_{n-1}}(\boldsymbol{x}_{n-1}, \boldsymbol{\theta}), C_n(\boldsymbol{x}_{n-1}, \boldsymbol{\theta})), \quad n \in 1{:}N,$$

where $C_n(\boldsymbol{x}_{n-1}, \boldsymbol{\theta}) := (t_n - t_{n-1})B(\boldsymbol{x}_{n-1}, \boldsymbol{\theta})B(\boldsymbol{x}_{n-1}, \boldsymbol{\theta})^\mathsf{T}$. We also have that the observation distributions are Gaussian

$$G_n(\cdot \mid \boldsymbol{x}_n) = \mathsf{Normal}(H(\boldsymbol{x}_n, \boldsymbol{\theta}), R(\boldsymbol{\theta})), \quad n \in 1{:}N.$$

For linear-in-the-state observation operators $H(\boldsymbol{x}_n, \boldsymbol{\theta}) = H(\boldsymbol{\theta})\boldsymbol{x}_n$, then the locally optimal proposal for this SSM is tractable, specifically

$$\begin{aligned}
&Q_n(\cdot \mid \boldsymbol{x}_{n-1}, \boldsymbol{y}_n) = \mathsf{Normal}\big(\\
&\quad A_{t_n - t_{n-1}}(\boldsymbol{x}_{n-1}) + C_n(\boldsymbol{x}_{n-1})H^\mathsf{T}(HC_n(\boldsymbol{x}_{n-1})H^\mathsf{T} + R)^{-1}(\boldsymbol{y}_n - HA_{t_n - t_{n-1}}(\boldsymbol{x}_{n-1})), \\
&\quad C_n(\boldsymbol{x}_{n-1}) - C_n(\boldsymbol{x}_{n-1})H^\mathsf{T}(HC_n(\boldsymbol{x}_{n-1})H^\mathsf{T} + R)^{-1}HC_n(\boldsymbol{x}_{n-1}) \\
&\big)
\end{aligned}$$

where we have elided the dependence of terms on $\boldsymbol{\theta}$ for compactness.

6.8 Test model

As a test case we consider a model of magnetohydrodynamic instability coupled to a slow dissipative background evolution (Arter 2012). The resulting *axissymmetric non-axissymmetric extended* (ANAET) model is described by the system of ODEs

$$\ddot{a} = -\gamma_r a - (\mu_1 + \mu_2 b)a^3 - \mu_6 a^6 \dot{a}, \qquad \dot{b} = \nu_1 - \nu_2 b^2 - (\delta_0 + \delta_1 b)a^2,$$

where a is a non-axissymmetric ideal mode coupled to an axissymmetric mode b, and $(\gamma_r, \mu_1, \mu_2, \mu_6, \nu_1, \nu_2, \delta_0, \delta_1)$ are free parameters. Defining $\boldsymbol{x} = (a, \dot{a}, b)$, these equations can be written as a first order ODE system

$$\frac{\mathrm{d}\boldsymbol{x}(t)}{\mathrm{d}t} = \begin{pmatrix} x_2(t) \\ -\gamma_r x_1(t) - (\mu_1 + \mu_2 x_3(t))x_1(t)^3 - \mu_6 x_1(t)^6 x_2(t) \\ \nu_1 - \nu_2 x_3(t)^2 - (\delta_0 + \delta_1 b)x_1(t)^2 \end{pmatrix}.$$

We also consider a simple stochastic evolution variant of this model defined by the SDE system

$$\mathrm{d}\boldsymbol{x}(t) = \begin{pmatrix} x_2(t) \\ -\gamma_r x_1(t) - (\mu_1 + \mu_2 x_3(t))x_1(t)^3 - \mu_6 x_1(t)^6 x_2(t) \\ \nu_1 - \nu_2 x_3(t)^2 - (\delta_0 + \delta_1 b)x_1(t)^2 \end{pmatrix} \mathrm{d}t + \beta \mathrm{d}\boldsymbol{w}(t),$$

with $W = X = 3$, with Wiener noise of magnitude controlled by a shared scalar parameter $\beta > 0$ introduced in to each of the three state components.

We assume only observations of first (a) state component and independent Gaussian observation noise, that is an observation model

$$y_n \sim \mathsf{Normal}(x_1(t_n), \sigma^2), \quad n \in 1{:}N.$$

6.8.1 Inference with PyMC

As a first example we consider jointly inferring the initial state $\boldsymbol{x}_0 = (a_0, \dot{a}_0, b_0)$ and parameters $\boldsymbol{\theta} = (\gamma_r, \mu_1, \mu_2, \mu_6, \nu_1, \nu_2, \delta_0, \delta_1, \sigma)$ of the original (deterministic) ANAET test model, given simulated observed data generated from the model. We generate simulated observed data $y_{1:N}$ for $N = 100$ times $t_n = n, n \in 1{:}100$ (see Figure 16) using the parameter and initial state values defined in Table 1.

Table 1: True values of model variables used to generate simulated data and prior distributions.

Variable	True value	Prior distribution
a_0	1	$\mathsf{Normal}(0, 1)$
$\dot{a}_0$	2.5	$\mathsf{Normal}(0, 1)$
b_0	0.01	$\mathsf{Normal}(0, 1)$
γ_r	1	$\mathsf{Normal}(0, 1)$

Variable	True value	Prior distribution
μ_1	0	$\mathsf{Normal}(0, 1)$
μ_2	-2	$\mathsf{Normal}(0, 1)$
μ_6	0.001	$\mathsf{Normal}(0, 1)$
ν_1	0.005	$\mathsf{Normal}(0, 1)$
ν_2	0.0001	$\mathsf{Normal}(0, 1)$
δ_0	0.0001	$\mathsf{Normal}(0, 1)$
δ_1	0	$\mathsf{Normal}(0, 1)$
σ	0.5	$\mathsf{HalfNormal}(1)$

We assign independent weakly informative standard normal priors to all variables other than than the observation noise scale parameter σ to which we give a half-normal prior to reflect the constraint that the parameter is non-negative.

We use a gradient-based *Hamiltonian Monte Carlo* (HMC) (Duane et al. 1987; Neal et al. 2011) MCMC method to generate approximate samples from the posterior distribution on the initial state and parameters $(\boldsymbol{x}_0, \boldsymbol{\theta})$ given the simulated observations $\boldsymbol{y}_{1:N}$. Specifically we use the implementation of the *No-U-Turn-Sampler* (NUTS) (Hoffman, Gelman, et al. 2014) in PyMC (Wiecki et al. 2023), which uses dual-averaging algorithm to adaptively set the step size and mass matrix during a warm-up sampling phase and uses a heuristic to dynamically set the number of steps in the simulated Hamiltonian trajectories in each proposed move.

To solve the ODE system we use a backwards differentiation formula (BDF) solver with adaptive time stepping, and propagate derivatives using an adjoint sensitivity analysis approach. Specifically we use the CVODES solver in the SUNDIALS suite (Hindmarsh et al. 2005; Gardner et al. 2022) via the Python package `sunode` (Seyboldt et al. 2022) which provides a PyTensor wrapper of the CVODES solver which is compatible with PyMC.

Table 2: Convergence diagnostics for MCMC output.

	ESS(bulk)	ESS(tail)	$\hat{R}$
γ_r	17	112	1.22
μ_1	13	40	1.57
μ_2	9	19	1.63
μ_6	5	31	2.16
ν_1	19	28	1.27
ν_2	6	52	1.85
δ_0	12	59	1.3
δ_1	6	37	1.88
$\mathrm{r}\sigma$	7	15	1.53
a_0	7	35	1.55
$\dot{a}_0$	10	29	1.77
b_0	8	75	1.44

We simulate four independent Markov chains in parallel to facilitate cross-chain convergence diagnostics, using 1000 adaptive warm-up iterations per chain and 1000 iterations in the main sampling phase. Simulating the chains took approximately 12 hours on a system with Intel Core i7-10610U CPU. Convergence diagnostics for the sampled chains are summarized in Table 2, specifically *effective sample size* (ESS) estimates and split-$\hat{R}$ potential scale reduction factor convergence diagnostics computed using the rank-normalization based approach described in Vehtari et al. (2021). $\hat{R}$ values greater than 1.01 are indicative of poorly converged Markov chains, which we can see is the case for all parameters here. The ESS estimates give an approximation of how many independent samples would given an equivalent variance in estimates for different summary statistics using the dependent Markov chain samples (ESS(bulk) here gives an indication of the effective sample size in the main bulk of the

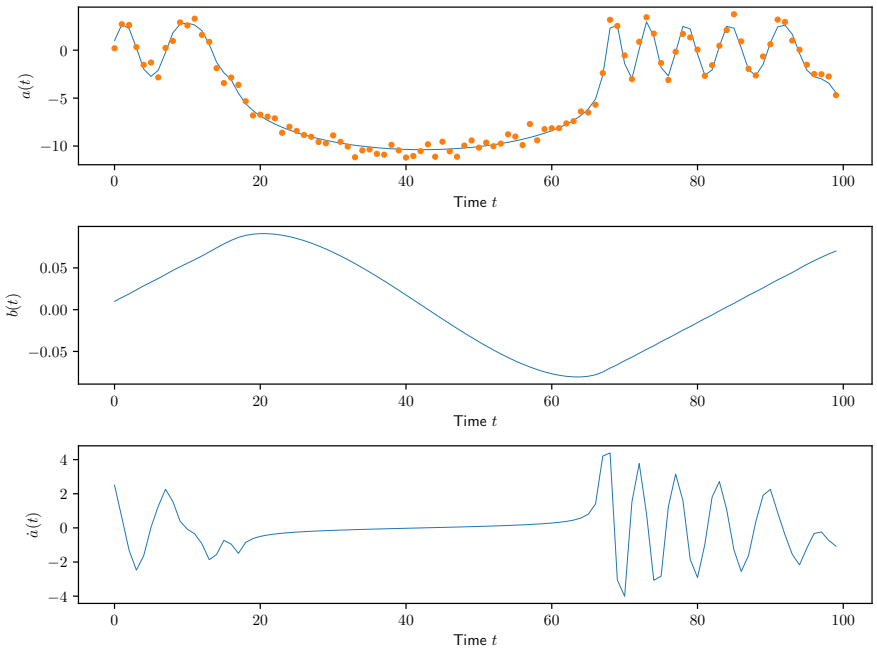


Figure 16: Simulated trajectories (blue lines) and noisy observations (orange markers) for ANAET model.

posterior and ESS(tail) in the tails). As here the number of samples Markov chain samples is 4000 and so we would expect ESS values of close to 4000 if the chain samples were close to independent, we can see the low ESS estimates here indicate chains which will give poor posterior estimates.

Figure 17 shows the estimates univariate and pairwise posterior marginal distributions for all (pairs of) variables in blue with the prior marginals shown in blue and true values used to generated the data shown in orange. Given the convergence issues with the Markov chains the posterior estimates here are likely to be very poor and so should be treated cautiously. The 'roughness' of the kernel density estimates of the posterior are likely a result of the high correlation between chain samples and the multimodality observed in the univariate marginals may also be an artefact of the poor chain convergence rather than being reflective of the true posterior.

We see that for most parameters the variable values used to generate data lie within bulk of the corresponding estimated posterior marginals, as would be expected for for calibrated inference estimates (Cockayne et al. 2022), however this is not the case for a few variables, most notably the observation noise scale parameter σ, with the posterior concentrating around a much higher value (~ 4.5) that the true value used to generate the data of 0.5. MCMC methods will often struggle to sample from posteriors where observations are highly informative (Au, Graham, and Thiery 2020), with a common failure mode in this case being that the sampler gets stuck in a region of space in which the observation noise scale parameter is large and the signal in the observed data is attributed to noise. The signal-to-noise ratio in the simulated observed data is relatively high here so this a potentially a source of some of the convergence issues. MCMC methods such as that described in Au, Graham, and Thiery (2020) which exploit more information about the geometry of the posterior distribution may help ameliorate such convergence issues, albeit at the cost of requiring additional implementation effort to compute the necessary higher order derivatives of the model functions.

Another issue likely contributing to the poor convergence of the MCMC chains here is that the adaptive ODE solver is failing to converge for many of the $(\boldsymbol{x}_0, \boldsymbol{\theta})$ values IVP solves are run for while simulating the Markov chains. This is likely to be due to the solver being numerically unstable in some regions of the latent space. During each Markov chain transition, the HMC based method being used here evaluates the posterior density and its gradients at a sequence of points along a simulated Hamiltonian dynamics trajectory in the latent space. Evaluating the posterior density and its gradient requires solving the IVP and if the solver fails to converge at any point in the trajectory the sampler is forced to fallback to a 'rejection' transition whereby it remains at the previous point in the latent

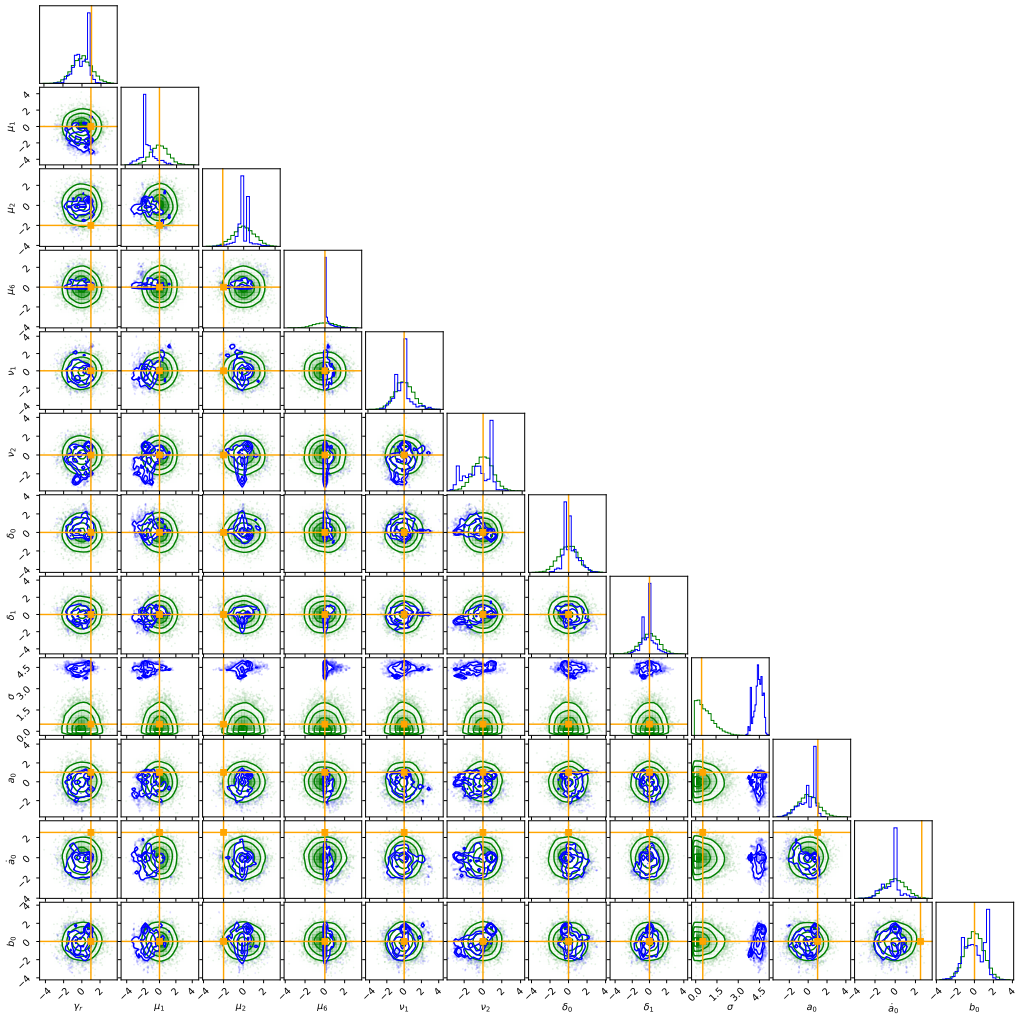


Figure 17: Estimated pairwise and univariate posterior marginals using MCMC chain samples (blue) against prior distribution (green) and true variable values used to generate data (orange).

space. If the Markov chain ends up in a point in the latent space where moving in most directions in the latent space will result in the IVP solver failing and a rejection occurring, the Markov chain can end up 'stuck' for many chain samples.

There are various approaches which may help address this issue. A key factor likely contributing to the poor performance in the test model here is the choice of prior distributions, with weakly-informative distributions placed on all variables, likely leading to significant prior mass being put on regions of the latent space which lead to physically implausible dynamics. Using domain expertise to set more informative prior distributions that avoid putting mass on regions of the latent space which are likely to lead to IVP solver errors is likely to lead to improved MCMC performance.

Another approach would be to use alternative ODE solvers more tailored to the properties of the ODE system being considered here or adjust the control parameters of the CVODES solver being currently used. Arter (2012) noted the challenge posed in accurately numerically simulating the ANAET system, and little attempt has been made here to identify suitable values for the solver control parameters such as tolerances and maximum number of steps, or explore alternative numerical solvers. Timonen et al. (2022) suggests an importance-sampling based approach for iteratively adjusting control parameters of ODE solvers when performing Bayesian inference in ODE models and accounting for the approximation error inherent to using coarser numerical approximations. The authors also note that using adaptive ODE solvers within a MCMC based approximate inference method can be problematic due to aforementioned convergence issues in regions of the latent space, and suggest a more robust alternative can be to use a non-adaptive ODE solver within their importance-sampling based framework for controlling approximation error.

A complementary approach to addressing these numerical issues to those already discussed, would be to exploit the sequential nature of the observations here by using a *sequential Monte Carlo* (SMC) sampling method (Del Moral, Doucet, and Jasra 2006). The sequence of observations $\boldsymbol{y}_{1:N}$ defines a natural *path of partial posteriors* (Dai et al. 2022) corresponding to the posterior distributions on the latent variables $(\boldsymbol{x}_0, \boldsymbol{\theta})$ given observations $\boldsymbol{y}_{1:n}$ for $n \in 0{:}N$. These $N+1$ distributions gradually bridge from the prior distribution for $n = 0$ to the full posterior for $n = N$. SMC methods are a generalization of particle filters, and similarly involve updating an ensemble of particles, here targetting a generic sequence of distributions rather than specifically the filtering distributions for a SSM. The Markov kernels used to update each of the particles in the ensemble for a particular target distribution in the sequence, can be constructed by using a single transition of a MCMC method (such as the NUTS method used in the experiments here) which leaves the target distribution invariant.

Compared to directly applying a MCMC method SMC samplers offer a number of advantages. The updates to each particle for each intermediate distribution can be performed in parallel, making them ideal for deployment on multi-node *high performance computing* (HPC) systems or single compute nodes with multiple processing units. For a path of partial posteriors, the earlier target distributions on the path will only require solving the IVP for a subset of the observation times, significantly reducing the computational cost of these earlier updates. By updating multiple particles in parallel across the latent space, SMC samplers can also be less prone to the sticking issues due to failure of numerical solvers in some regions of the latent space, as if at least some of the particles end up in regions of the space for which the numerical solver is stable, the updates to these particles will succeed and particles for which updates are rejected due to solver fails are likely to be subsequently replaced by these successfully updated particles in the resampling steps between particle updates. Similarly SMC samplers typically exhibit much more robust performance for posterior distributions exhibiting multimodality (Dai et al. 2022).

An alternative approach to addressing both the numerical issues involved in repeatedly solving IVPs, and high computational cost inherent to this, is to a *gradient matching* based inference approach which fit a *Gaussian process* (GP) to the unobserved state trajectories and rather than solving the IVP match the time derivatives of the GP model to the values of the time derivatives arising from the vector field function f defining the ODE model (Calderhead, Girolami, and Lawrence 2008; Dondelinger et al. 2013). By avoiding the necessity of solving the IVP to evaluate the posterior density used in these approaches, we both sidestep the numerical issues involved with ensuring stability of the solver across the latent space, and significantly reduce the computational cost of inference. The cost is however that we no longer generate samples from the true posterior of interest but instead from a proxy distribution which may be less informative about some latent variables in some cases, and which the faithfulness of which to the true posterior will depend on how well the GPs used to model the state trajectories reflect the true trajectories of the ODE system.

Instead of using GPs as proxies for the unobserved state trajectories, an alternative is to use a GP as an emulator for the log-likelihood. Specifically within the context of HMC methods, a GP emulator for the log-likelihood can be used in evaluating the (log) posterior density values gradients required to simulate Hamiltonian trajectories in the latent space, to propose new a point for the chain to move to (Paun and Husmeier 2022). Using a GP emulator in place of directly evaluating the log likelihood avoids both the cost and potential for convergence issues in repeatedly solving IVPs for each point on the simulated Hamiltonian trajectory. If the probability of accepting a proposal continues to be evaluated using the true posterior density however, it can be ensured that such an MCMC method still asymptotically converges to the true posterior. Requiring only one full posterior density evaluation per Markov chain iteration both dramatically reduces the computational cost of inference and also sidesteps the need to be able to compute the derivatives of the true posterior density, often a significant implementation barrier in practice.

6.8.2 Filtering with ParticleDA.jl

As a second example we consider using a particle filter to estimate the state trajectories $\boldsymbol{x}_{1:N}$ (with $\boldsymbol{x}_n = \boldsymbol{x}(t_n)$, $t_n = n$ and $N = 100$ as previously) of the stochastic variant of the ANAET model, given simulated observed data $\boldsymbol{y}_{1:N}$ generated from the model. Here we fix the parameters to the values

defined in Table 1, with additionally the diffusion coefficient parameter controlling the scale of the noise introduced into the dynamics set to $\beta = 0.02$. We generate a set of simulated observed data from the SSM using a fixed initial state with components as defined in Table 1 when simulating the observations, but when filtering use an initial state distribution with $a_0 \sim \mathsf{Normal}(0, 1)$, $\dot{a}_0 \sim \mathsf{Normal}(2, 1)$ and $b_0 \sim \mathsf{Normal}(0, 1)$. Our SSM formulation uses a Gaussian approximation to the Markov transition kernels based on a splitting numerical scheme which uses an adaptive ODE solver to solve for the deterministic component of the dynamics and Euler-Maruyama discretisation for the driving Wiener noise processes.

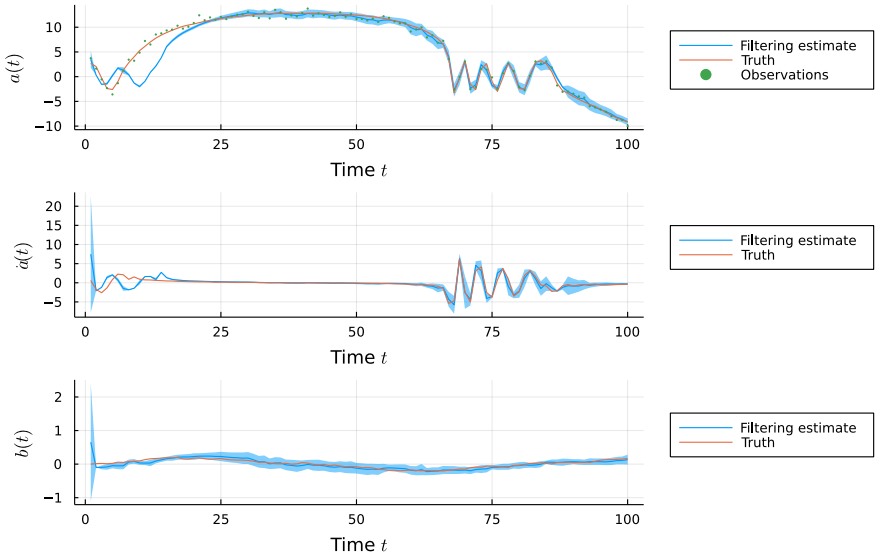


Figure 18: True state trajectories (orange), observed data (green markers) and filtering estimates (blue) for stochastic ANAET model using particle filter with $P = 125$ and locally optimal proposals.

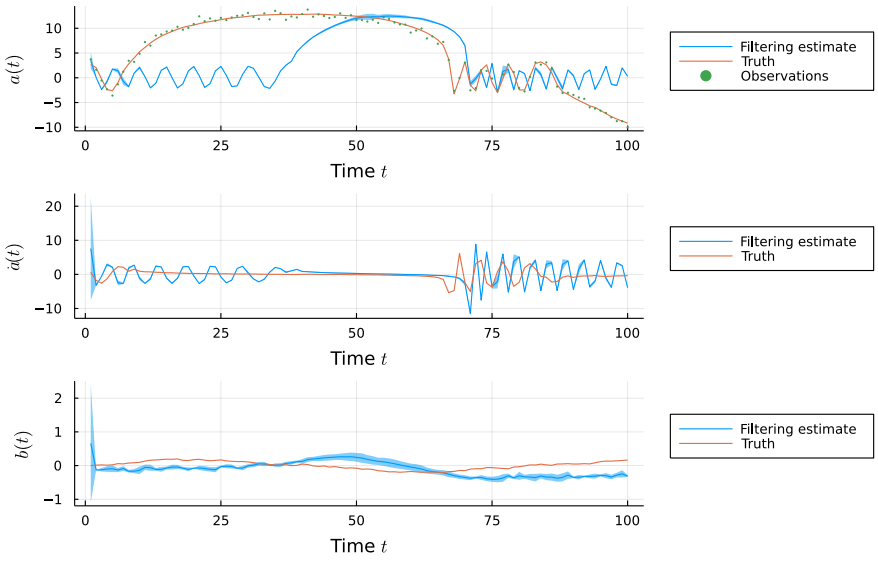


Figure 19: True state trajectories (orange), observed data (green markers) and filtering estimates (blue) for stochastic ANAET model using particle filter with $P = 125$ and bootstrap proposals.

We use the particle filter implementation in the Julia package `ParticleDA.jl` (Giles et al. 2023) to compute particle approximations to the filtering distributions $\pi_{1:N}$, using its implementations of both the bootstrap and locally optimal proposals. Figures 18, 20 and 22 show results for the locally optimal proposals with $P = 125$, $P = 250$ and $P = 500$ particles respectively, and Figures 19, 21 and 23 show results for the bootstrap proposals with $P = 125$, $P = 250$ and $P = 500$ particles respectively. In all cases, for the filtering estimates, the blue line shows the estimated filtering distribution mean

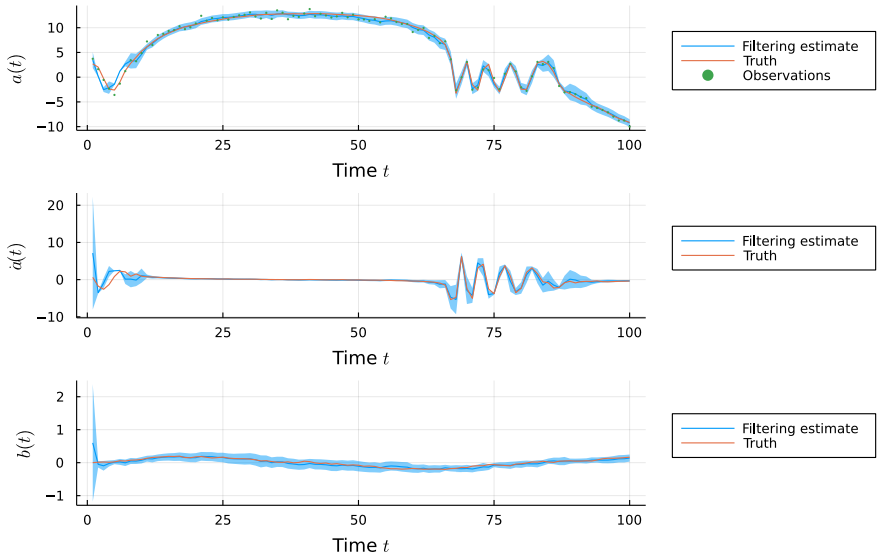


Figure 20: True state trajectories (orange), observed data (green markers) and filtering estimates (blue) for stochastic ANAET model using particle filter with $P = 250$ and locally optimal proposals.

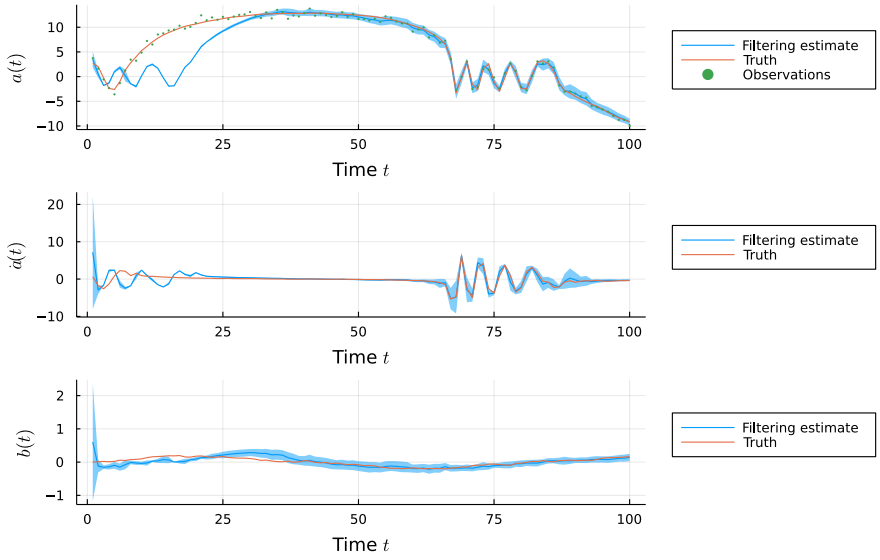


Figure 21: True state trajectories (orange), observed data (green markers) and filtering estimates (blue) for stochastic ANAET model using particle filter with $P = 250$ and bootstrap proposals.

and the blue filled region the estimated mean $\pm$ three estimated standard deviation interval.

From Figures 22 and 23 see that particle filters using both the locally optimal and bootstrap proposals appear to give reasonable estimates of the filtering distribution with $P = 500$ particles, with the true state trajectories generally within the intervals estimated to contain most of the mass of the filtering distributions as expected. For the smaller ensembles of $P = 250$ (Figures 20 and 21) and $P = 125$ (Figures 18 and 19), we see that the estimates of filters using both proposals appear to degrade at some points (particularly at earlier times in the trajectories where the is greater uncertainty due to fewer observations), but that filters using the locally optimal proposals outperform filters using the bootstrap proposals.

Here we are using a relatively coarse numerical approximation to the underlying SDE, with a single step Euler-Maruyama based scheme used to give a Gaussian approximation to Markov transitions between states at the observation times. If we had larger inter-observation intervals or wanted to use a fine time resolution when simulating the underlying SDE system, we could a discretisation scheme which uses multiple smaller steps in each inter-observation interval. There are also numerical schemes for simulating SDE systems with higher orders of convergence such as the Milstein method (Mil'shtejn

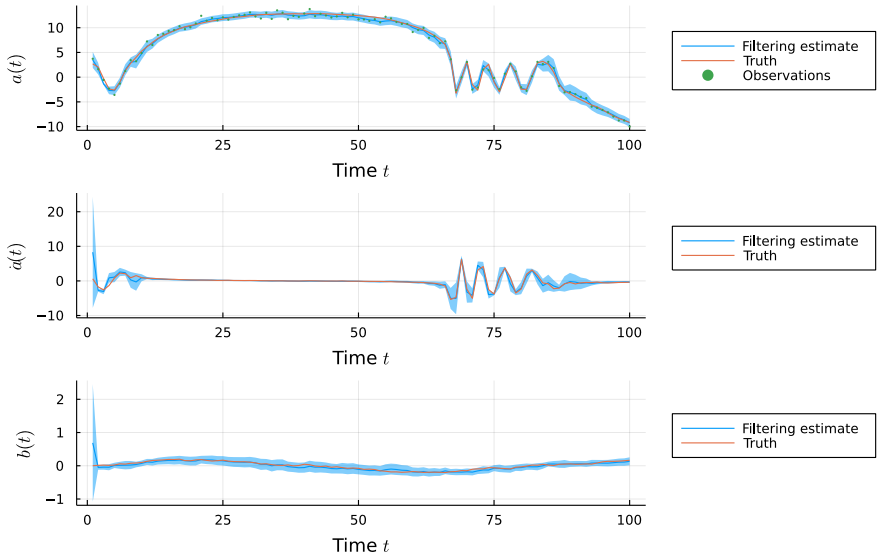


Figure 22: True state trajectories (orange), observed data (green markers) and filtering estimates (blue) for stochastic ANAET model using particle filter with $P = 500$ and locally optimal proposals.

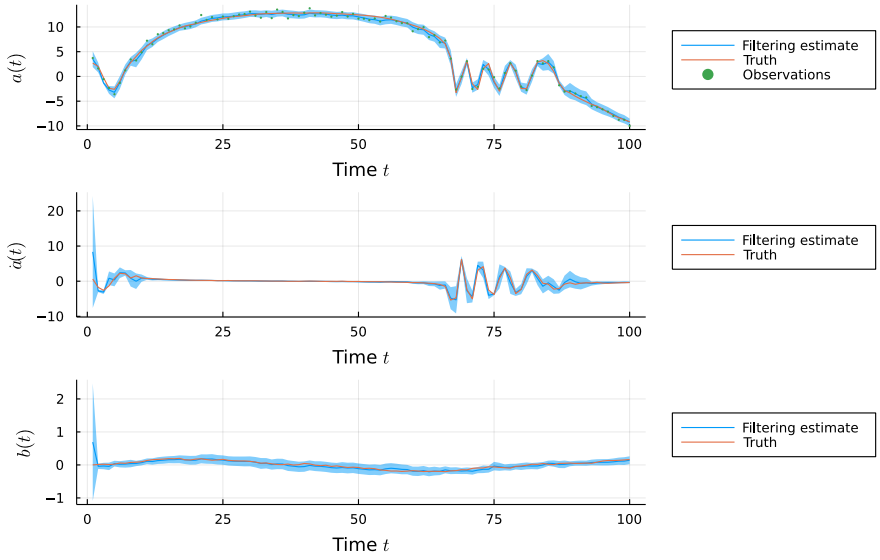


Figure 23: True state trajectories (orange), observed data (green markers) and filtering estimates (blue) for stochastic ANAET model using particle filter with $P = 500$ and bootstrap proposals. Top panel shows

1975) which could be used in-place of an Euler-Maruyama approximation. In these cases while we can sample from the resulting Markov transition kernels $M_{1:N}$, they would no longer have tractable density functions. While we could still apply a particle filter with bootstrap proposals to such SSMs, use of the locally optimal proposals would no longer be possible.

The assumption of fixed known parameters made here is limiting in practice where often we would wish to jointly infer these parameters with the state trajectories. One option for jointly inferring the parameters and state trajectories is to use particle MCMC methods (Andrieu, Doucet, and Holenstein 2010) which run a particle filter to generate proposed updates to both the state trajectories and parameters within a pseudo-marginal MCMC method (Andrieu and Roberts 2009). Nesting a particle filter within a MCMC scheme can be computationally expensive however, and pseudo-marginal MCMC schemes often suffer from a 'sticking' pathology whereby chains will show long series of rejected proposals (Murray and Graham 2016). An alternative is to use gradient-based MCMC methods directly targetting the joint posterior distribution on the state trajectories and parameters. The high dimensional latent space and complex dependencies between the latent variables can lead to a

challenging posterior geometry in such settings however MCMC schemes have been proposed which can perform efficient inference in this setting (Graham, Thiery, and Beskos 2022).

7 Software and Implementation. HPC Deployment

Deliverable D3.1 and D3.2:

D3.1: The new release 28 March 2023 of the SEAVEA toolkit featuring advanced surrogate modelling (see above new SSC method). Further releases will integrate more of the methods developed in AQUIFER.

D3.2: Release of the DA platform was achieved as *FabParticleDA*. The link to the SEAVEA UQ platform is within the SEAVEA UQ platform released 28 March 2023:
`https://github.com/djgroen/FabParticleDA/tree/master`. The test case works locally. HPC deployment at scale is part of Activity 4.

HPC deployment: As part of our efforts to support NEPTUNE, we have been working on coupling codes using MUSCLE3 as outlined in the proposal. Firstly, we replaced the MPI implementation with MUSCLE3 and verified a test case with both MUSCLE3 and MPI versions. We found that the results from the MUSCLE3 implementation matched those obtained with the MPI version.

We then conducted a test case on single desktop, which showed that MUSCLE3 was 10x slower than the MPI version. However, we expected this due to the smaller size of the test case, and communications time on MUSCLE3 took too much time comparing with the computational time. Next, we deployed MUSCLE3 on Archer2 using a larger test case and ran both an MPI job and a MUSCLE3 job on the same resources. We found that they completed almost at the same time. This gives us confidence that MUSCLE3 has the potential to efficiently couple large HPC codes. Moving forward, our next step is to evaluate the performance of MUSCLE3 with larger test cases with more complex scenarios which should help the NEPTUNE project perform HPC simulations across different scales.

References

Andrieu, Christophe, Arnaud Doucet, and Roman Holenstein (2010). "Particle Markov chain Monte Carlo methods". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 72.3, pp. 269–342.

Andrieu, Christophe and Gareth O. Roberts (2009). "The pseudo-marginal approach for efficient Monte Carlo computations". In: *The Annals of Statistics* 37.2, pp. 697–725. DOI: 10.1214/07-AOS574. URL: https://doi.org/10.1214/07-AOS574.

Arter, Wayne (2012). *Blue Sky Solutions to the Magnetohydrodynamic Trigger Problem*. UK-Germany National Astronomy Meeting NAM2012. URL: https://doi.org/10.13140/RG.2.2.35052.77449.

Au, Khai Xiang, Matthew M Graham, and Alexandre H Thiery (2020). "Manifold lifting: scaling MCMC to the vanishing noise regime". In: *arXiv preprint arXiv:2003.03950*.

Calderhead, Ben, Mark Girolami, and Neil Lawrence (2008). "Accelerating Bayesian inference over nonlinear differential equations with Gaussian processes". In: *Advances in Neural Information Processing Systems* 21.

Cockayne, Jon et al. (2022). "Testing whether a learning procedure is calibrated". In: *Journal of Machine Learning Research* 23, pp. 1–36.

Dai, Chenguang et al. (2022). "An invitation to sequential Monte Carlo samplers". In: *Journal of the American Statistical Association* 117.539, pp. 1587–1600.

Del Moral, Pierre, Arnaud Doucet, and Ajay Jasra (2006). "Sequential Monte Carlo samplers". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 68.3, pp. 411–436.

Dondelinger, Frank et al. (2013). "ODE parameter inference using adaptive gradient matching with Gaussian processes". In: *Artificial Intelligence and Statistics*. PMLR, pp. 216–228.

Doucet, Arnaud, Simon Godsill, and Christophe Andrieu (2000). "On sequential Monte Carlo sampling methods for Bayesian filtering". In: *Statistics and computing* 10, pp. 197–208.

Duane, Simon et al. (1987). "Hybrid Monte Carlo". In: *Physics letters B* 195.2, pp. 216–222.

Edeling, Wouter Nico, Richard P Dwight, and Pasquale Cinnella (2016). "Simplex-stochastic collocation method with improved scalability". In: *Journal of Computational Physics* 310, pp. 301–328.

Gardner, David J et al. (2022). "Enabling new flexibility in the SUNDIALS suite of nonlinear and differential/algebraic equation solvers". In: *ACM Transactions on Mathematical Software (TOMS)*. DOI: 10.1145/3539801.

Giles, D. et al. (2023). "ParticleDA.jl v.1.0: A real-time data assimilation software platform". In: *Geoscientific Model Development Discussions* 2023, pp. 1–20. DOI: 10.5194/gmd-2023-38. URL: https://gmd.copernicus.org/preprints/gmd-2023-38/.

Gordon, Neil J, David J Salmond, and Adrian FM Smith (1993). "Novel approach to nonlinear/non-Gaussian Bayesian state estimation". In: *IEE proceedings F (radar and signal processing)*. Vol. 140. 2. IET, pp. 107–113.

Graham, Matthew M, Alexandre H Thiery, and Alexandros Beskos (2022). "Manifold Markov chain Monte Carlo methods for Bayesian inference in diffusion models". In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 84.4, pp. 1229–1256.

Hindmarsh, Alan C et al. (2005). "SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers". In: *ACM Transactions on Mathematical Software (TOMS)* 31.3, pp. 363–396. DOI: 10.1145/1089014.1089020.

Hoffman, Matthew D, Andrew Gelman, et al. (2014). "The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo". In: *J. Mach. Learn. Res.* 15.1, pp. 1593–1623.

Mil'shtejn, GN (1975). "Approximate integration of stochastic differential equations". In: *Theory of Probability & Its Applications* 19.3, pp. 557–562.

Ming, Deyu and Serge Guillas (2021). "Linked Gaussian process emulation for systems of computer models using Matérn kernels and adaptive design". In: *SIAM/ASA Journal on Uncertainty Quantification* 9.4, pp. 1615–1642.

Ming, Deyu, Daniel Williamson, and Serge Guillas (2022). "Deep gaussian process emulation using stochastic imputation". In: *Technometrics*, pp. 1–12.

Murray, Iain and Matthew Graham (2016). "Pseudo-marginal slice sampling". In: *Artificial Intelligence and Statistics*. PMLR, pp. 911–919.

Neal, Radford M et al. (2011). "MCMC using Hamiltonian dynamics". In: *Handbook of Markov chain Monte Carlo* 2.11, p. 2.

Paun, L Mihaela and Dirk Husmeier (2022). "Emulation-accelerated Hamiltonian Monte Carlo algorithms for parameter estimation and uncertainty quantification in differential equation models". In: *Statistics and Computing* 32, pp. 1–25.

Seyboldt, Adrian et al. (Dec. 2022). *pymc-devs/sunode: v0.4.0*. URL: https://github.com/pymc-devs/sunode.

Timonen, Juho et al. (2022). "An importance sampling approach for reliable and efficient inference in Bayesian ordinary differential equation models". In: *arXiv preprint arXiv:2205.09059*.

Vehtari, Aki et al. (2021). "Rank-normalization, folding, and localization: An improved $\hat{R}$ for assessing convergence of MCMC (with discussion)". In: *Bayesian analysis* 16.2, pp. 667–718.

Wiecki, Thomas et al. (Mar. 2023). *pymc-devs/pymc: v5.1.2*. Version v5.1.2. DOI: 10.5281/zenodo.7730656. URL: https://doi.org/10.5281/zenodo.7730656.

Zhang, Ruda, Simon Mak, and David Dunson (2022). "Gaussian Process Subspace Prediction for Model Reduction". In: *SIAM Journal on Scientific Computing* 44.3, A1428–A1449.