Advanced Quantification of Uncertainties In Fusion modelling at the Exascale with model order Reduction (AQUIFER)
UKAEA NEPTUNE 2057701-TN-03
Final report

Serge Guillas
University College London

14 March 2024, revised 16 April 2024

Contents

1 Project team and deliverables

Many activities are joint with the EPSRC SEAVEA project due to synergies and cost efficiencies (e.g. *principal investigator* (PI) time, events, *high performance computing* (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, co-investigator,
- Dr. Xiao Xue, research fellow,
- Dr. Matthew Graham, senior research data scientist,
- Dr. Duncan Leggat, research software engineer,
- Dr. Tuomas Koskela, principal research software engineer,
- Yiming Yang, PhD student.

Below we present our deliverables and their locations in the report.

D1.2	final report	this report
D2.2	accelerated Bayesian calibration	see sections 11.2 and 12
D2.3	stochastic emulation for particles	see section 7
D2.4	DA approaches	see section 10
D2.5	integrated approaches multi-physics	see section 4.4
D2.6	physics-aware and deep GP	see section 5
D3.1	SEAVEA tookit and DA	see section 13
D3.2	software implementation and HPC deployment	see section 13.1
M2.1 (M6)	active subspace single models	see section 8
M2.3 (M8)	emulation of a PIC proxyapp	see section 5
M2.5 (M10)	example of emulation of coupled proxyapps	see section 4.4
M2.6 (M11)	example of benefits of physics-aware and deep GPs	see section 5,6
D4.1	scalability	see sections 11.1 and 10.5
D4.2	benchmark DA filter	see section 10.5
D4.3	scalability coupled UQ and DA	see section 10.5
M4.1 (M16)	UQ campaign single-scale, ARCHER2	see section 11.1
M4.2 (M16)	DA campaign single-scale, ARCHER2	see section 10.5

Table 1: Deliverables: AQUIFER. M2.3 (M8): this task was largely not possible as originally stated due to the lack of stochasticity in the current versions of NESO-Particles, so the work in 5 on stochastic emulation was done instead. The work UKAEA did (in conjunction with UCL, at the December 2022 hackathon) on NESO-UQ (https://github.com/ExCALIBUR-NEPTUNE/NESO-UQ) applies forward UQ to non-stochastic electrostatic PIC and could easily be extended to apply the EasySurrogate part of the toolkit to PIC.

2 Events and reporting

Our level of direct interactions has been 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.

- Hack session with the developers on 2022-05-06 fully capturing the new work in SEAVEA toolkit (FabNEPTUNE)
- UKAEA workshop, 05-06/09/2022 (Abingdon)
- UCL-UKAEA collaborative workshop, 10/03/2023, UCL

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

Hackathons:

- ICCS SEAVEA Hackathon (24 June 2022) at Brunel University London
- SEAVEA Hackathon at University College London (8-9 December 2022)
- SEAVEA Hackathon at Brunel University London (26-27 June 2023)
- CompBioMed & SEAVEA Hackathon at LRZ, Garching, Germany (15 September 2023)
- SEAVEA Hackathon at UCL (1, 4-5 December 2023)

Workshops:

- ICCS SEAVEA Workshop titled '*verification, validation and uncertainty quantification* (VVUQ) on the Exascale' at Brunel University London (23 June 2022)
- SEAVEA Applications meeting at University College London (12 December 2022)
- CompBioMed conference at Science Congress Center Munich, Garching, Germany (12-14 September 2023)
- CIUK ExCALIBUR session at Manchester Central, Manchester, UK (7-8 December 2023)
- SEAVEA workshop at Physics and Applied Mathematics Unit in ISI Kolkata, India on 21st December 2022.
- SEAVEA workshop at Centre of Excellence in climate modelling in Indian Institute of Delhi, India on 9th May 2023
- SEAVEA knowledge exchange at HPC for ABMs Workshop Aberdeen (Feb/March 2024)
- SEAVEA knowledge exchange at Leogang Workshop (Jan 2024)

3 Software outputs

A variety of different software outputs were produced during the project, including development of new and existing packages and integrations with other software outputs produced in the wider NEPTUNE project. Table 2 provides a summary of the software repositories worked on in the project and mentioned in this report.

Repository	Description
Team-RADDISH/ParticleDA	Julia package for distributed particle-filter based data assimilation.
Team-RADDISH/ ParticleDA-UseCases	Example integrations of ParticleDA with external models, specifically: NektarDriftwave, a Nektar++ solver for a plasma physics model; ANAET, a toy ordinary differential equation model of magnetohydrodynamic instability.
UCL/neso-calibration	Python package providing a simple interface for executing NESO solvers and associated examples calibrating NESO models.
UCL/calibr	Parallelized Bayesian calibration of simulations using Gaussian process emulation.
UCL/FabNESO	NESO plugin for FabSim3, facilitating execution of NESO simulations on both local and remote high performance computing systems via a unified interface.
UCL-CCS/FabNEPTUNE	Plugin for FabSim3, facilitating execution of NEPTUNE simulations (Convection2D and Convection3D).
djgroen/FabParticleDA	Plugin for FabSim3, facilitating execution of ParticleDA filtering runs.
wedeling/MD-active-subspace	Python implementations of Deep active subspace and kernelized active subspace and the application in high-dimensional Molecular Dynamics (MD) simulations.

Table 2: Summary of repositories containing packages and code examples developed during AQUIFER.

4 Functional emulation for temperature profile: reduced order modelling

4.1 Study case: 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 T_x on the bottom boundary at steady state by emulator such that:

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

4.2 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

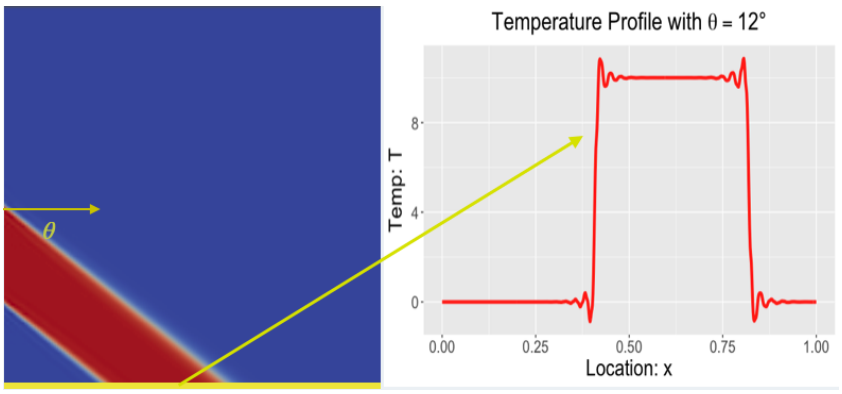
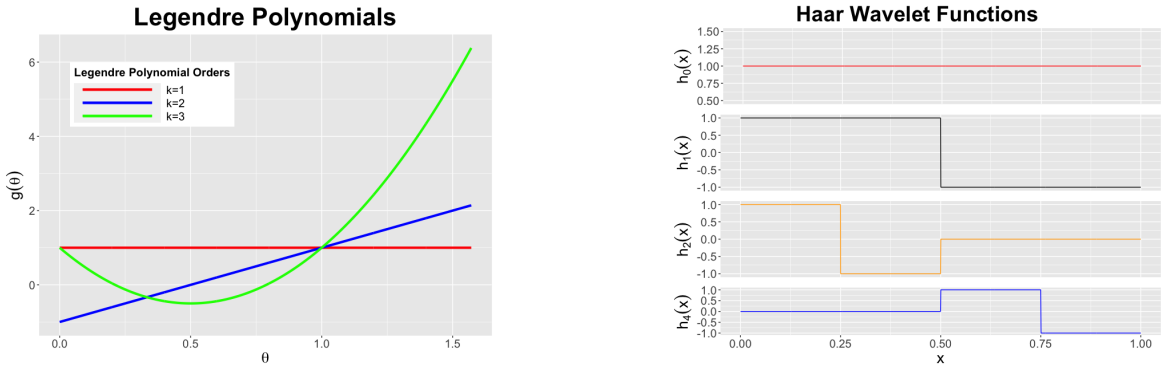


Figure 1: (left) Temperature transfer with incident angle θ (right) Temperature profile on bottom boundaries given θ. The occurrence of nonphysical oscillations, arises due to the discretization of the domain and the 8th order polynomial interpolation in the numerical solver Firedrake (Rathgeber et al. 2016). This issue is a common problem in polynomial interpolation (see Runge's phenomenon (Boyd 1992; Wendland 2004)).



(a) Legendre polynomials

(b) Haar wavelet functions

Figure 2: Choices for regressor functions

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.

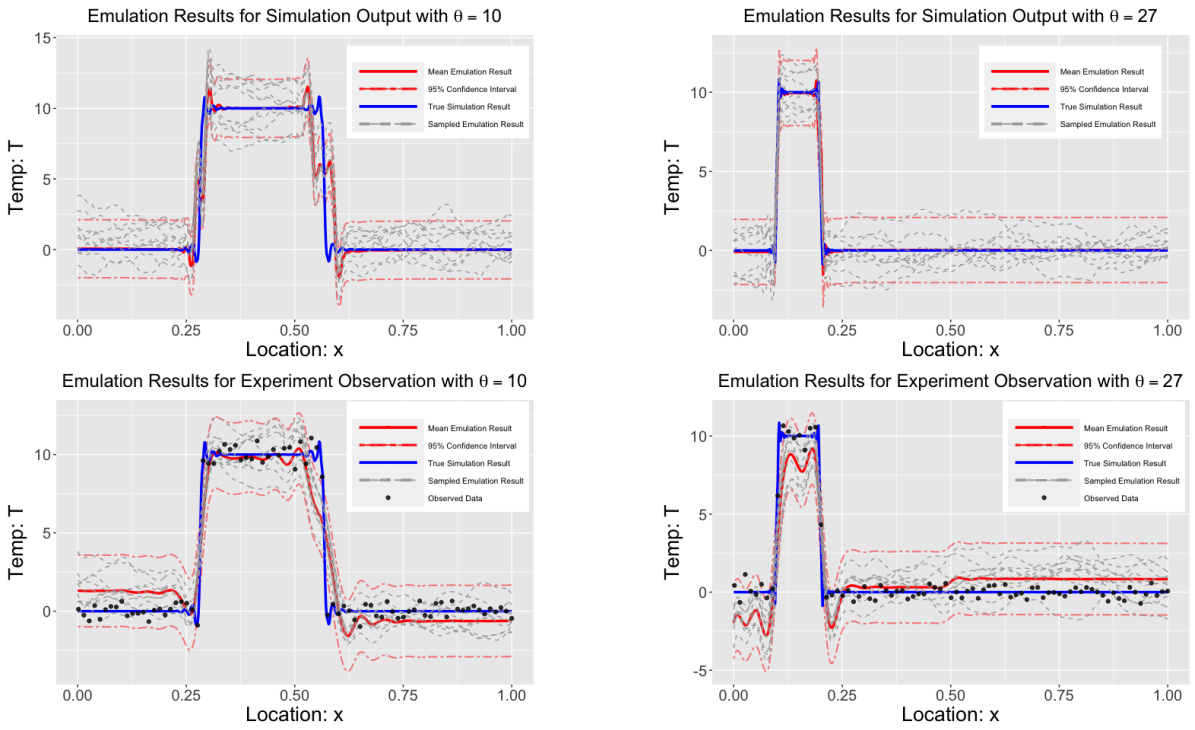


Figure 3: Emulation results for simulation outputs (first row) and pseudo observations with two different θs. The red curves are the mean emulation result, the blue dotted red curves are the 95% Confidence Interval, the blue curves are the true simulation result, and the dotted gray curves are sampled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

4.3 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 discretized values over spatial field. We implement the OPE in two settings:

- **Emulation for simulation**: In this case, We use all the 641 discretized values to fit the OPE for emulating the computational model.
- **Emulation for pseudo-experimental "observation"**: We uniformly sampled 80 values from the 641 discretized 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 2023).

4.4 Linked emulation for coupled proxyapps

In this section, we demonstrate the performances of the Linked Emulator for the coupled proxyapps. The linked emulator consists of two individual GP emulators, which are connected in a feed-forward way. The coupled proxyapp consists of the an-isotropic Heat Transportation as shown in Section 4.1 and take the boundary temperature profile as the input to an isotropic heat diffusion model with the central temperature as the *quantity of interest* (QOI), as shown in Figure 4.

- We applied the *principal components analysis* (PCA) to lower the dimensions of boundary profile and build the first GP emulator for emulating the obtained principal components.

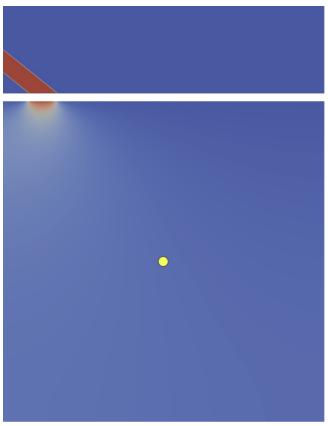


Figure 4: Demonstration of emulated coupled. The top one is the an-isotropic heat transportation and the bottom one is the isotropic heat diffusion model. The inputs are the magnet strength and incident angle. The emulated *quantity of interest* (QOI) is the central temperature, as shown as the yellow dot.

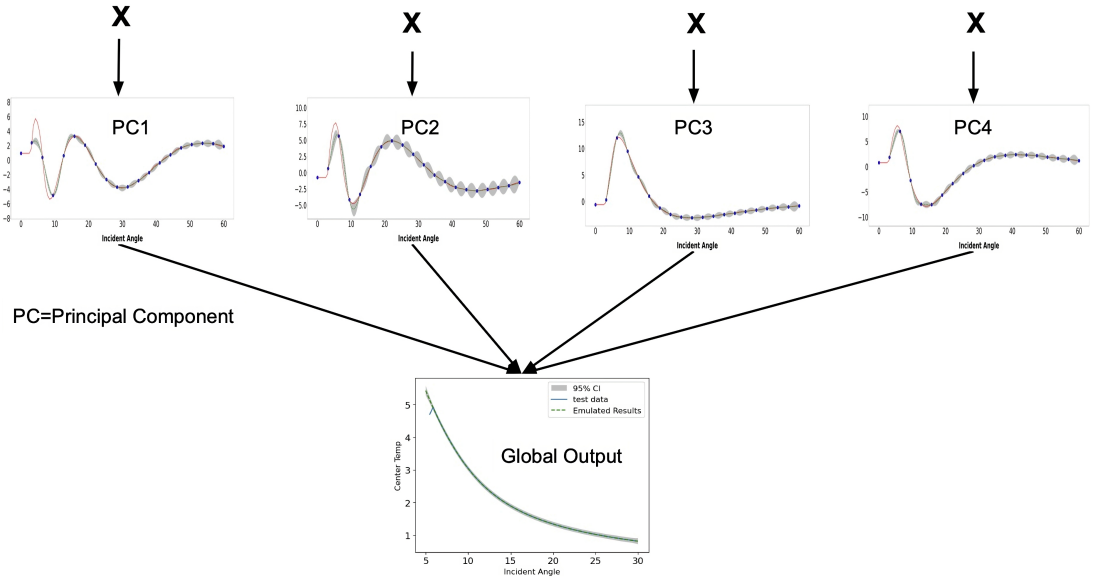


Figure 5: The emulation results and structure of linked GP emulators.

- Then the emulated principal components are used as the input for emulating the QOI by the second GP emulators.

The emulation outcomes and the architecture of the linked emulator are illustrated in Figure 5. Here, the inputs X consist of the incident angle θ and magnet strength B, with the first model generating the temperature profile along the bottom domain as its output. To reduce the dimensionality of the temperature profile (e.g., the resolution of the discretization mesh) from 600 to 4, we employed PCA, also known as *proper orthogonal decomposition* (POD), as depicted in the middle of Figure 4. The second numerical model utilizes the obtained temperature profile as the top boundary condition to simulate the heat diffusion process. Consequently, the second emulator processes the reduced temperature profile as its input, with the central temperature denoted as output Y.

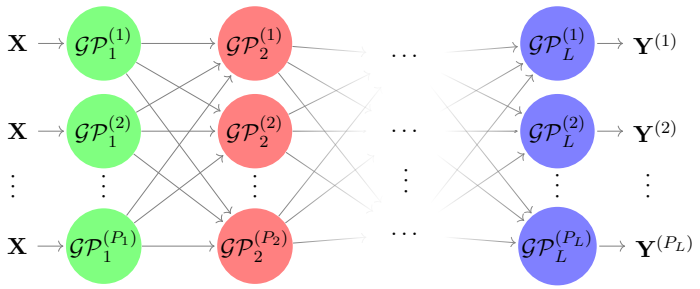


Figure 6: The generic *deep Gaussian process* (DGP) hierarchy

5 Deep Gaussian processes and physics-aware emulation

5.1 Deep Gaussian processes

A *deep Gaussian process* (DGP) is a probabilistic model that builds on the idea of a 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 6). 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 functions, as we will discuss in the following case study. Furthermore, DGPs provide more informative uncertainty estimation in predictions (Ming, Williamson, and Guillas 2023), making them valuable in applications where accurate risk assessment is critical.

5.2 Study case: Lorenz model

The Lorenz model is derived from a two-dimensional model of *Rayleigh–Bénard convection* (RBC), specified by the system of *ordinary differential equations* (ODEs)

$$\begin{aligned} \dot{x} &= \sigma(y - x) \\ \dot{y} &= x(\rho - z) - y \\ \dot{z} &= xy - \beta z \end{aligned} \tag{1}$$

where ρ, σ, β are respectively the Rayleigh number (Ra), the Prandtl number (Pr), and the coupling strength. For this model, the Nusselt number (Nu), which quantifies the heat flux, is given by $Nu = 1 + \frac{2z_\infty}{\rho}$. The term z_∞ is approximated using the long-term average, calculated over 100 000 integration steps. The $(Ra, Pr) - Nu$ surface of the Lorenz model is discontinuous and exists many sub-regimes behaving differently which are shown in Figure 7. In the study, we fixed the coupling strength to 1. The domain for Rayleigh number and Prandtl number is $[0, 100]$.

5.3 Deep Gaussian process results

In this section, we demonstrate the powerful ability of DGPs to emulate non-stationary function: the $(Ra, Pr) - Nu$ surface of the Lorenz model. We compare DGPs with classical GPs, and treed GPs Gramacy and Lee 2008 which combine the flexibility of decision trees with the smoothness of GPs 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 *multi-output Gaussian process* (MOGP) emulator package[1]. The settings for each model are outlined below:

- GP: 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 $\gamma = 0.5$.
- DGP: The DGP model consists of two GPs model above, with length parameters $\gamma_1 = 1.0, \gamma_2 = 0.5$ for stabilizing the numerical computation. The two GPs are connected in serial: We use the *dgpsi* package[2] for implementing the DGP, which is available in both Python and R.

[1] `https://github.com/alan-turing-institute/mogp-emulator`
[2] `https://github.com/mingdeyu/DGP`

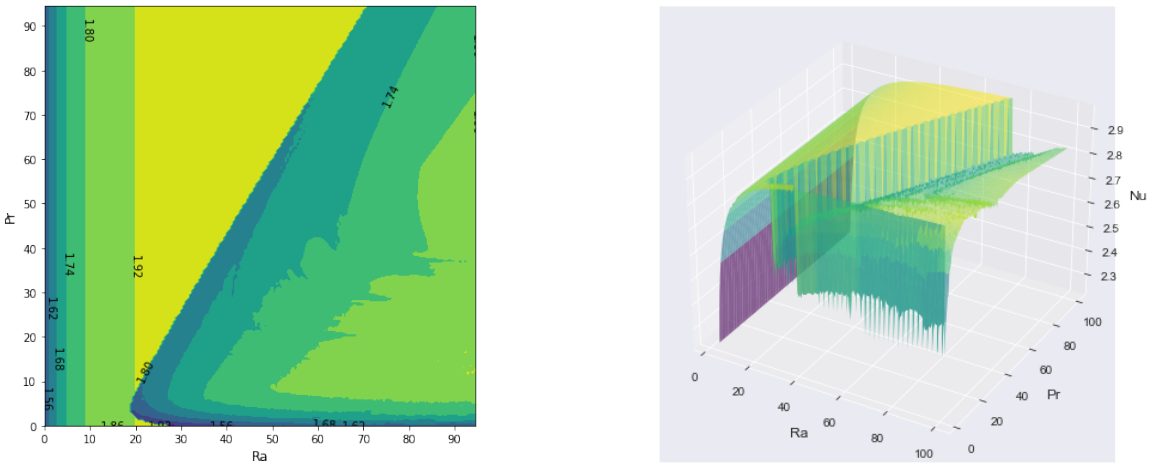


Figure 7: The $(Ra, Pr) - Nu$ surface of Lorenz Model. There are many level sets which show an extreme non-stationarity. Left: Two-dimensional contour plot. Right: Three-dimensional plot.

- Treed GP: 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* package[3]. 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 8. As shown in the figure, both GP and DGP start to capture the discontinuities with only 30 samples, whereas treed GP requires 100 samples. This could be due to the requirement that splits of treed 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 9, we compute the *root mean square error* (RMSE) to measure the performance improvements with an increasing number of 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 10. 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. In summary, DGPs outperforms standard GPs and treed GPs 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.

5.4 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. We again consider the simplified Lorenz model for RBC (see Equation 1), here fixing the coupling strength to $\beta = \frac{8}{3}$.

[3] `https://cran.r-project.org/web/packages/tgp/index.html`

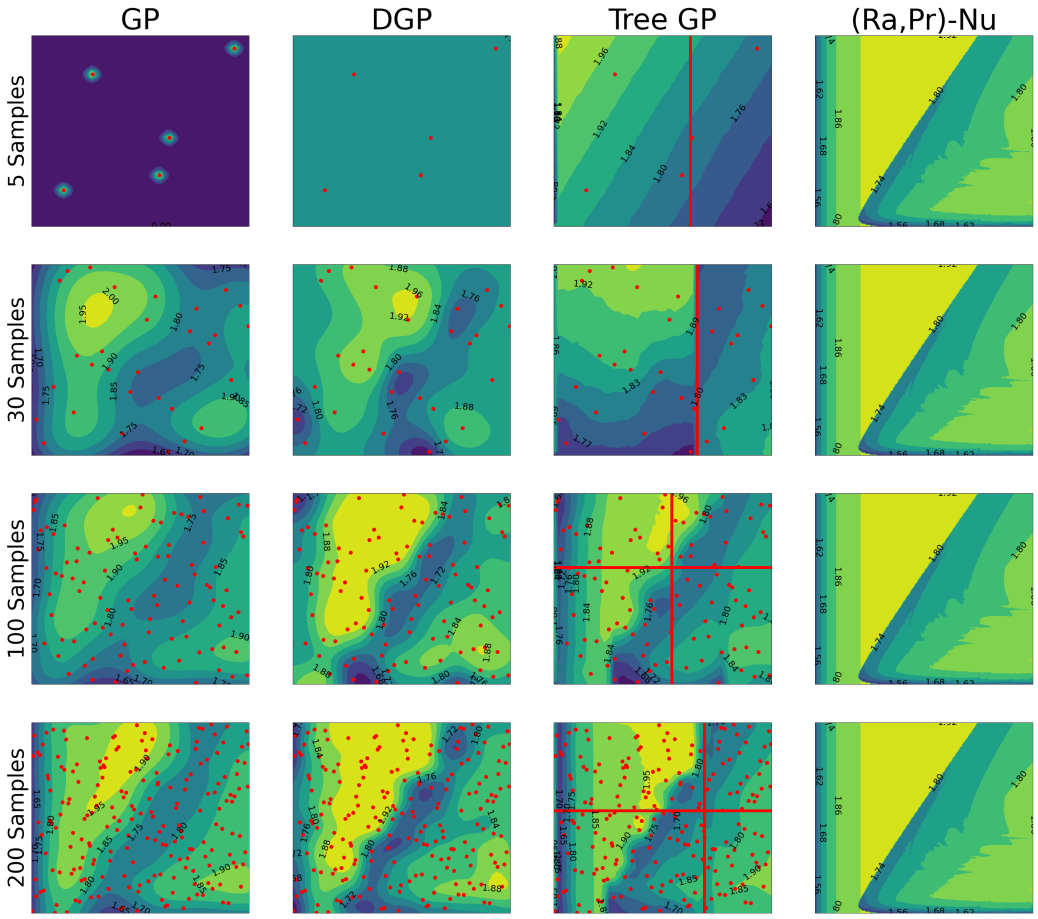


Figure 8: The mean of emulation result: GP, DGP treed GP and true surface (left to right) with 5, 30, 100, 200 samples (top to bottom)

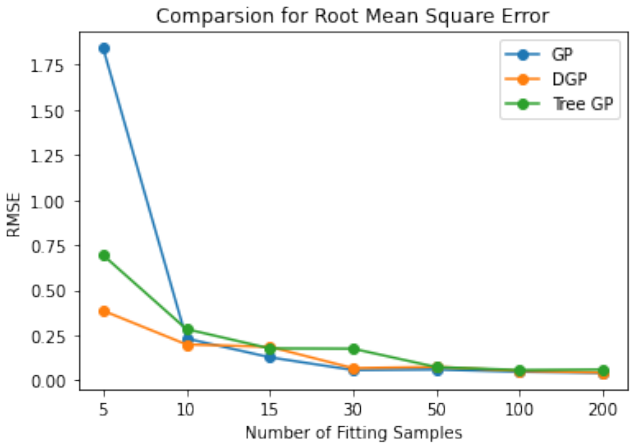


Figure 9: The *root mean square error* (RMSE) of emulation means with respect to the number of fitting samples.

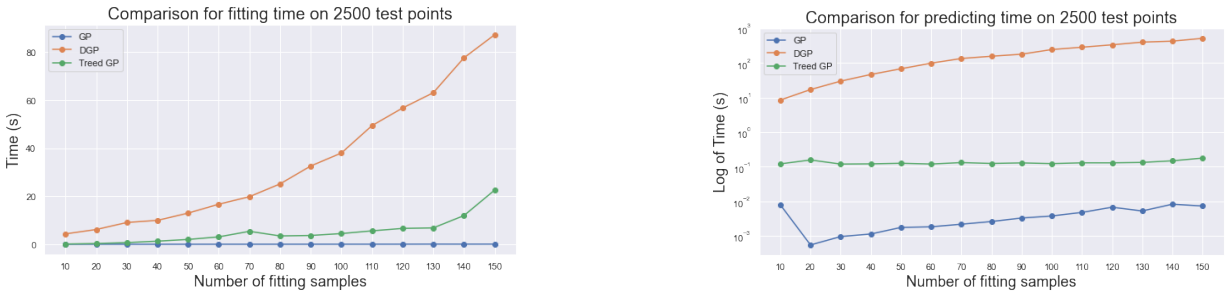


Figure 10: Comparison of fitting (left) and prediction (right) time with respect to the number of fitting samples over 2500 test samples.

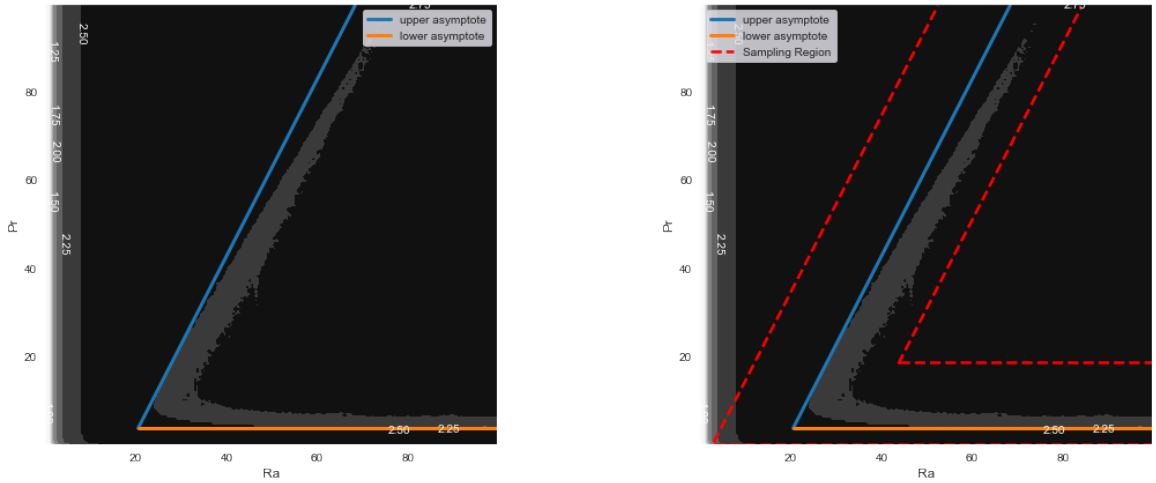


Figure 11: Left: The upper and lower asymptotes $Pr_u(Ra)$ and $Pr_l(Ra)$. Right: Physics-aware sampling region.

In the Lorenz model's $(Ra, Pr) - Nu$ surface, there are known asymptotes that dictate the stability of the fixed points in Lorenz dynamics (Dullin et al. 2007). These asymptotes are represented by the discontinuous boundaries shown in Figure 7, and are characterized by upper and lower limits $Pr_u(Ra)$ and $Pr_l(Ra)$ (see Figure 11), $Pr_u = Ra - 2(\beta + 2)$ and $Pr_l = \beta + 1$.
We created a physics-aware sampling region that covers both asymptotes, with a margin of 15 units (as shown in Figure 11). To cover the non-cube region, we utilized two *latin hypercube design*s (LHDs) (as depicted in Figure 12). In the following investigation, we compared the physics-aware sampling approach with the LHD 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.

5.5 Physics-aware emulation result

The results presented in Figure 13 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 14 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 of 30 samples in Figure 14). 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 LHDs

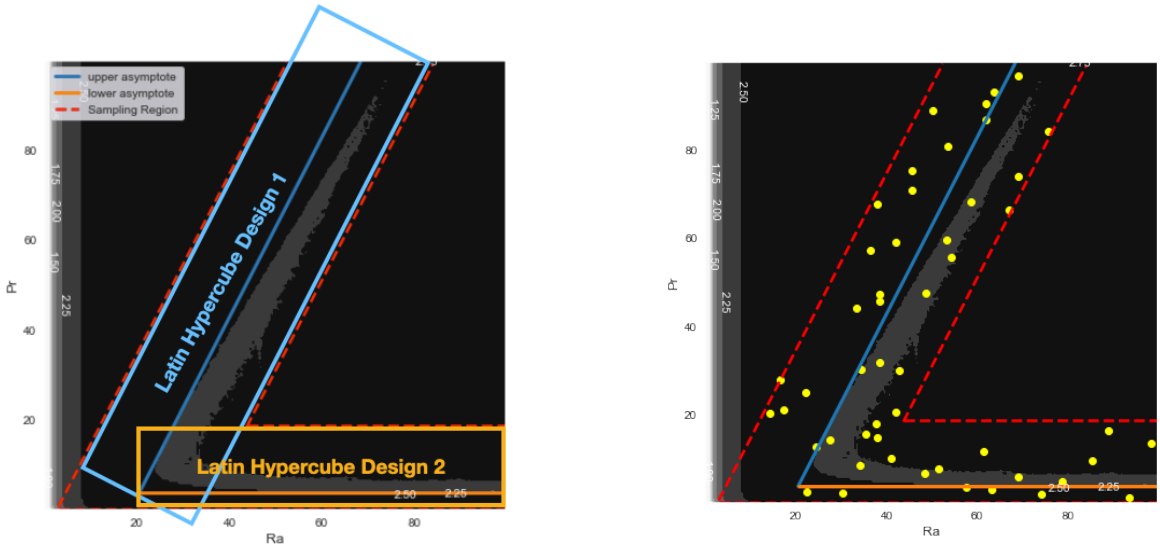


Figure 12: Left: Physics-aware sampling scheme. Right: An example for the sampling scheme

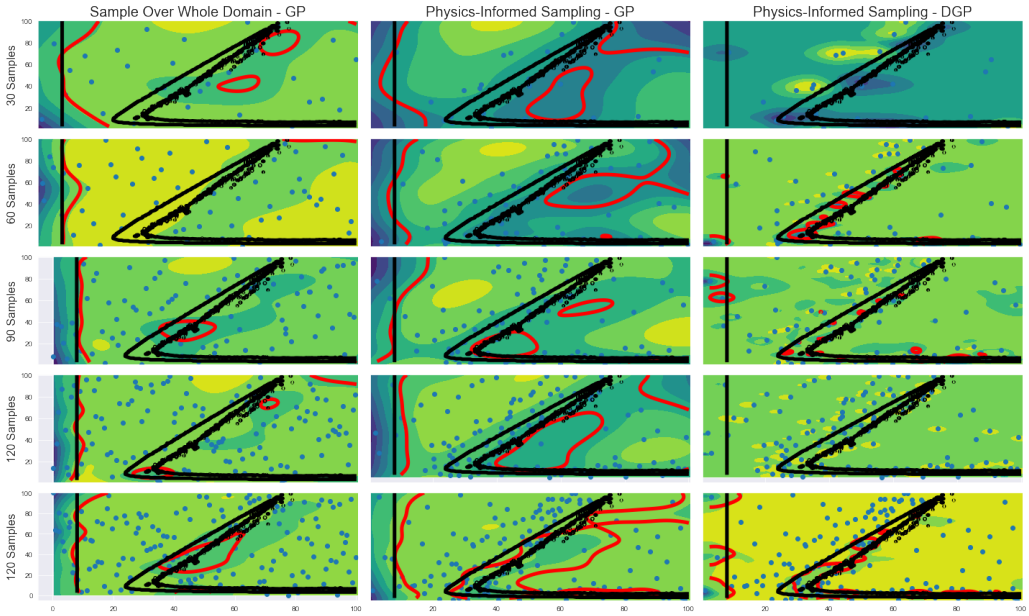


Figure 13: Contour level plots of the predicted means (red curves) and true discontinuous boundaries (black curves). Columns represent methods used, namely GP, GP with physics-aware sampling, and DGP with physics-aware sampling (left to right). Rows correspond to the number of fitting samples used, with 30, 60, 120, and 150 samples (top to bottom).

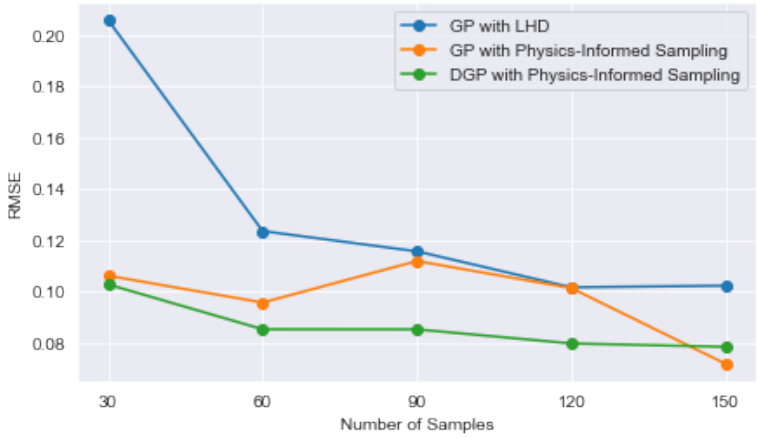


Figure 14: The RMSE for GP, GP with physics-aware sampling, and DGP with physics-aware sampling

and a small uncovered area at the left-bottom corner (see Figure 12, 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.

6 Gradient-based sequential design for localising sharp changes

GP emulators typically assume the computer models are stationary in the input domain, which is usually a overly strong assumption in practice (Gramacy 2020). The presence of sharp changes, or even discontinuities in the output of the computer model can result in deterioration of GP models in terms of both the accuracy and efficiency (See Figure 15). However, such sharp changes widely exist in real world and often indicate bifurcations or critical transitions within the systems under investigation. These instances can be found in switch-like behavior in genomics (Gardner, Cantor, and Collins 2000), bifurcations in fluid dynamics (Rybin et al. 2015; Dullin et al. 2007), and steep transition of competing mechanical phenomena in nuclear safety analysis (Lee and McCormick 2011). In this work, we consider the localization of sharp variations with double purposes:

- **Physical finding.** Identifying these sharp variations holds importance as they can result in significant implications of the investigated systems with better understanding.
- **Better emulation.** Knowing the locations of the sharp changes can aid in constructing the emulators. Thus, the downstream tasks can also be better solved.

6.1 Gradient of Gaussian process emulator

One key property of GPs in our favor is that any linear transformation, such as differentiation and integration, of a GP remain a GP (Rasmussen, Williams, et al. 2006). Gradient is a linear differential operator denoted as ∇. Given a constructed GP emulator $\hat{f}$, the gradient $\nabla\hat{f}(\mathbf{x}_0)$ is a D-dimensional GP emulator. The dth element $[\nabla\hat{f}(\mathbf{x}_0)]_d$ is the partial derivative with respect to dth dimension of x_{0d}. The predictive posterior of $\nabla\hat{f}$ is driven by derivatives of posterior predictive μ_0 and variance σ_0^2 (Scheuerer 2010; McHutchon 2015; Rasmussen, Williams, et al. 2006) as,

$$\mathbb{E}(\nabla\hat{f}(\mathbf{x}_0)) = \nabla\mu_0(\mathbf{x}_0) \quad \text{and} \quad \text{cov}(\nabla\hat{f}(\mathbf{x}_0)) = \nabla \otimes \nabla^T\sigma_0^2(\mathbf{x}_0) \tag{2}$$

From the Equation 2 and predictive posterior of GP emulator, we can obtain the gradient of the predictive posterior explicitly with mean and variance (Graepel 2003; Rasmussen, Williams, et al.

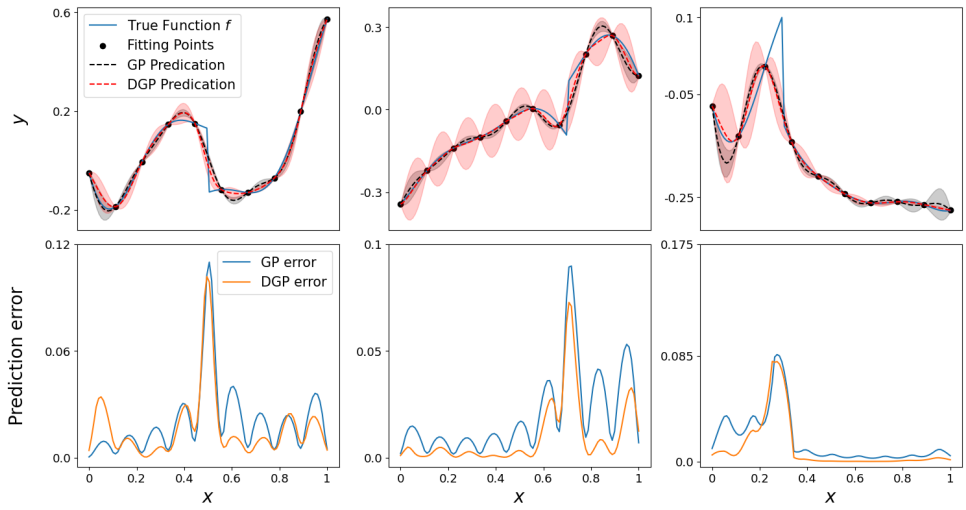


Figure 15: Demonstration of prediction error concentration in regions of High variations. Upper: functions with sharp variations (sample paths of a non-stationary GP) are interpolated using 10 fitting points. Predictions from a GP model with mean zero and squared exponential kernel with length scale $\gamma = 1$. Lower: absolute prediction error.

2006; Solak et al. 2002):

$$\begin{aligned} &\nabla\mu_0(\mathbf{x}_0) = \nabla\mathbf{r}(\mathbf{x}_0)^T\mathbf{R}(\mathbf{x})^{-1}\mathbf{y} \quad \text{and} \\ &\nabla \otimes \nabla^T\sigma_0^2(\mathbf{x}_0) = \sigma^2\big(H - \nabla\mathbf{r}(\mathbf{x}_0)^T\mathbf{R}(\mathbf{x})^{-1}\nabla\mathbf{r}(\mathbf{x}_0)\big), \end{aligned} \tag{3}$$

where $\mathbf{y} \in \mathbb{R}^N$ is the set of emulated targets, $[\mathbf{R}(\mathbf{X})]_{ij} = k(\mathbf{X}_{i*}, \mathbf{X}_{j*}) + \eta\mathbf{1}_{\{\mathbf{X}_{i*}=\mathbf{X}_{j*}\}}$ (the $\mathbf{X}_{i*}$ indicates the ith column of the design matrix $\mathbf{X}$), $\nabla\mathbf{r}(\mathbf{x}_0) = [\frac{\partial k(\mathbf{x}_0,\mathbf{x})}{\partial x_{01}}, ..., \frac{\partial k(\mathbf{x}_0,\mathbf{x})}{\partial x_{0D}}]^T \in \mathbb{R}^D$ and $H = \nabla \otimes \nabla^T k(\mathbf{x}_0, \mathbf{x}_0) \in \mathbb{R}^{D\times D}$ is the Hessian matrix with (i, j) element $[H(\mathbf{x}_0)]_{ij} = \frac{\partial^2 k(\mathbf{x}_0,\mathbf{x}_0)}{\partial x_{0i}\partial x_{0j}}$ and will be a constant if $k(\cdot, \cdot)$ is shift invariant (Wendland 2004).

6.2 Extension to linked Gaussian process emulation

In order to deal with the non-stationarity, in this section, we extend the gradient information from GP into *linked Gaussian process* (LGP) by taking the derivative of its closed-form expectation (Ming and Guillas 2021; Ming, Williamson, and Guillas 2023). Assume that the LGP is constructed with M design points $\mathcal{D} = (\mathbf{x}_m, \mathbf{w}_m, y_m)_{m=1}^M$. The gradient of LGP at new position $\mathbf{x}_0 \in \mathcal{X} \subset \mathbb{R}^D$ can be obtained directly by taking the derivative of $\tilde{\mu}_0(\mathbf{x}_0)$ such that:

$$\nabla\tilde{\mu}_0(\mathbf{x}_0) = \nabla\mathbf{I}(\mathbf{x}_0)^T A, \tag{4}$$

where $A = \mathbf{R}(\mathbf{w})^{-1}\mathbf{y} \in \mathbb{R}^M$, the $\mathbf{I}(\mathbf{x}_0) \in\in \mathbb{R}^M$ (See (Ming and Guillas 2021)), and $\nabla\mathbf{I}(\mathbf{x}_0) \in \mathbb{R}^{M\times D}$ with $[\nabla\mathbf{I}(\mathbf{x}_0)]_{ij} = \frac{\partial I_i(\mathbf{x}_0)}{\partial x_{0j}}$.

6.3 Deep Gaussian process gradient estimation

The DGP emulator can be viewed as a LGP with *latent* internal Input/Output (I/O) $\mathbf{w}$. We can impute N sets of $\mathbf{w}$ iteratively from the conditional Gaussian distribution, and construct N copies of LGP to approximate the DGP (see Ming, Williamson, and Guillas 2023 for more details). Given the N imputed data denoted with $(\mathcal{D}_i)_{i=1}^N$ where $\mathcal{D}_i = (\mathbf{X} = \mathbf{x}, \mathbf{W} = (\mathbf{w}_l^p)_i, \mathbf{Y} = \mathbf{y})$, we can derive the approximate predictive mean and variance of the DGP gradient through a mixture of the gradients from the N constructed LGP emulators,

$$\nabla\tilde{\mu}_0(\mathbf{x}_0) = \frac{1}{N}\sum_{i=1}^{N}\nabla\tilde{\mu}_{0,i}(\mathbf{x}_0) \tag{5}$$

By exploiting the estimations, several indicators can be used to quantify the local variations, reflecting the degree of sharp changes and discontinuities. In this work, we choose the gradient norm which is defined by

$$\begin{aligned} Q(\mathbf{x}_0) &= \|\nabla f(\mathbf{x}_0)\| \\ &= \sqrt{\nabla f(\mathbf{x}_0)^T \nabla f(\mathbf{x}_0)} \\ &\approx \sqrt{\nabla \tilde{\mu}_0(\mathbf{x}_0)^T \nabla \tilde{\mu}_0(\mathbf{x}_0)} \\ &= \|\nabla \tilde{\mu}_0(\mathbf{x}_0)\| \end{aligned} \tag{6}$$

Similarly, the estimation of gradient norm by DGP emulator can be obtained by,

$$Q(\mathbf{x}_0) \approx \frac{1}{N} \sum_{i=1}^{N} \|\nabla \tilde{\mu}_{0,i}(\mathbf{x}_0)\| \tag{7}$$

where each $\nabla \tilde{\mu}_{0,i}(\mathbf{x}_0)$ is computed by each ith imputed LGP.

6.4 Gradient-based acquisition function

In the sequential design, a GP-based emulator of the computer experiment is firstly constructed based on the space-filling initial design with N_0 points $\mathcal{D}_0 = (\mathbf{x}_i, y_i)_{i=1}^{N_0}$. The commonly used space-filling designs are optimized LHDs and minimax-distance designs; see (Santner et al. 2003) for an overview. Once the emulator is constructed, the sequential design begins a loop over computational budget N (ie. the maximum number of computer model runs). In $n+1$th step, there are two steps : (a) Using the emulator to solve an acquisition criteria J_n to determine the next run of the computer model such as $\mathcal{D}_{n+1} = \mathcal{D}_n \cup (\mathbf{x}_{n+1}, y_{n+1})$. (2) Update the emulator with data $\mathcal{D}_{n+1}$. More precisely, the next evaluation $\mathbf{x}_{n+1}$ is determined such that

$$\mathbf{x}_{n+1} = \arg\max_{\mathbf{x} \in \mathcal{X}} J_n(\mathbf{x}) \tag{8}$$

In this paper, we consider the high-variation region as the super level set of the gradient norm, such that $B = \{\mathbf{x} \in \mathcal{X} | g_f(\mathbf{x}) \geq c\}$ where $c \in \mathbb{R}_{\geq 0}$ is certain threshold and $g_f(\mathbf{x}) = \|\nabla f(\mathbf{x})\|$. Then, equivalently, we formulate this problem as a *level set estimation* (LSE) or *contour localization* (CL) problem. The most popular sampling criterion for LSE is the entropy (Oakley 2004; Gotovos et al. 2013; Cole et al. 2023; Booth, Renganathan, and Gramacy 2023). Denote the $p_{\mathbf{x}} = p(\|\nabla f(\mathbf{x})\| \geq c)$ as the probability of $\mathbf{x} \in S$, i.e $\mathbf{x}$ located with in the high-variation region. The entropy acquisition function in LSE:

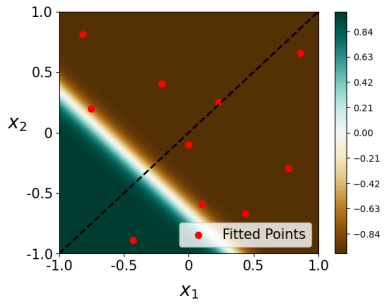
$$S(\mathbf{x}) = -p_{\mathbf{x}} \log(p_{\mathbf{x}}) - (1 - p_{\mathbf{x}}) \log(1 - p_{\mathbf{x}})$$

The S will be high in the boundary of region $B, \partial B$ (i.e., $p_{\mathbf{x}} \approx 0.5$). It is indeed the transition region we are looking for.

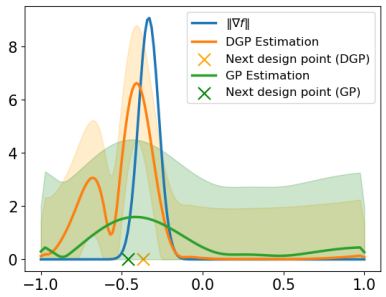
$$\mathbf{x}_{n+1} = \arg\max_{\mathbf{x} \in \mathcal{X}_{\text{cand}}} S(\mathbf{x})$$

For the general LSE problem, the $p_{\mathbf{x}}$ can be easily evaluated as the *cumulative distribution function* (CDF) of the predictive distribution of GP emulator. In our sequential learning case, we hope to allocate more samples around the region B but also achieves good explorations of the input domain. So we do not need to accurately evaluate the full distribution of gradient norm. We only need an indication for the degree of variation and the uncertainty of the input domain. We combine the gradient norm estimation of DGP and the predictive variance as these indications to form the new acquisition function,

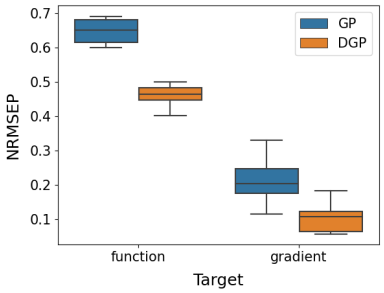
$$\begin{aligned} S(\mathbf{x}) &= \frac{1}{N} \sum_{i=1}^{N} -p_{\mathbf{x}}^i \log(p_{\mathbf{x}}^i) - (1 - p_{\mathbf{x}}^i) \log(1 - p_{\mathbf{x}}^i) \\ \text{and} \quad p_{\mathbf{x}}^i &= 1 - \Phi_{\text{trun}}\left(\frac{c - \|\nabla \tilde{\mu}_{0,i}(\mathbf{x})\|}{\tilde{\sigma}_{0,i}(\mathbf{x})}\right), \end{aligned} \tag{9}$$



(a) The two-dimensional plateau function



(b) Emulation results of gradient norm



(c) Quantitative analysis of emulation results in function and gradient

Figure 16: The demonstration of gradient-norm emulation. (a) indicates the two-dimensional plateau function. The black dashed line indicates the diagonal of the input domain, in which we evaluated the gradient in (b), red points are used to fit the GP and DGP. (b) GP and DGP emulation results of gradient norm (mean +/- 2 standard deviation, where negative standard deviation truncated at 0). (c) Quantitative analysis of emulation results over 150 points on diagonal. The mean and standard deviations of these results are obtained over 10 random initial designs.

where Φ_{trun} is the CDF of truncated normal distribution. In most cases, the threshold c is not known. We take a substitution of the explicit threshold level by an implicit level as

$$c_{\text{imp}} = (1-\alpha)\max_{\mathbf{x}\in\mathcal{X}} \|\nabla\tilde{\mu}_0(\mathbf{x})\|,$$

where $\alpha \in [0,1]$ and is pre-determined. We show a demonstrative figures in the qualitative and quantitative performance of the proposed acquisition function in Figure 16. The Figure shows a simple case of a two-dimensional plateau function,

$$f(x) = 2\Phi\Big(\sqrt{2}(-4-6(x_1+x_2)\Big) - 1 \quad \text{and} \quad x \in [-1,1]^2$$

We constructed the emulators using a dataset of 10 points sampled randomly via LHD, as shown in Figure 16 (left). For visualization purposes, we present the emulation results for the gradient norm along the domain's diagonal line, indicating the next design points selected by both GP and DGP methods. The design point chosen by DGP more close to the high variation region, aligning with our objectives. The right panel provides a quantitative analysis of the gradient norm emulation's accuracy, obtained by fitting the emulators with 20 different sets of LHD points.

6.5 Study case: phase transitions in Rayleigh-Bénard convection

We again consider the simplified Lorenz model for RBC (see Equation 1), here fixing the coupling strength to $\beta = \frac{8}{3}$.

6.5.1 Settings

- DGP **emulator:** Two-layered DGP with squared exponential kernel
- **Design:** Initialize with three LHD samples and adding designs (one in each iteration) sequential up to 50.
- **Evaluation metrics:**
  1. Localization: Log likelihood of Bernoulli distribution

$$\text{LLK} = \sum_{i=1}^{n_t} y_i \log\left(p_{\mathbf{x}_i}\right) + (1-y_i)\log\left(1-p_{\mathbf{x}_i}\right)$$

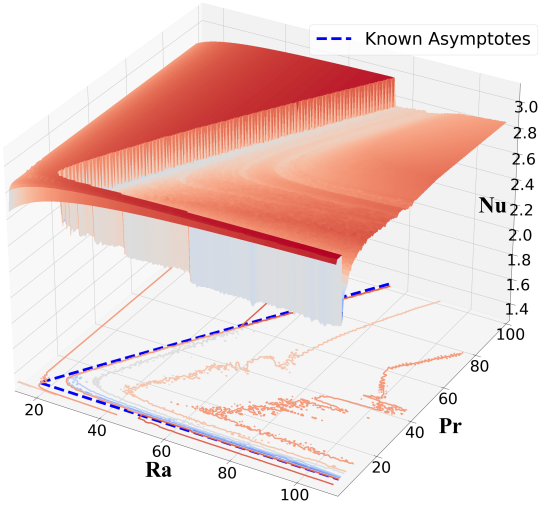


Figure 17: The $(Ra, Pr) - Nu$ surface of Lorenz Model. The dashed blue lines are known asymptotes of the phase transition. More details could be found in Dullin et al. 2007.

where the $y_i = \mathbf{1}\{\mathbf{x}_i \in B\}$ and the $p_{\mathbf{x}_i} = 1 - \Phi(\frac{\hat{c} - \mu_Q(\mathbf{x}_i)}{\sigma_Q(\mathbf{x}_i)})$.

2. Approximation: *normalized root mean square error* (NRMSE)

$$\text{NRMSE} = \frac{\sqrt{\frac{1}{n_t}\sum_{i=1}^{n_t}(\mu_0(\mathbf{x}_i) - f(\mathbf{x}_i))^2}}{\max_{i=1:n_t} f(\mathbf{x}_i) - \min_{i=1:n_t} f(\mathbf{x}_i)}$$

where $(\mathbf{x}_i, f(\mathbf{x}_i)_{i=1}^{n_t}$ is the gridded test data set with $n_t = 2500$.

- Compared methods: 1. GP with proposed sequential design, 2. DGP with 'mutual information for computer experiments' (MICE) Beck and Guillas 2016, 3. DGP with 'active learning Cohn' (ALC) Cohn, Ghahramani, and Jordan 1994.

6.5.2 Qualitative analysis

Figure 18 demonstrates the sequential localization procedure of our proposed algorithms. The color indicates the probability that the gradient norm beyond the implicit threshold c_{imp}, which is the phase transition region. We can see that the boundaries are clearly captured by using 50 samples, especially the diagonal one. However, empirically, we found the algorithms are not robust enough and sensitive to the hyper settings such as initial design and c_{imp}. Our next step will figure out how to stabilize the algorithms with external physical constraint. One possible way is to incorporate the numerical parameter continuation method, since most of such phase transitions can be attributed to the bifurcations and instabilities of dynamical systems, see Ma and Wang 2005.

6.5.3 Quantitative analysis

In addition to qualitative analysis, we also made a quantitative analysis to compare with other sequential design methods in this RBC problem. The results are obtained from re-running 20 times with different LHD initial designs. From Figure 19, due to the existences of sharp variations, our designed methods outperform than other sequential design methods in the accuracy of both localization and global approximation.

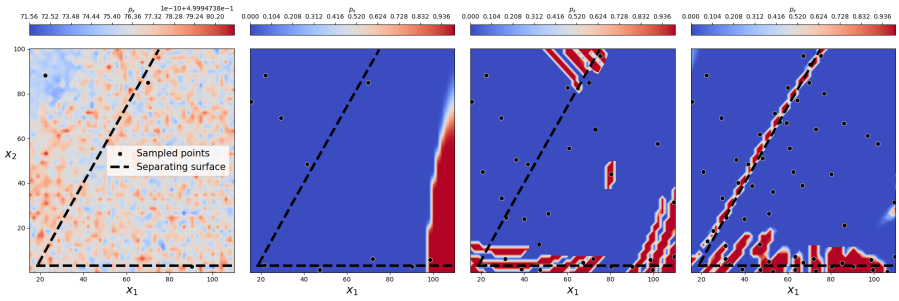


Figure 18: Localization of high variation region B. The color indicates the probability that the gradient norm beyond the implicit threshold c_{imp}. The four surfaces are estimated with $3, 10, 30, 50$ samples from left to right.

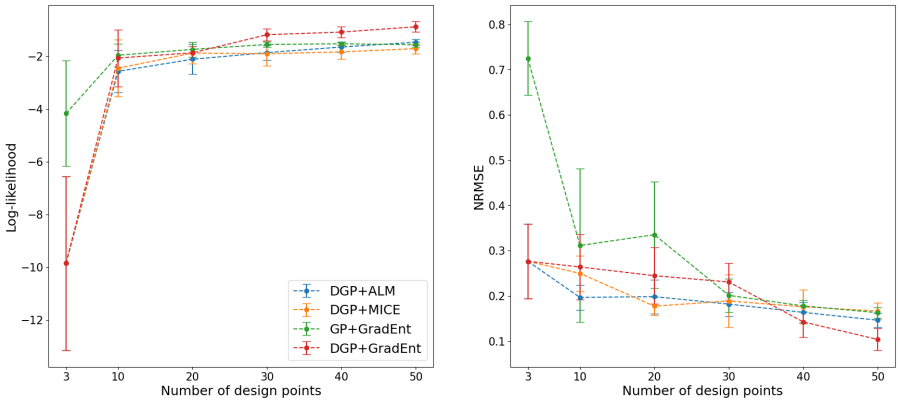


Figure 19: Error analysis of RBC parameter surfaces with 20 random initial designs. Left: Number of design points versus negative log-likelihood. Right: Number of design points versus NRMSE.

7 Stochastic emulation by heteroscedastic Gaussian processes

7.1 Study Case: parameter study in Tritium breeding ratio

Tritium breeding ratio (TBR), which describes the number of Tritium atoms formed per incident neutron, is an essential quantity in sustaining the fusion reaction. TBR depends on the large amount of factors in a complex probabilistic way. In this case study, We simplify the problems by only focusing on the following two factors,

- **Material property**: Lithium-6 enrichment (%) (percentage of Lithium-6 in the material of breeding blanket)
- **Reactor geometry**: Blanket radius with fixed neutron point source.

The neutron transportation can only be characterized with respect to certain stochastic process given design parameters (i.e. the material property and reactor geometry). The *stochastic nature* requires numerous Monte Carlo samples for TBR estimation, making it computationally difficult to explore the design parameter space, especially in high dimensional space. An accurate emulator/surrogate model with reliable uncertainty estimation is desired to capture its output variance.

7.2 Heteroscedastic Gaussian processes

GP regression provides a Gaussian distribution approximation $\hat{P}(\mu, \Sigma)$ to the distribution P. However, the homogeneity of variance in standard GP is problematic. When design parameter $\boldsymbol{\theta}$ changes, the physical system will change a lot, as well the inside stochasticity.

Standard GP regression: $y = f(x) + \epsilon$

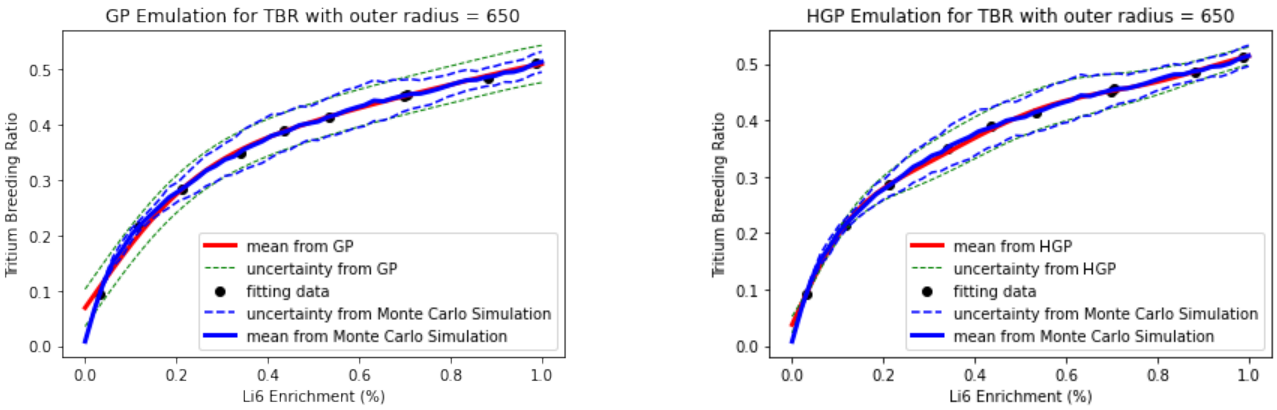


Figure 20: Left: Standard GP regression. Right: HGP regression. The dark dot is the mean of Monte Carlo samples. The fitting data consists of OpenMC mean and OpenMC variance.

- $f(x) \sim GP(\mu_f(x), K_f(x, x'))$
- $\epsilon \sim \mathcal{N}(0, \sigma^2)$

Heteroscedastic GP regression: $y = f(x) + \epsilon(x)$

- $f(x) \sim GP(\mu_f(x), K_f(x, x'))$
- $\epsilon(x) \sim \mathcal{N}(0, r(x))$
  s.t, $r(x) = \exp\big(g(x)\big)$ and $g(x) \sim GP(\mu_g(x), K_g(x, x'))$

However, the marginal likelihood and predictive density is no longer analytically tractable. There are massive past works in approximate inference: variational inference, *Markov chain Monte Carlo* (MCMC) sampling Ming, Williamson, and Guillas 2023. In Figure 20, we compared the performances between standard and heteroscedastic GPs. Both of the two models are fitted with mean and variance estimation of TBR from batch of 500 particles in the OpenMC simulation with a simple onion tokamak configuration. From Figure 20, we can clearly observe that the *heteroskedastic Gaussian process* (HGP) (right panel) captures the variation of variances better than standard GP (left panel).

8 Application: uncertainty quantification and sensitivity analysis for molecular dynamics simulations with kernelized active subspace

The objective of the present project is to perform a high-dimensional active-subspace based uncertainty quantification analysis to assess the influence of aleatoric, simulation and particularly force-field parameter uncertainty on classical *molecular dynamics* (MD) predictions (see Figure 21). The original active subspace approach requires the explicit estimation of the derivatives of model response $\nabla f(x)$, which are not available in the MD simulation. Instead, we applied *gradient-based kernel dimension reduction* (GKDR) for the dimension reduction with a GP denoted as *kernelized active subspace Gaussian process* (KAS-GP) in the following discussion, as the low-dimensional surrogate and make a comparison with the recently proposed neural-network based active subspace called the *deep active subspace* (DAS) method. Both of the two approaches are derivative-free and achieve similar performances in the dimension reduction, as well the associated tasks: *uncertainty quantification* (UQ) and *sensitivity analysis* (SA).

8.1 Investigated molecular dynamics simulations

- Epoxy-resin thermosetting polymer materials, predicting mechanical properties, namely E, the Young's modulus, an indicator of the stiffness of the material, and the Poisson ratio, which is the ratio of lateral deformation to axial deformation when straining the material in a given direction.

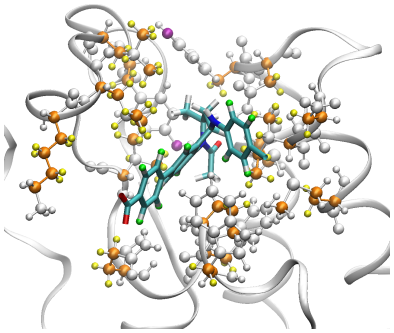


Figure 21: Positions of the atoms in the binding site of the ligand-protein complex, of which the force-field parameters are most sensitive in *enhanced sampling of molecular dynamics with approximation of continuum solvent* (ESMACS) study. The ligand is represented as bond, and the protein is shown as ribbon in white. The residues at the binding site are shown as ball and stick. The sp3 carbon atoms (coloured orange) and attached hydrogen atoms (yellow) from protein have the most sensitive parameters, along with hydrogen atoms (green) from ligand. The parameters for the oxygen atoms (purple) from two tyrosine residues are also important to the sensitivity.

- protein-ligand biomolecular systems, where binding free energies are computed. Two types of binding free energies are investigated: the absolute binding free energy (shortened as "binding free energy" hereafter) which is a quantitative measure of the strength of protein-ligand binding, and the relative binding free energy which is the difference of binding free energies between two ligands. Reliable prediction of such properties plays an important role in drug discovery and personalized medicine

Within this study, we have used two different molecular dynamics engines, LAMMPS for epoxy-resin polymer materials and NAMD for protein-ligand biomolecular systems. The absolute and relative binding free energies are calculated using the *enhanced sampling of molecular dynamics with approximation of continuum solvent* (ESMACS) and *thermodynamic integration with enhanced sampling* (TIES) protocols, respectively.

8.2 Results

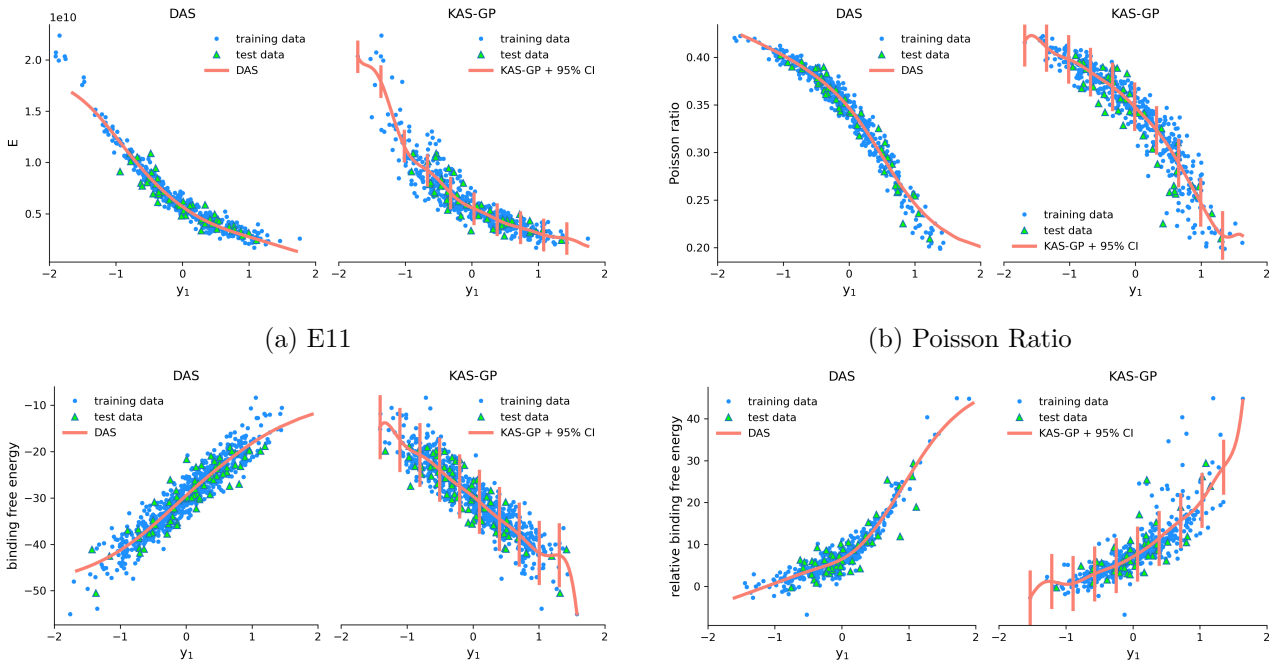
The epoxy KAS-GP and DAS surrogates for E and the Poisson ratio are plotted vs the $d = 1$ dimensional active subspace y_1 in Figure 22a and 22b. Note that the one-dimensional function captures the overall trend of the data well, for both QOIs. Importantly, the variation of the MD data $f(x)$ at a given location in the active subspace, i.e. $\mathrm{Var}[f(x)\|y_1]$, is heavily concentrated around the prediction with quantified uncertainty from GP. This holds for both the training data and the test data (10% of the data set), which was not used in constructing the surrogates. Hence, while the original MD model is a function of a 103-dimensional input space , it is well approximated by a one-dimensional surrogate. The one-dimensional ESMACS binding-energy surrogates are shown in Figure 22c. A one-dimensional active subspace is clearly visible, although the variance of the training and test data around the surrogate is larger than for the epoxy surrogates. The TIES one-dimensional relative binding free energy surrogates are shown in Figure 22d. The one-dimensional active subspace is again visible, although (like the ESMACS case) it is not of the same quality as for the epoxy surrogates.

8.3 Global derivative-based sensitivity analysis

To assess which inputs are most influential, commonly-used options are global variance-based sensitivity methods such that,

$$\nu_i := \int \left(\frac{\partial f}{\partial x^i} \right)^2 p\,(x)\,\mathrm{d}x. \tag{10}$$

These indices measure the (average) sensitivity of f to small perturbations in the inputs x, and are especially suited for identifying non-influential parameters. To connect (10) to the active subspace



(a) E11

(b) Poisson Ratio

(c) Binding free energy

(d) Relatively binding free energy

Figure 22: The DAS and KAS-GP surrogates of all QOIs plotted along the first active variable y_1.

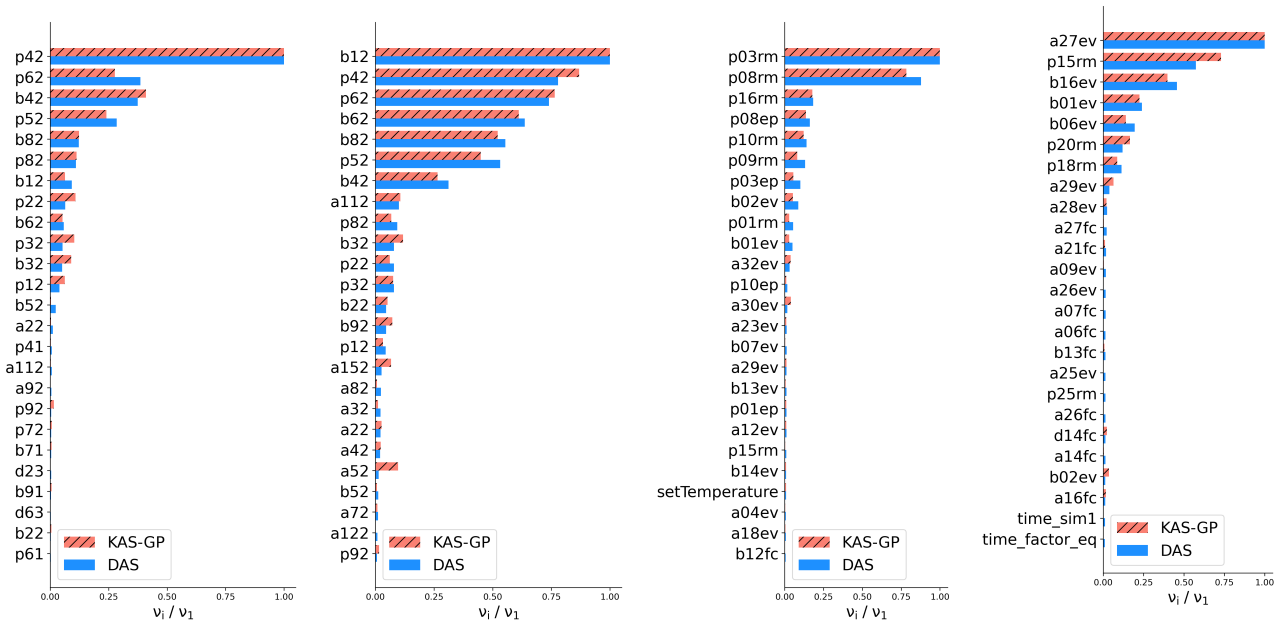
method, note that the ν_i are the diagonal elements of the C matrix;

$$[\nu_1, \cdots, \nu_d]^T = \text{diag}\,(C)\,. \tag{11}$$

In most cases, ∇f will not be available. In DAS method, the derivative can be estimated by the automatic differentiation via the deep neural network. To establish a connection between Equation 10 and the GKDR method, we can view the ν_i as the global derivative-based sensitivity index in the embedded *reproducing kernel Hilbert space* (RKHS). By the reproducing property of $k_{\mathcal{Y}}$, we can write the $\phi_i = \frac{\partial}{\partial x^i}\mathbb{E}\big[k_{\mathcal{Y}}(\cdot, Y)|X = x\big]$ and represents the partial derivative of the embedded conditional expectation. Then we can estimate the sensitivity index ν_i by GKDR such that

$$\begin{aligned}\tilde{\nu}_i &= \int \langle \phi_i^x, \phi_i^x \rangle_{\mathcal{H}_{\mathcal{Y}}} dP(x) \\ &\approx \frac{1}{n}\sum_{j=1}^{n} \text{Diag}(\hat{M}(x_j))_i \\ &= \text{Diag}(\tilde{M}_n)_i\end{aligned} \tag{12}$$

This approach mirrors the methodology employed in both active subspace and DAS methods, where the sensitivity index ν_i is estimated using the i-th diagonal element of the matrix $\tilde{M}_n$. The derivative information is implicitly provided through the derivative kernel $\frac{\partial k(\cdot,x)}{\partial x^i}$ for $i = 1, .., d$, facilitating a derivative-free approach. A comparative analysis of the sensitivity analysis results obtained through both GKDR and DAS methods (refer to Figure 23) shows a high degree of similarity. This empirical evidence substantiates the theoretical link between GKDR and active subspace methods, demonstrating their agreement in practical applications.



(a) E11 (b) Poisson Ratio (c) Binding free energy (d) Relatively binding free energy

Figure 23: The 25 largest global derivative-based sensitivity indices for the DAS and KAS-GP (stands for GKDR with low dimensional GP surrogate) method, ordered as $\nu_1 \geq \nu_2 \geq \cdots \geq \nu_{25}$. The indices are normalized by ν_1 and ordered according to the DAS ranking, although the KAS-GP ranking is similar.

9 Polynomial chaos expansion and stochastic collocation (surrogate models: novel simplex stochastic collocation)

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 (*polynomial chaos expansion* (PCE) / *stochastic collocation* (SC)) will display nonphysical oscillations, see Figure 24. 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

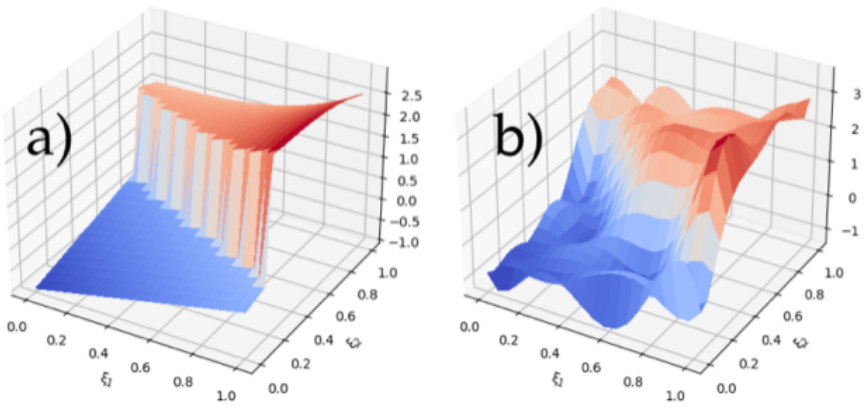


Figure 24: Discontinuous test function and standard SC surrogate.

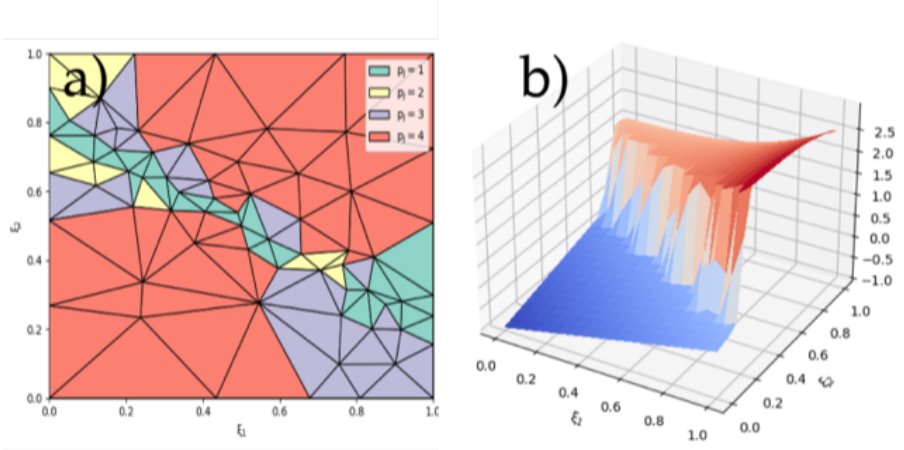


Figure 25: Local mesh and resulting SSC surrogate for the test function in 24 a.

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 24 b. The SSC sampling plan and surrogate are shown in Figure 25, which can be replicated by running the Jupyter notebook tutorial in the EasyVVUQ 'tutorials' folder [4]. Next steps involving applying the SSC method to a Nektar++ case.

10 Data assimilation in plasma physics models

Data assimilation encompasses algorithmic approaches for combining assumptions about a system formulated as a *numerical model*, with *observations* of the system, to estimate how the *state* of the system evolves over time. As a distinct field, data assimilation has its origins in numerical weather prediction, where combining physical models with data is the central task. More generally data assimilation can be considered a form of (Bayesian) statistical inference in which we combine *prior* information (the numerical model) with observed data to infer a *posterior* distribution on the system state given both observations and prior model.

Numerous data assimilation approaches have been proposed. For *linear-Gaussian* models — that is models with linear state dynamics, observations that linearly depend on the state and Gaussian observation and state noise — the *Kalman filter* (Kalman 1960) offers an efficient approach for exactly computing the sequence of distributions on the system state at each observation time given all observations up to that time (the *filtering distributions*), by recursively updating estimates of the mean and covariance of the Gaussian filtering distributions. For models with non-linear state transitions or observation operators, the *extended Kalman filter* can be used, which use linearizations of the state and observation updates computed using the Jacobians (matrices of partial derivatives) of the corresponding operators.

As (extended) Kalman filters require updating a covariance matrix which is of size $d_x \times d_x$ where d_x is the state dimension, and solving a linear system in a matrix of size $d_y \times d_y$, where d_y is the observation dimension, they become infeasible to apply to models for which d_x or d_y are very large, with $\mathcal{O}(d_x^2)$ memory and $\mathcal{O}(d_y^3)$ floating point operation costs. *Ensemble Kalman filters* (Evensen 1994; Evensen 2006) offer an alternative approach that form a Monte Carlo approximation to the Kalman filter updates using an *ensemble* of N particles, with typically $N \ll d_x$, with the $\mathcal{O}(Nd_x)$ memory and $\mathcal{O}(\min(d_y, N)^3 + \min(d_y, N)^2 \max(d_y, N) + d_x \min(d_y, N)N)$ floating point operations costs of ensemble Kalman filter updates (Mandel 2006; Roth et al. 2017) allowing application to models with much larger state dimensions, including use in operational numerical weather prediction systems.

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

While they can be applied to models with non-linear state transitions and observation operators, the ensemble Kalman filter updates will only converge to exactly computing the filtering distributions as $N \to \infty$ for linear-Gaussian models, and the quality of their estimates can be poor in models with strongly non-linear dynamics or observations, or non-Gaussian noise.

Variational methods (Rabier and Liu 2003) are another widely used approach for data assimilation. Here, the problem of estimating a sequence of distributions on the system state, is relaxed to instead computing a point estimate of the 'most likely' state at each observation time. This is implemented by iteratively minimizing a cost function which combines terms accounting for our prior knowledge about the system (based on short-term forecasts from previous state estimates and an assumed forecast error covariance) and our current (noisy and partial) observations of the system state.

Two main variants exist - 3D-Var, which finds a system state which minimizes a cost function including terms only for observations of the initial state, and 4D-Var, which includes terms for observations of both the initial state and the future states within some *assimilation window*. In both cases, for non-linear observation models, computation of the gradient of the cost function require being able to compute the action of the adjoint of the Jacobian of the observation operator (sometimes referred to as the *tangent linear model*), and in the 4D-Var case, when the state transition operator is non-linear we also require being able to compute the action of the adjoint of the Jacobian of the state transition operator.

Variational methods are widely used in practice in operational numerical weather prediction systems, though as they produce only point estimates, they typically require separate approaches for uncertainty quantification. The standard 'strong constraint' formulations of variational methods also do not account for error in the model's predictions of how the state evolves over time, effectively assuming the model is perfect over the assimilation window. Weak constraint 4D-Var (Evensen, Vossepoel, and Leeuwen 2022) relaxes the perfect model assumption, at a cost of increasing the dimension of the space being optimized over, with the approach jointly optimizing both an initial state and values for model error 'forcing terms'.

Particle filters (Gordon, Salmond, and Smith 1993) offer an alternative to both variational methods and ensemble Kalman filter. Similarly to ensemble Kalman filters, particle filters use an ensemble of state particles to represent the estimates of the sequence of filtering distributions. Rather than form a Monte Carlo approximation to the Kalman filter updates for linear-Gaussian state space models, the particle filter instead use a (importance sampling) Monte Carlo approximation to the recursions updating the filtering distribution at one time point to the next for *general* state space models, and so in the limit of the ensemble size $N \to \infty$ give consistent estimates of the true filtering distributions for arbitrary state space models with potentially non-linear dynamics and observation operators or non-Gaussian noise distributions.

Particle filters iteratively alternate between a *proposal* step in which the new values for each particle in the ensemble are independently generated from a proposal distribution and a *resampling* step in which the particles are resampled according to a set of importance weights, with highly weighted particles being duplicated and low weighted particles being discarded. The proposal step and computation of (unnormalized) importance weights can be parallelized across all particles, and for complex state space models will typically dominate the computational cost of filtering, making particle filters ideally suited to deployment on highly parallel HPC systems, and so for performing data assimilation at the exascale. While normalization of the importance weights and the resampling step do require synchronization across particles, the operation cost remains linear in the ensemble size N and only computation of the (scalar) normalized importance weights requires collective operations across all processors, with redistribution of the particles during resampling able to be done using sparse point-to-point transfers. In comparison, for the ensemble Kalman filter the operation costs scale as $\mathcal{O}(N^3 + d_y N^2 + d_x N^2)$ if $d_y > N$ and $\mathcal{O}(d_y^3 + d_y^2 N + d_x d_y N)$ otherwise, and the linear algebra operations in the filtering updates require collective operations on the full state vectors (particles) when distributing across multiple processors, resulting in a much higher communication overhead.

Although particle filters offer promise in terms of their ability to give consistent estimates for non-linear and non-Gaussian state space models, and ease of scaling on HPC systems, a key drawback of particle filters is that they can require an ensemble size which scales exponentially with the dimen-

sionality of the model (Snyder 2011; Fearnhead and Künsch 2018), to avoid a degeneracy sometimes terms *ensemble collapse*, by which all members of the ensemble except one are assigned negligible weights. While use of improved proposal distributions can help combat this issue to a degree (Snyder 2011; Slivinski and Snyder 2016), algorithmic extensions such as tempering (Frei and Künsch 2013; Beskos, Crisan, and Jasra 2014; Bunch and Godsill 2016; Beskos, Crisan, Jasra, et al. 2017) or spatial localization (Rebeschini and Handel 2015; Farchi and Bocquet 2018; Graham and Thiery 2019), are likely to be required for statistically robust use of particle filters for data assimilation in state space models of very high dimension, such as for the numerical models of plasma physics of interest in the NEPTUNE project.

10.1 State space models

Particle (and ensemble Kalman) filters assume the system of interest is represented as a *state space model*. Specifically we assume we have a sequence of observations $\boldsymbol{y}_{1:T} = (\boldsymbol{y}_t)_{t=1}^T$ of the system at $T > 0$ times with each observation a d_y dimensional real-valued vector, $\boldsymbol{y}_t \in \mathbb{R}^{d_y}$. The state of the system at each time index $t \in 1{:}T$ is also assumed to be representable by a d_x dimensional real-valued vector $\boldsymbol{x}_t \in \mathbb{R}^{d_x}$.

The state dynamics are assumed to be *Markovian*, with the state at each time index depending only on the previous state, with initial and transition distributions specified by densities p_0 and $p_{1:t}$ respectively,

$$\boldsymbol{x}_0 \sim p_0(\cdot); \qquad \boldsymbol{x}_t \sim p_t(\cdot \mid \boldsymbol{x}_{t-1}) \quad t \in 1{:}T;$$

and the observations at each time index t are assumed to depend only on the current state, with condition distributions specified by densities $g_{1:t}$

$$\boldsymbol{y}_t \sim g_t(\cdot \mid \boldsymbol{x}_t) \quad t \in 1{:}T.$$

Here the notation $\boldsymbol{z} \sim q(\cdot)$ denotes that a random vector $\boldsymbol{z}$ is distributed according to a probability distribution with density q.

A key point here is that the state transitions are assumed to be *stochastic*. This may seem initially restrictive for the setting of interest here, simulation of plasma physics, where typically models are specified as time-dependent systems of *partial differential equation*s (PDEs), with the state evolution in such models inherently deterministic. In any real setting however the model will only be a partial description of the underlying phenomena being modelled, and if we do not account for this by specifying we have some uncertainty in how the state evolves over time compared to in the true system, we will struggle to get useful outputs from any data assimilation algorithm.

As a simple example we could consider trying to fit a *simple harmonic motion* (SHM) model to observations of a pendulum state. Under this deterministic model the pendulum state at all future (and past) times would be entirely determined by its initial (angular) position and momentum, so the problem reduces down to finding an initial state such that the simulated trajectories are consistent with the observations, with the simulated trajectories being sinusoids of constant frequency and amplitude. In practice the real world observed pendulum state trajectories will not be exactly sinusoidal – depending on the amplitude of the oscillations the neglected non-linear effects in the small angle approximation underlying SHM may be significant, and probably more importantly, frictional effects not accounted for in the SHM model will always mean the oscillations will decrease in amplitude over time. When we try to fit the model to the observed data we will therefore not be able to find some initial state for the system which leads to trajectories consistent with all the observed data, with we likely to instead only be able to find states which are consistent with short contiguous periods of the observations for which the SHM assumptions roughly hold.

If we instead fit, for example, a stochastic differential equation model to the observations which includes terms both describing a deterministic evolution according to our SHM model and a white noise term representing unmodelled aspects of the dynamics like frictional effects (and when discretized in time can be formulated as a state space model), then the additional flexibility in the model would allow us to find state sequences that are consistent with all the observational data, while still allowing us to make useful inferences about variables of the deterministic model which will provide a naturally

parsimonious description of key aspects of the dynamics (near constant frequency oscillations) with the noise model then only needing to account for the features not described by our model (decaying amplitude, slightly non-sinusoidal oscillations, slow variations in frequency). If we increase the complexity of the deterministic component of model to include additional aspects of the system – for example including a damping term to partially account for frictional effects – then more of the observed behaviour will be able to explained by the deterministic component of the model, with the stochastic component therefore needing to account for less unmodelled aspects, potentially reflected for example by we inferring a smaller scale parameter for the injected white noise process.

These considerations applies equally to plasma physics, with models likely to differ from the true systems of interest both due to unmodelled physics and limitations in the ability of the numerical methods used to simulate models to resolve behaviour at all temporal and spatial scales. In such models, a key question is how to represent our uncertainties about the evolution of the system state, and importantly ensure that the perturbations we introduce to model our uncertainty do not disrupt the key underlying physical behaviours we are seeking to model, including any physical constraints on the system state such as boundary conditions.

10.2 Particle filters

The objective of the particle filter is to estimate the filtering distribution for each time index t which is the conditional probability distribution of the state $\boldsymbol{x}_t$ given observations $\boldsymbol{y}_{1:t}$ up to time index t, with the the filtering distribution at time index t denoted π_t. As a slight abuse of notation we denote distributions and their densities (with respect to the Lebesgue measure unless otherwise noted) with the same symbol, so that π_t denotes both the filtering distribution at time index t and its density.

An ensemble of particles $(\boldsymbol{x}_t^{(n)})_{n=1}^N$ represents an approximation to the filtering distribution at each time index $t \geq 1$ as an empirical distribution $\pi_t(\cdot \mid \boldsymbol{y}_{1:t}) \approx \frac{1}{N}\sum_{n=1}^N \delta_{\boldsymbol{x}_t^{(n)}}(\cdot)$, where $\delta_{\mathbf{x}}$ is a Dirac measure at $\boldsymbol{x}$. For the proposal step at each time index t, new values for the particles are sampled from a proposal distribution with density q_t given the particle values at the previous time index and current observations

$$\tilde{\boldsymbol{x}}_t^{(n)} \sim q_t(\cdot \mid \boldsymbol{x}_{t-1}^{(n)}, \boldsymbol{y}_t), \quad n \in 1{:}N;$$

and (unnormalized) importance weights computed for each proposed particle value

$$w_t^{(n)} = \frac{p_t(\tilde{\boldsymbol{x}}_t^{(n)} \mid \boldsymbol{x}_{t-1}^{(n)})\, g_t(\boldsymbol{y}_t \mid \tilde{\boldsymbol{x}}_t^{(n)})}{q_t(\tilde{\boldsymbol{x}}_t^{(n)} \mid \boldsymbol{x}_{t-1}^{(n)}, \boldsymbol{y}_t)}, \quad n \in 1{:}N.$$

Both of the sampling of proposals and computation of the unnormalized importance weights can be performed independently (and so in parallel) for each particle.

In the resampling step, the weighted proposed particle ensemble is resampled from the weighted empirical distribution

$$\boldsymbol{x}_t^{(n)} \sim \frac{\sum_{n=1}^N w_t^{(n)} \delta_{\tilde{\boldsymbol{x}}_t^{(n)}}(\cdot)}{\sum_{n'=1}^N w_t^{(n')}}, \quad n \in 1{:}N;$$

to produce a new uniformly weighted ensemble to use as the input to the next filtering step. This step requires synchronization across the particles.

A particular simple choice of proposal distribution, is the naïve 'bootstrap' proposals which ignore the observations and sample the proposals from the state transition distributions

$$q_t(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{y}_t) = p_t(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}), \quad t \in 1{:}T;$$

with the expression for the unnormalized importance weights then simplifying to $w_t^{(n)} = g_t(\boldsymbol{y}_t \mid \tilde{\boldsymbol{x}}_t^{(n)})$, $n \in 1{:}N$. The bootstrap proposals can be straightforwardly implemented any state space model for which we can sample from the state transition distributions and evaluate the observation densities, however the resulting particle filter can perform poorly when the observations are informative about

the state, with the proposed particles typically ending up far from the regions containing the most of the mass of the trued filtering distributions, resulting in importance weights with high variance, and weight degeneracy with all but one normalized importance weight close to zero.

An alternative to the bootstrap proposals which can reduce the tendency for weight degeneracy and ensemble collapse, is to use the *locally optimal proposal distribution* (Doucet, Godsill, and Andrieu 2000)

$$q_t(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{y}_t) = \frac{p_t(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1})\, g_t(\boldsymbol{y}_t \mid \boldsymbol{x}_t)}{\int p_t(\boldsymbol{x} \mid \boldsymbol{x}_{t-1})\, g_t(\boldsymbol{y}_t \mid \boldsymbol{x})\, \mathrm{d}\boldsymbol{x}}, \quad t \in 1{:}T;$$

which minimizes the variance of the importance weights, with the importance weights in this case simplifying to $w_t^{(n)} = \int p_t(\boldsymbol{x} \mid \boldsymbol{x}_{t-1}^{(n)})\, g_t(\boldsymbol{y}_t \mid \boldsymbol{x})\, \mathrm{d}\boldsymbol{x},\ n \in 1{:}N$ which are independent of the sampled values of the particle proposals $\tilde{\boldsymbol{x}}_t^{(1:N)}$. Sampling from this locally optimal proposal distribution and computing the corresponding importance weights, is intractable in general as the integral in the definition of the proposal distribution does not usually have a closed form solution.

However, for an important subclass of state space models with additive Gaussian state and observation noise and linear observation operators, that is

$$\boldsymbol{x}_t \sim \mathsf{Normal}(\cdot \mid F_t(\boldsymbol{x}_{t-1}), Q_t), \quad \boldsymbol{y}_t \sim \mathsf{Normal}(\cdot \mid H_t \boldsymbol{x}_t, R_t), \qquad t \in 1{:}T;$$

the locally optimal proposals have the tractable form

$$q_t(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{y}_t) = \mathsf{Normal}\left(\boldsymbol{x}_t \mid m_t(\boldsymbol{x}_{t-1}, \boldsymbol{y}_t), C_t\right),$$

with

$$m_t(\boldsymbol{x}_{t-1}, \boldsymbol{y}_t) = F_t(\boldsymbol{x}_{t-1}) + Q_t H_t^T (H_t Q_t H_t^T + R_t)^{-1} (\boldsymbol{y}_t - H_t F_t(\boldsymbol{x}_{t-1})),$$

and

$$C_t = Q_t - Q_t H_t^T (H_t Q_t H_t^T + R_t)^{-1} H_t Q_t;$$

and corresponding importance weights

$$w_t^{(n)} = \mathsf{Normal}(\boldsymbol{y}_t \mid H_t F_t(\boldsymbol{x}_{t-1}), H_t Q_t H_t^T + R_t).$$

10.3 ParticleDA

ParticleDA (Giles et al. 2023) is a Julia package implementing particle filtering algorithms for data assimilation, with a focus on allowing the particle filtering updates to be computed at scale on HPC systems using both thread-based parallelism and distributed processing using a *message passing interface* (MPI) implementation. It was developed initially as part of a project *real-time advanced data assimilation for digital simulation of numerical twins on HPC* (RADDISH), with an initial focus on geoscientific applications such as tsunami modelling and numerical weather prediction. As well as its support for both thread-based and MPI based parallelism, it provides implementations of particle filters using both bootstrap and locally optimal proposal distributions.

As part of UCL's work on AQUIFER, we made various improvements to ParticleDA, both to extend the range of models it can be used with and improve performance:

- The locally optimal proposal implementation was generalised to allow use with a more general class of state space models (`https://github.com/Team-RADDISH/ParticleDA.jl/pull/218`).
- ParticleDA's model and filter interfaces were refactored to remove the previous implicit assumption of models being finite difference based spatial discretization of a time-dependent PDE, with the state vector correspond to the values of the state field(s) at a regular grid of points (`https://github.com/Team-RADDISH/ParticleDA.jl/pull/232`). Post refactoring ParticleDA now requires only that the model's state can be represented as a flat vector of fixed dimension, allowing ParticleDA to be applied to non-spatial models such as systems of ODEs or to time-dependent PDE models using for example finite element based spatial discretizations on potentially arbitrarily structured spatial meshes.

- The parallelization of the particle proposal steps across multiple threads was changed from a static scheme in which groups of particles were pre-assigned to each thread, to a scheme in which the particles updates are chunked across a user specifiable number of tasks, with these tasks then being scheduled across threads dynamically, with this improving load balancing and reducing the chance of threads sitting idle (`https://github.com/Team-RADDISH/ParticleDA.jl/pull/247`).

10.4 ANAET state space model

As an initial test case illustrating how ParticleDA can be used to perform data assimilation, we consider a toy 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 (γ_r, mu_1, μ_2, μ_6, ν_1, ν_2, δ_0, δ_1) are free parameters. Defining $\boldsymbol{x} = (a, \dot{a}, b)$, these equations can be written as a first order ODE system

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

Here, to formulate as a state space model, we consider a simple stochastic evolution variant of this model defined by the *stochastic differential equation* (SDE) system

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

with $\boldsymbol{w}(t)$ a Wiener process, with the magnitude of the noise introduced into the state dynamics controlled by a shared scalar parameter $\beta > 0$.

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

$$y_t \sim \mathsf{Normal}(\cdot \mid x_1(\tau_t), \sigma^2), \quad t \in 1{:}T.$$

Our state space model formulation uses a Gaussian approximation to the state transition distributions for the SDE system based on a splitting numerical scheme which uses an adaptive ODE solver to solve for the deterministic component of the dynamics and Euler–Maruyama discretization for the driving Wiener noise processes.

The code for the Julia implementation of the model, along with installation instructions and example usages with ParticleDA is available at `https://github.com/Team-RADDISH/ParticleDA-UseCases/tree/main/ANAET`.

As an example, we will use a particle filter to estimate the state trajectories $\boldsymbol{x}_{1:T}$ with $\boldsymbol{x}_t = \boldsymbol{x}(\tau_t)$, $\tau_t = t$ and $T = 100$, given simulated observed data $\boldsymbol{y}_{1:T}$ generated from the model. The model parameters are fixed to the values $\gamma_r = 1$, $\mu_1 = 0$, $\mu_2 = -2$, $\mu_6 = 0.001$, $\nu_1 = 0.005$, $\nu_2 = 0.0001$, $\delta_0 = 0.0001$, $\delta_1 = 0$, $\sigma = 0.5$ and $\beta = 0.02$. We generate a set of simulated observed data from the model using a fixed initial state $\boldsymbol{x}_0 = (1, 2.5, 0.01)$ when simulating the observations, but when filtering use an initial state distribution with $\boldsymbol{x}_0 \sim \mathsf{Normal}(\cdot \mid (0, 2, 0), \mathbf{I})$, where $\mathbf{I}$ is the identity matrix.

We use ParticleDA to compute particle approximations to the filtering distributions $\pi_{1:T}$, using its implementations of both the bootstrap and locally optimal proposals. Figures 26, 28 and 30 show results for the locally optimal proposals with $N = 125$, $N = 250$ and $N = 500$ particles respectively, and Figures 27, 29 and 31 show results for the bootstrap proposals with $N = 125$, $N = 250$ and $N = 500$ particles respectively. In all cases, for the filtering estimates, the blue line shows the

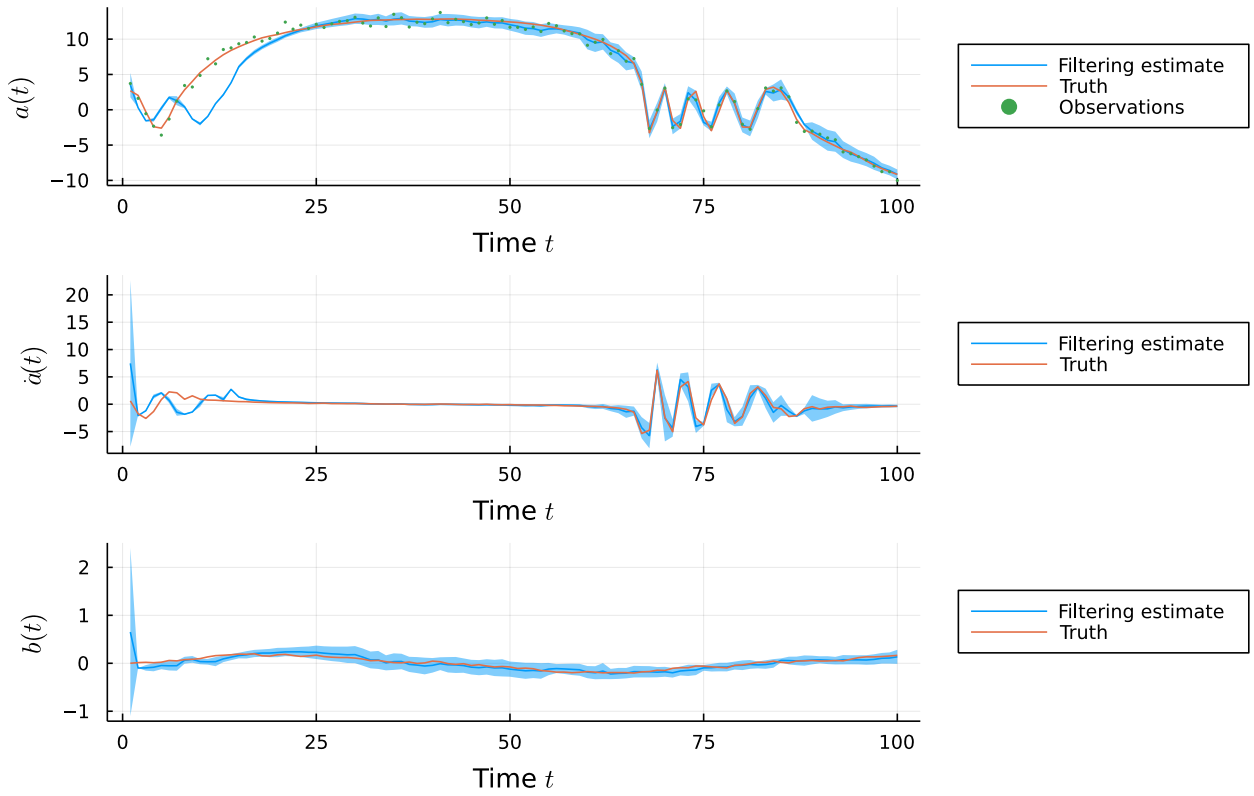


Figure 26: True state trajectories (orange), observed data (green markers) and filtering estimates (blue) for stochastic ANAET model using particle filter with $P = 125$ and locally optimal proposals.

estimated filtering distribution mean and the blue filled region the estimated mean ± three estimated standard deviation interval.

From Figures 30 and 31 see that particle filters using both the locally optimal and bootstrap proposals appear to give reasonable estimates of the filtering distribution with $N = 500$ 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 $N = 250$ (Figures 28 and 29) and $N = 125$ (Figures 26 and 27), 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.

10.5 NektarDriftwave state space model

As a second more complex test model, we consider integrating ParticleDA with a Nektar++ solver for a plasma physics relevant time-dependent PDE model. Specifically we define a state space model which extends the `nektar-driftwave` proxyapp, which uses Nektar++ to solve the two-dimensional Hasegawa–Wakatani equations,

$$\begin{aligned}
\partial_3\zeta(s_1, s_2, \tau) + \{\phi, \zeta\}(s_1, s_2, \tau) &= \alpha(\phi - n)(s_1, s_2, \tau), \\
\partial_3 n(s_1, s_2, \tau) + \{\phi, n\}(s_1, s_2, \tau) &= \alpha(\phi - n)(s_1, s_2, \tau) - \kappa\partial_2\phi(s_1, s_2, \tau), \\
(\partial_1^2 + \partial_2^2)\phi(s_1, s_2, \tau) &= \zeta(s_1, s_2, \tau);
\end{aligned}$$

where the Poisson bracket operator is defined as

$$\{a, b\}(s_1, s_2, \tau) = \partial_1 a(s_1, s_2, \tau)\partial_2 b(s_1, s_2, \tau) - \partial_2 a(s_1, s_2, \tau)\partial_1 b(s_1, s_2, \tau).$$

The two dimensional spatial domain is assumed to be rectangular with periodic boundary conditions. Nektar++ is used to spatially discretize the system using a spectral element method, with here a

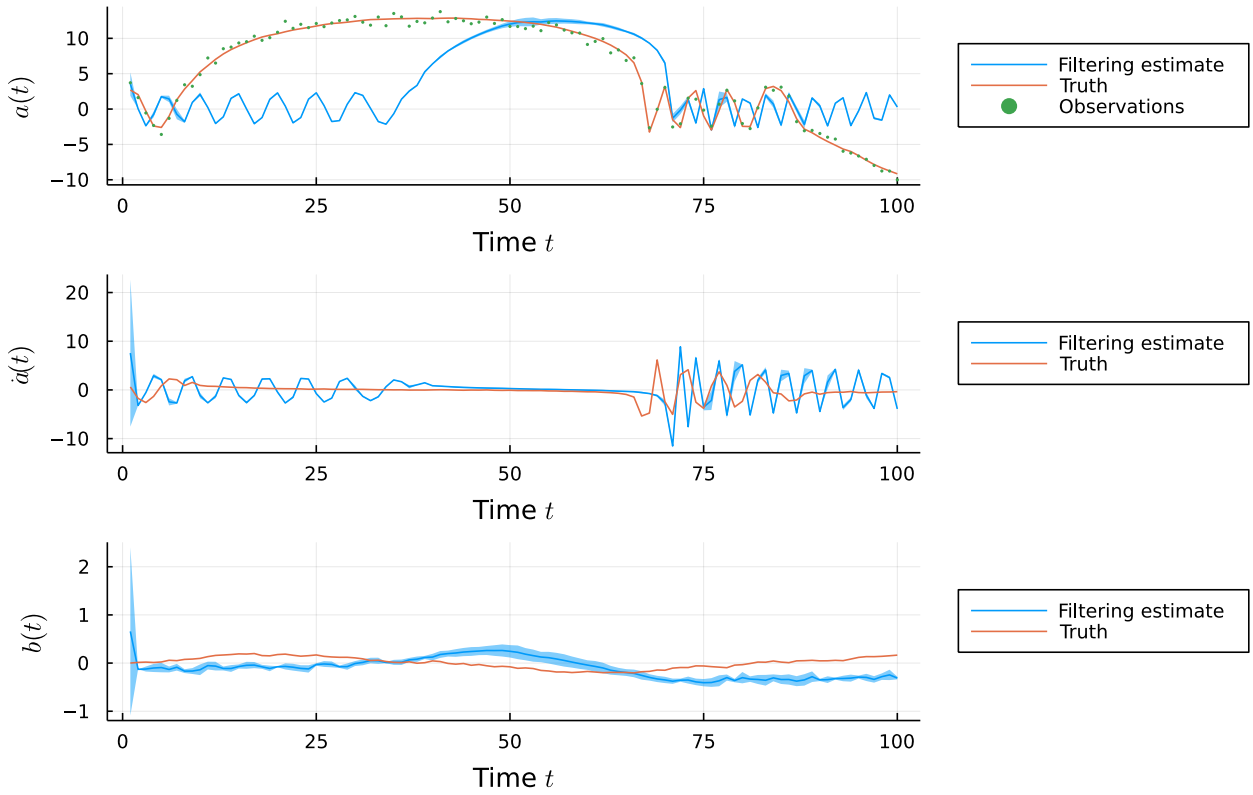


Figure 27: True state trajectories (orange), observed data (green markers) and filtering estimates (blue) for stochastic ANAET model using particle filter with $P = 125$ and bootstrap proposals.

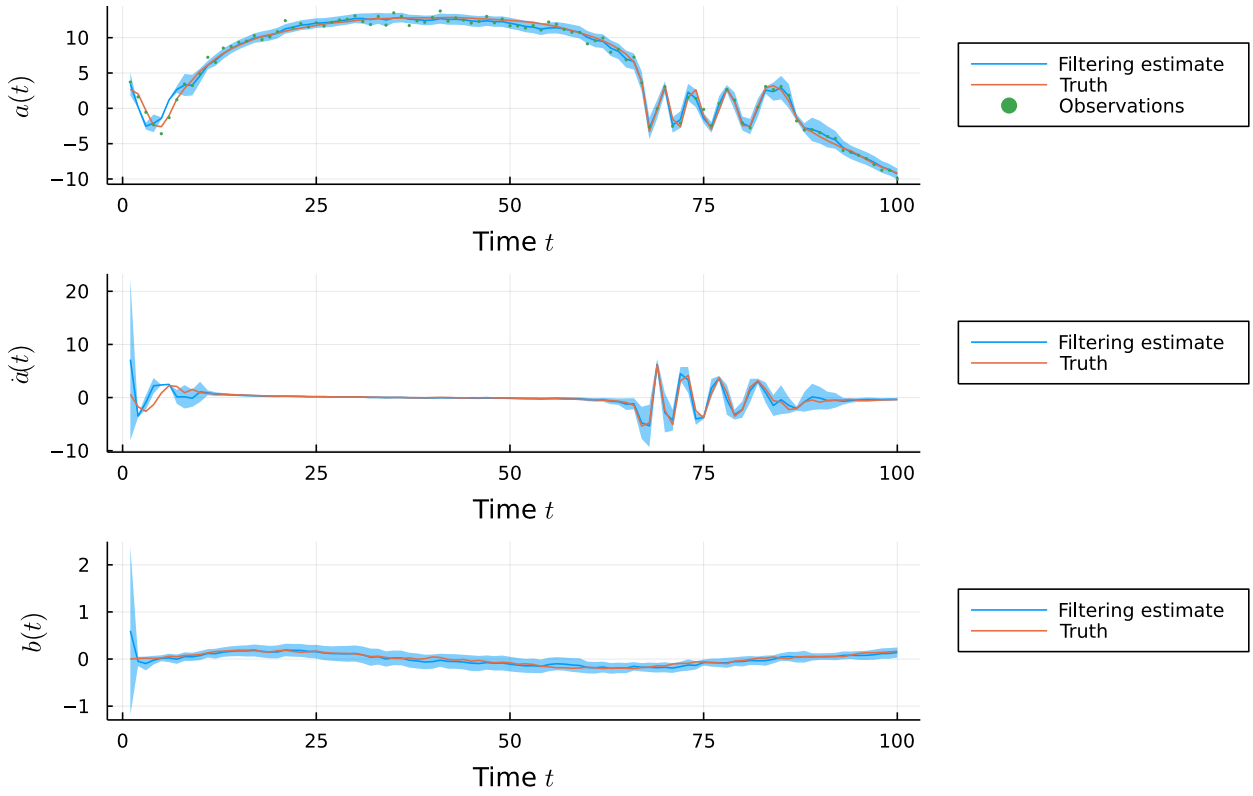


Figure 28: True state trajectories (orange), observed data (green markers) and filtering estimates (blue) for stochastic ANAET model using particle filter with $P = 250$ and locally optimal proposals.

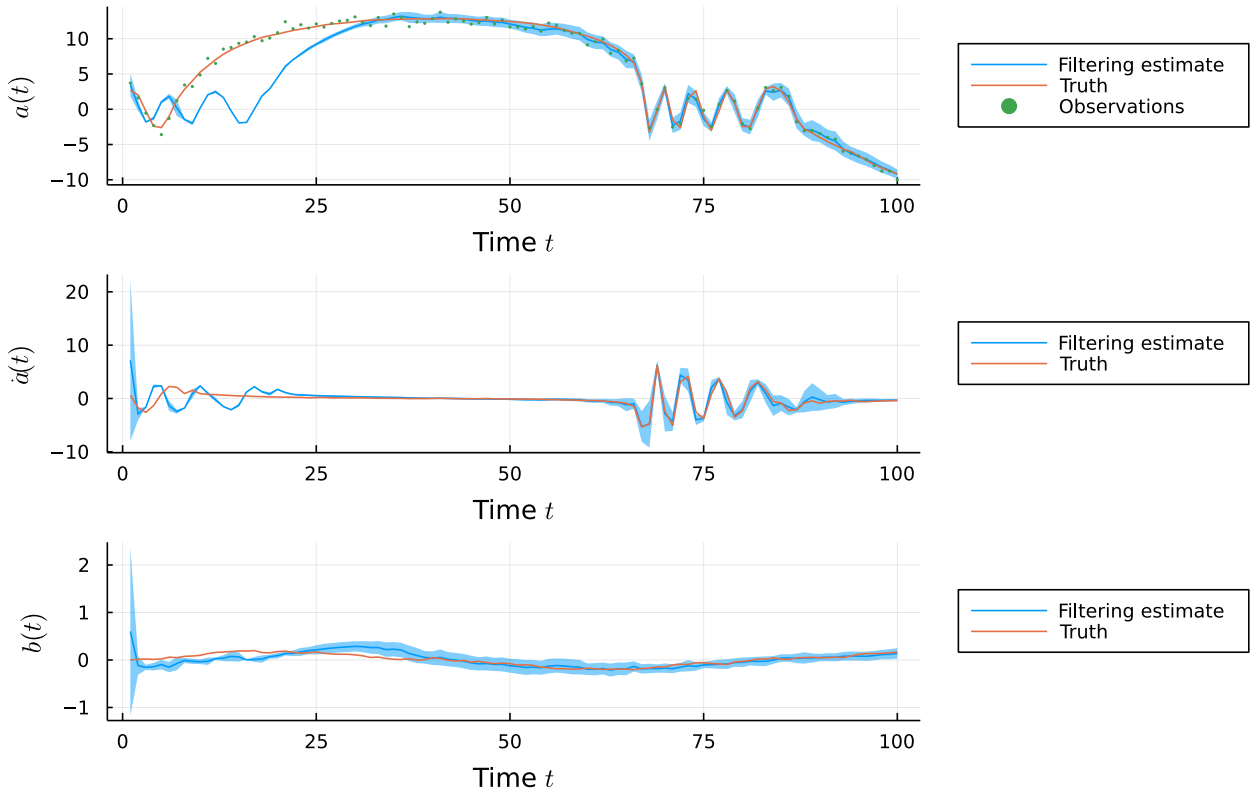


Figure 29: True state trajectories (orange), observed data (green markers) and filtering estimates (blue) for stochastic ANAET model using particle filter with $P = 250$ and bootstrap proposals.

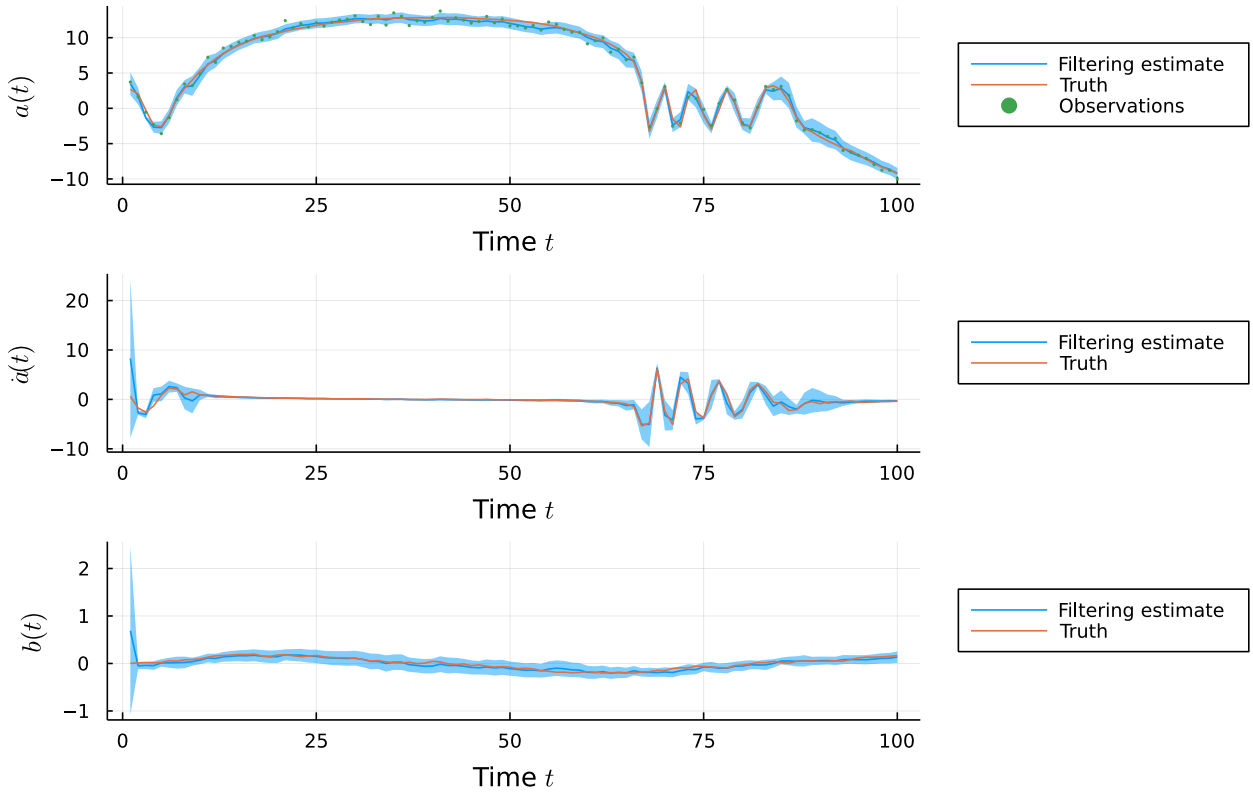


Figure 30: True state trajectories (orange), observed data (green markers) and filtering estimates (blue) for stochastic ANAET model using particle filter with $P = 500$ and locally optimal proposals.

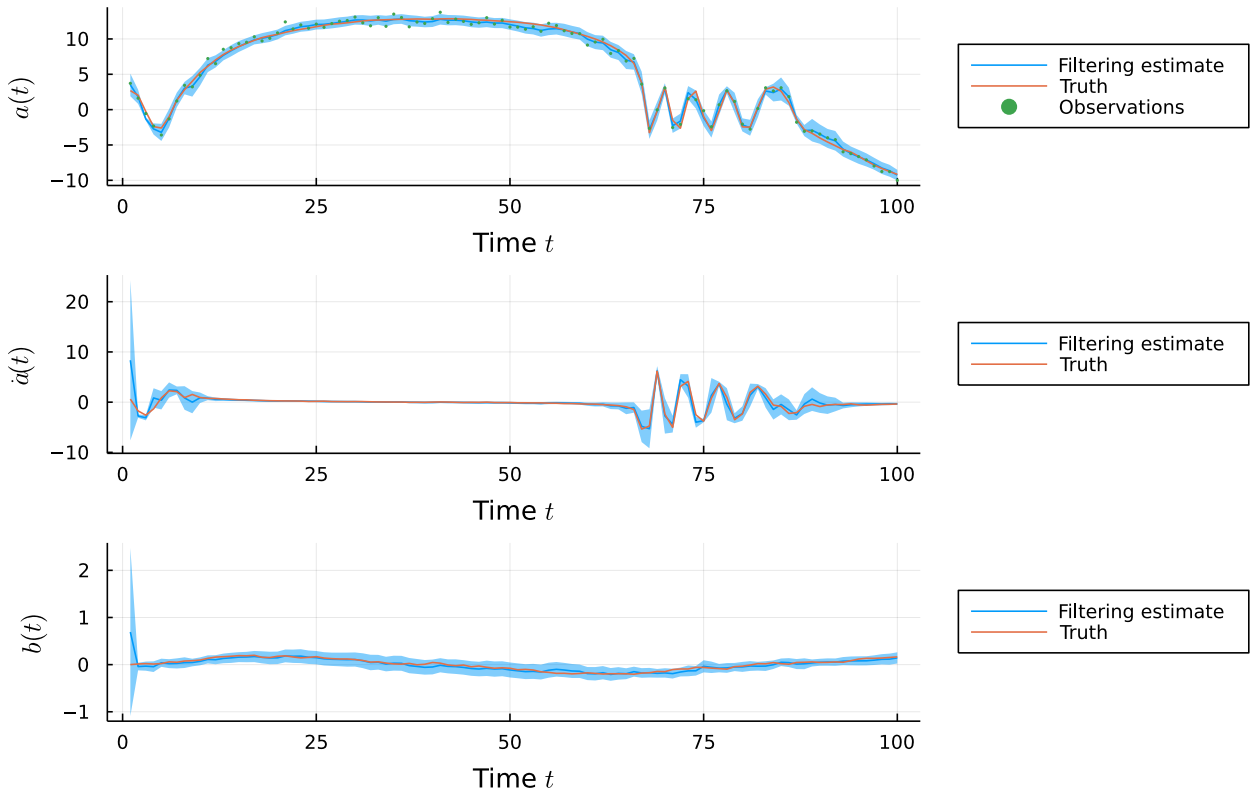


Figure 31: True state trajectories (orange), observed data (green markers) and filtering estimates (blue) for stochastic ANAET model using particle filter with $P = 500$ and bootstrap proposals.

regular quadrilateral mesh used. The time discretization is performed using an explicit fourth-order Runge–Kutta method, with fixed time step.

To formulate as a state space model, we define the system state as the concatenation of the coefficients of the spectral element expansions of the fields ζ (vorticity) and n (number density), with the electrostatic potential ϕ being fully determined by ζ. The state dimension is therefore equal to $d_x = 2m_1m_2(p+1)^2$ where m_1 is the number of quadrilateral elements along the s_1 spatial coordinate, m_2 is the number of quadrilateral elements along the s_2 spatial coordinate and p is the polynomial order of the spectral element expansion.

To introduce stochasticity in the state dynamics, after using the `nektar-driftwave` solver to simulate the Hasegawa–Wakatani equations forward in time by a fixed inter-observation time interval $\Delta = \tau_t - \tau_{t-1}$, we perturb the ζ and n state fields by adding zero-mean stationary Gaussian random fields with Matérn covariance function

$$C(s_1, s_2 \mid \sigma, \ell, \nu) = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{s_1^2 + s_2^2}}{\ell} \right)^{\nu} K_\nu \left(\frac{\sqrt{s_1^2 + s_2^2}}{\ell} \right)$$

where σ^2 is the marginal variance, ℓ a lengthscale parameter and ν a smoothness parameter, Γ the gamma function and K_ν the modified Bessel function of the second kind of order ν. By simulating Gaussian random fields with appropriate choices of the covariance parameters (σ^2, ℓ, ν) on the spatial domain of the problem and with same boundary conditions as the state field variables ζ and n, we can produce perturbations of the state fields which conserve key aspects of the fields such as their smoothness properties and boundary conditions. To simulate such Gaussian random fields, we use that they correspond to (scaled) solutions $\sigma\sqrt{4\pi\nu\ell^{-2\nu}}u$ to the *stochastic partial differential equation* (SPDE)

$$(\ell^{-2} - \partial_1^2 - \partial_2^2)^{\frac{\nu+1}{2}} u(s_1, s_2) = \mathcal{W}(s_1, s_2),$$

where $\mathcal{W}$ is a Gaussian white noise process, as first observed by (Whittle 1954) and popularized in spatial statistics by (Lindgren, Rue, and Lindström 2011). For $\nu = 2m - 1$, $m \in \mathbb{N}$ we additionally

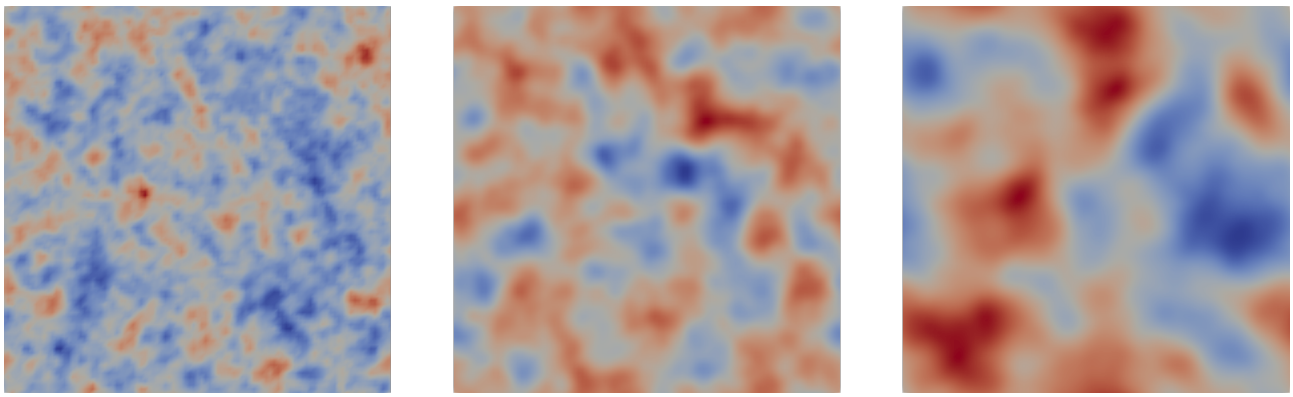


Figure 32: Examples of Gaussian random fields Matérn covariance generated using Nektar++ by solving a SPDE. All fields are on a 40×40 spatial domain with periodic boundary conditions. From left to right the covariance function lengthscale and smoothness parameters used where $(\ell = 1, \nu = 1)$, $(\ell = 1, \nu = 3)$ and $(\ell = 2, \nu = 3)$.

have that $u = v^{(m)}$ where $v^{(m)}$ is recursively defined by

$$(\ell^{-2} - \partial_1^2 - \partial_2^2)v^{(m)}(s_1, s_2) = v^{(m-1)}(s_1, s_2),$$

with base case $v^{(0)} = \mathcal{W}$. To simulate Gaussian random fields with Matérn covariance on a spatial domain in Nektar++, we can therefore iteratively solve a Helmholtz equation using the advection–diffusion–reaction solver, with forcing (righthand side) function set to Gaussian white noise for the base case, and the field output of the previous iterations for subsequent iterations. We specify the same mesh and boundary conditions as for the fields used in the `nektar-driftwave` solver for the deterministic component of the dynamics. Figure 32 shows examples of Gaussian random fields generated using different values for the lengthscale ℓ and smoothness ν parameters.

The observed data at each observation time $\tau_t = t\Delta$ is then assumed to correspond to noisy observations of the ζ field at a set of spatial points $(s_1^{(i)}, s_2^{(i)})_{i=1}^{d_y}$, that is

$$\boldsymbol{y}_t \sim \mathsf{Normal}\left(\cdot \mid \left(\zeta(s_1^{(i)}, s_2^{(i)}, t\Delta)\right)_{i=1}^{d_y}, \sigma_y^2 \mathbf{I}\right).$$

The code for the Julia implementation of the model, along with installation instructions and example usages with ParticleDA is available at `https://github.com/Team-RADDISH/ParticleDA-UseCases/tree/main/NektarDriftwave`.

To test the scaling of the ParticleDA filter implementation on a HPC system, we ran multiple filtering runs with differing combinations of thread and MPI based parallelism on ARCHER2, the UK national supercomputing service. ARCHER2 provides a cluster of 5860 compute nodes each with dual 64-core processors, and so 128 cores per node.

We first simulated a fixed sequence of $T = 100$ observations from the NektarDriftwave state space model, using a quadrilateral mesh of dimensions $m_1 = 32$, $m_2 = 32$ on a spatial domain $[-20, 20] \times [-20, 20]$ with periodic boundary conditions. The polynomial order used in the spectral expansion was $p = 3$, resulting in an overall state dimension of $d_x = 32768$. The ζ field was observed at $d_y = 9$ points $(-10, -10)$, $(0, -10)$, $(10, -10)$, $(-10, 0)$, $(0, 0)$, $(10, 0)$, $(-10, 10)$, $(0, 10)$, $(10, 10)$ with observation noise standard deviation $\sigma_y = 0.1$. The zero-mean independent Gaussian random fields used to perturb the ζ and n state fields between each observation time used a Matérn covariance function with parameters $\nu = 3$, $\ell = 1$ and $\sigma = 0.05$, with a fixed inter-observation interval of $\Delta = 0.5$ and a time integration step size for the `nektar-driftwave` solver of 0.0005. The adiabacity operator and background density gradient lengthscale parameters of the Hasegawa–Wakatani equations were set to respectively $\alpha = 2$ and $\kappa = 1$. The state was initialized with independent Gaussian random fields for ζ and n with the same covariance function and parameters as the state noise and means

$$\mathbb{E}[\zeta(s_1, s_2, 0)] = \exp\left(-(s_1^2 + s_2^2)/4\right)(s_1^2 + s_2^2 - 4)/4,$$
$$\mathbb{E}[n(s_1, s_2, 0)] = \exp\left(-(s_1^2 + s_2^2)/4\right).$$

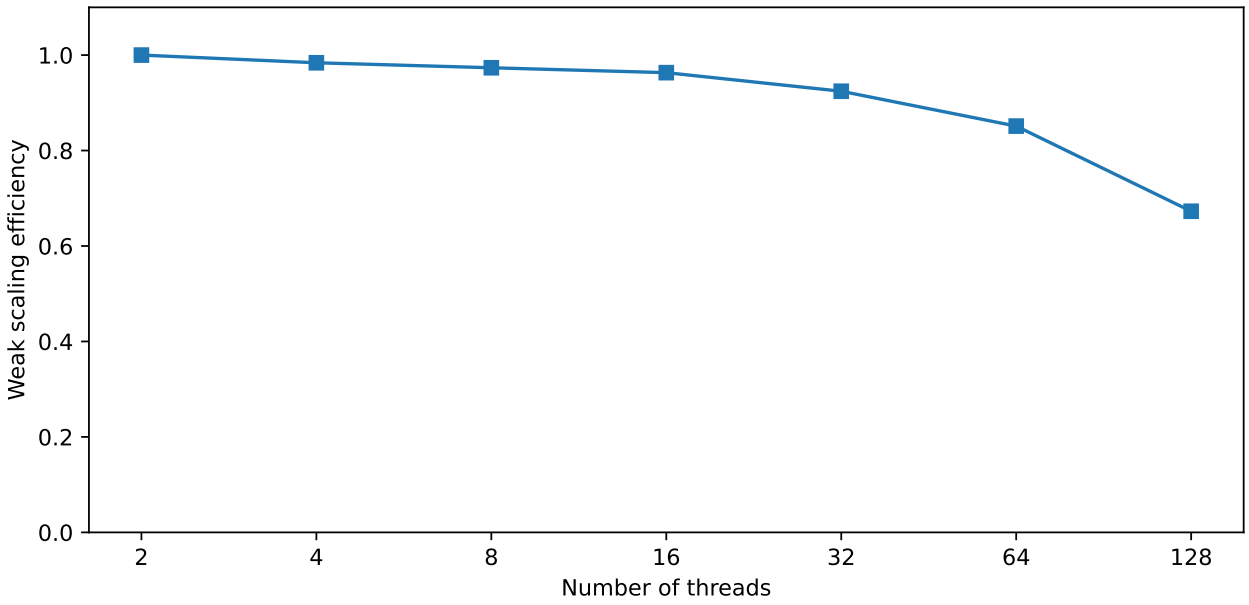


Figure 33: Weak scaling efficiency for thread-based parallelization of particle filter based data assimilation using ParticleDA with NektarDriftwave state space model on a single ARCHER2 node

We first consider how using multiple threads to parallelize the particle updates on a single ARCHER2 node performs, running a particle filter with bootstrap proposals and one particle per thread. We filter using the sequence of $T = 100$ observations simulated as described above, using the same model parameters for filtering as used for generating the observations. Figure 33 shows the *weak scaling efficiency* for varying number of threads, with weak scaling measuring parallel performance as both the compute resource (number of threads here) and problem size (number of particles in ensemble) are increased. Here we specifically defined the weak scaling efficiency as the (compute time on for 2 particles on 2 threads) / (compute time for N particles on N threads). We can see that the weak scaling efficiency shows a slow but consistent drop off as the number of threads increases, with an efficiency of 0.96 at 16 threads and 0.67 at 128 threads. For reference the compute time with 2 threads was 6590s.

As each compute node on ARCHER2 has 128 cores, if the application was compute bound we would expect to see the efficiency remain close to unity up to 128 threads. The drop-off here suggests the performance bottleneck on a single node is not processor availability. The NektarDriftwave state space model implementation uses file-based input / output to interact with the Nektar++ solvers used to solve both the deterministic dynamics and generate the Gaussian random fields used to perturb the states, resulting in multiple writes and reads of the state to and from HDF5 files on disk for each particle and at each filtering step. As all threads on a single node will be competing for the same bandwidth for reading from and writing to disk, we believe this may be the performance bottleneck in this case. To avoid this bottleneck, it would be preferable to share memory for storing the states between the ParticleDA Julia code and Nektar++ C++ code to avoid the round-tripping to and from disk, however we did not have time to explore this option.

ParticleDA also supports distributing the particle updates using MPI, allowing us to test scaling when using multiple ARCHER2 nodes. We first again consider weak scaling efficiency, using 128 particles and MPI ranks per node (one per core). Figure 34 show the weak scaling efficiency for varying number of nodes, here defined as the (compute time for 128 particles on 1 node) / (compute time for $128n$ particles on n nodes). Unlike the multi-threading case, the weak scaling efficiency remains close to unity as we increase the number of nodes, dropping only to 0.96 at 32 nodes, the maximum tested here, with again for reference the compute time on a single node here 8890s.

We speculate that the improved scaling efficiency observed here compared to the multi-threading case is due to the distribution of the computation across multiple nodes increasing both the number

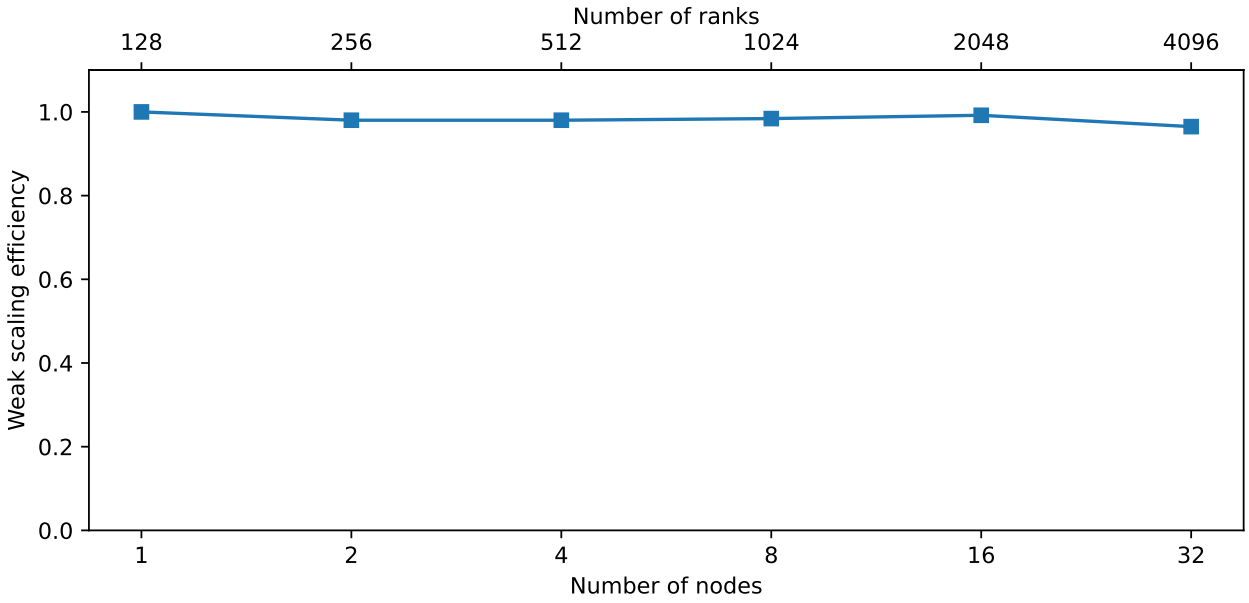


Figure 34: Weak scaling efficiency for MPI-based parallelization of particle filter based data assimilation using ParticleDA with NektarDriftwave state space model on ARCHER2.

of processors available but also bandwidth for reading and writing to disk. We also note that the multi-node weak scaling performance observed here is significantly better than previously shown in (Giles et al. 2023) for filtering runs with another state space model (a two-dimensional linear long wave model of tsunamis) on ARCHER2, where the weak scaling efficiency dropped off sharply when distributing across multiple nodes.

This can be explained by the relative cost of the proposal and resampling steps between the two models, with the former (embarrassingly parallelizable) step dominating the run time for the model here (the proposal steps constituted 96.5% of the run time on one node and 94.1% on 32 nodes). In comparison the state updates in the tsunami model were much cheaper to compute and the total ensemble sizes used much larger, resulting in a much higher proportion of time being spent in the resampling step, which due to the requirement for communication across ranks becomes more costly as the number of nodes (and ranks) is increased. As the plasma physics models of interest to NEPTUNE will be better reflected by the computational demands of the NektarDriftwave example here, we believe the regime of interest will generally be one were the cost of the model simulation in the proposal steps dominate, and the results here indicate ParticleDA should be able to achieve high scaling efficiency in this setting.

As a final performance benchmark, we also measured the *strong scaling* performance of particle filtering with ParticleDA on the NektarDriftwave model, which measures the speed-up achieved as the problem size (number of particles) is held constant and the compute resource (number of processors) increased. We ran particle filters (with bootstrap proposals) on the same $T = 100$ simulated observation sequence as previously with a fixed ensemble size of $N = 1024$ particles, distributing the filtering updates using MPI across one or more nodes with 128 ranks per node (one per core). Figure 35 shows the strong scaling speed-ups (compute time for $N = 1024$ particles on n nodes / compute time for $N = 1024$ particles on 1 node) measured for runs on up to 8 nodes. We see that we achieve close to ideal of a linear speed-up as we increase the the number of nodes, suggesting ParticleDA is also able to achieve good strong scaling for this model.

The previous results related to the computational performance and scalability of the particle filter algorithms implemented in ParticleDA. Another import metric is how they perform statistically, in terms of the accuracy of their estimation of the true filtering distributions. As we have no way of computing the true filtering distributions for the state space model here, we cannot easily directly measure the estimation performance. An indicative proxy for whether a particle filter is likely to

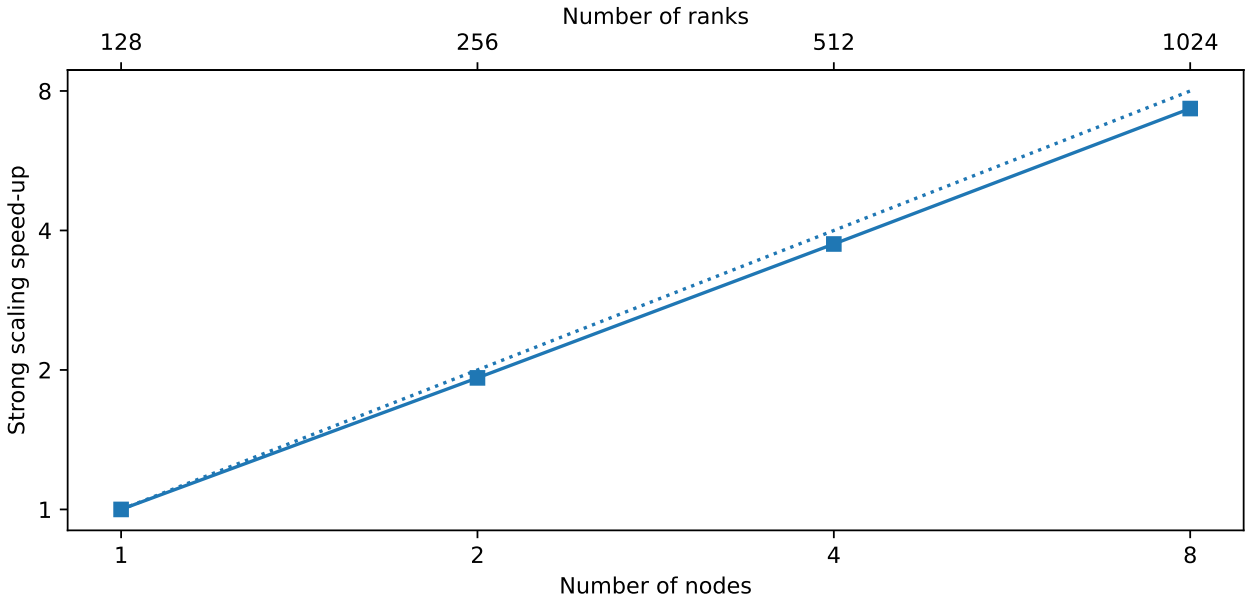


Figure 35: Strong scaling efficiency for MPI-based parallelization of particle filter based data assimilation using ParticleDA with NektarDriftwave state space model on ARCHER2. Dotted line shows ideal linear speed-up.

Number of particles N	$\min_t \widehat{\mathsf{ESS}}(t, N)$	$\text{median}_t \widehat{\mathsf{ESS}}(t, N)$	$\max_t \widehat{\mathsf{ESS}}(t, N)$
128	3.17	66.5	120
256	6.16	142	244
512	3.99	251	477
1024	3.44	486	940
2048	23.5	954	1800
4096	2.54	1580	3810

Table 3: Estimated *effective sample size* (ESS) statistics for particle filter runs on NektarDriftwave model with bootstrap proposals for varying ensemble size N.

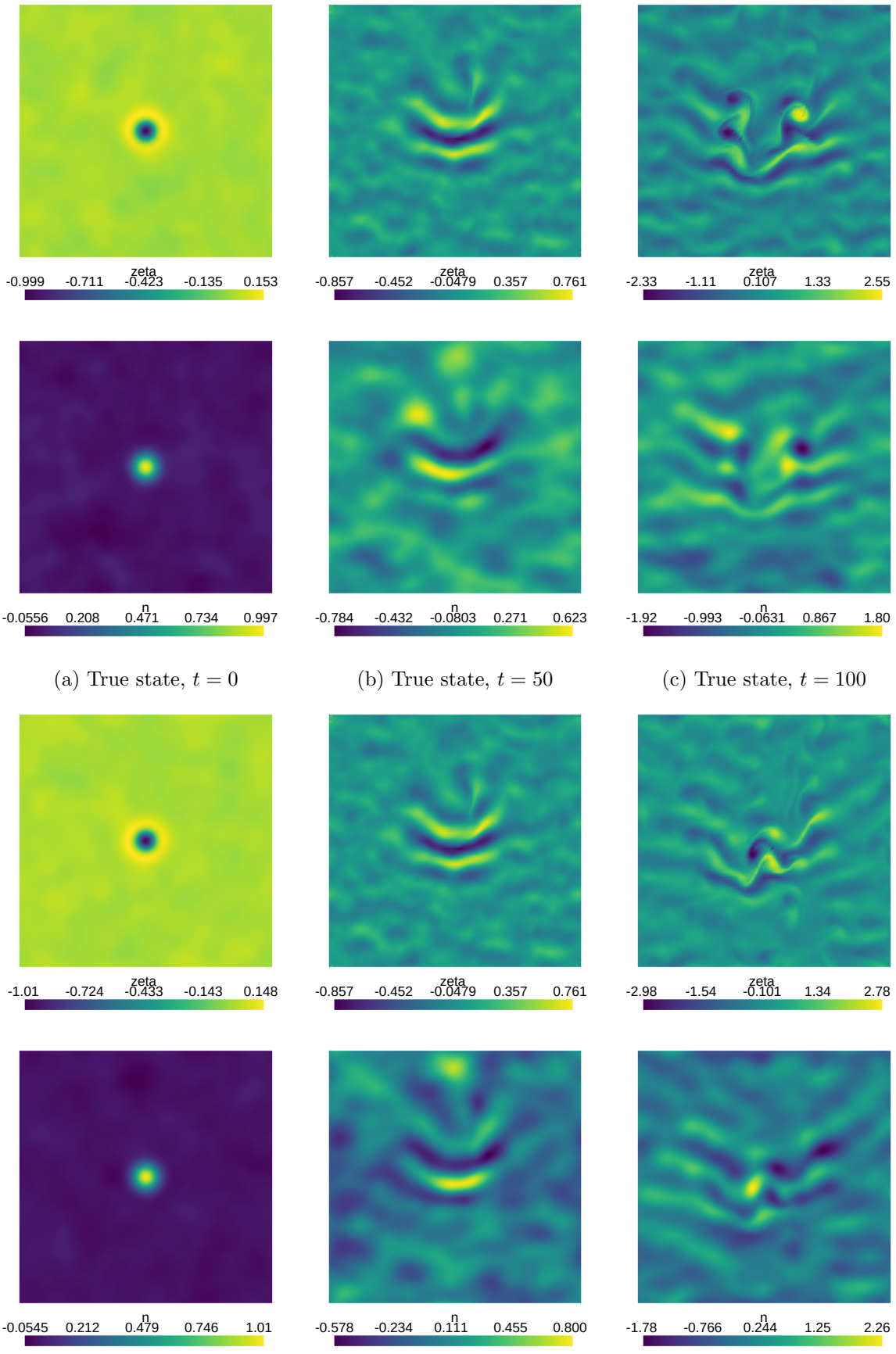
be giving poor estimates of the filtering distributions is the *effective sample size* (ESS), a measure of the number of independent samples from the filtering distribution that would give a Monte Carlo estimator with equivalent variance to that computed using the importance weighted samples from the particle filter. An estimate of the ESS for the filtering distribution at time index t can be computed from the particle importance weights as

$$\widehat{\mathsf{ESS}}(t, N) = 1 / \sum_{n=1}^{N} \left(w_t^{(n)} \right)^2 .$$

Table 3 shows summary statistics of the ESS estimates computed from runs of the bootstrap particle filter on the NektarDriftwave model (using the same $T = 100$ simulated observation sequence and model set up as previously) for varying ensembles sizes N. The ideal case would be if the ESS estimates remained close to the ensemble size N. Low ESS estimates are instead indicative of high variance in the importance weights and a possible tendency to the ensemble collapse degeneracy discussed previously.

Here we see that the minimum ESS estimates across the filtering runs are consistently low for all ensemble sizes tested, suggesting the filtering distribution estimates from the bootstrap particle filter used here will have high variance. This is also evident when comparing the particle filter estimates of the mean of the filtering distributions to the true state fields used to generate the observations used for filtering, with the latter corresponding to samples from the true filtering distributions in this simulated data setting. The true state fields and corresponding estimates of the filtering distribution means are shown for three time indices $t \in (0, 50, 100)$ in Figure 36, with the particle filter estimates here computed using an ensemble size of $N = 2048$. It can be seen that the true and estimated mean state fields differ significantly at later time indices, and while the filtering distribution means would not be expected to exactly match the true states, the large differences here and small estimated filtering distribution variances (not shown) are indicative of weight degeneracy causing the ensemble to collapse and no longer 'track' the true state, as also indicated by the low minimum ESS estimates.

The poor statistical performance here is not unexpected as the dimensionality of the state space here is relatively large $d_x = 32768$, and while the observation dimension $d_y = 9$ is small the signal to noise ratio in the observations is relatively high. While we may be able to achieve somewhat improved performance using the locally optimal proposal, while the current formulation of the NektarDriftwave state space model technically should allow tractable computation of the locally optimal proposal distribution, this is not yet supported in the current implementation. More generally, as noted previously to achieve robust statistical performance on complex high-dimensional state space models, additional approaches such as tempering and spatial localization are likely to be required. Implementing support for such approaches in ParticleDA would be an important avenue to consider in any future work.



(a) True state, $t = 0$ (b) True state, $t = 50$ (c) True state, $t = 100$

(d) Estimated state, $t = 0$ (e) Estimated state, $t = 50$ (f) Estimated state, $t = 100$

Figure 36: True state fields at three time indices $t \in (0, 50, 100)$ for NektarDriftwave state space model and corresponding particle filter estimates of the mean of the filtering distributions using an ensemble with $N = 2048$ particles.

11 FabNESO

As part of UCL's work on AQUIFER, we developed *FabNESO* (`https://github.com/UCL/fabneso`) a *NEPTUNE exploratory software* (NESO) plug-in for *FabSim3* (Groen et al. 2023). FabNESO is designed to facilitate the execution of NESO simulations on both local and remote high performance computing systems via a unified interface. It defines a series of *tasks* to support different applications and analyses with NESO simulations, with tasks defined for running single and ensembles of simulations, forward uncertainty quantification using *EasyVVUQ* (Richardson et al. 2020) and accelerated Bayesian calibration of model parameters using *PyVBMC* (Huggins et al. 2023). Instructions for installing and configuring FabNESO are provided on the repository README and HTML documentation for the interface for all tasks and utility functions is provided at `http://github-pages.ucl.ac.uk/FabNESO/`.

FabNESO can be used with any solver executable provided by NESO (or Nektar++ more generally) which follows the standard Nektar++ solver command line interface — that is it will accept as arguments paths to one or more XML files specifying the conditions of the equation system to solve and geometry and mesh discretization of the spatial domain to solve on. Tasks take a `solver` argument which can be used to specify the name of the solver executable within the path specified by the (machine specific) `neso_bin_dir` configuration option, which should point to the local path containing the built NESO binaries. Pre-defined configurations are provided for the `Electrostatic2D3V` and `H3LAPD` solvers based on respectively the `two_stream` and `2Din3D-hw` examples for these solvers in the main NESO repository, with these configuration directories containing the conditions and mesh XML files to use, with tasks taking a `config` argument to specify the configuration to use. Users can add additional configurations by adding further sub-directories with the necessary files to the FabNESO plugin `config_files` sub-directory. Most tasks also provide some support for overriding the default values of the parameters specified in these configuration files - either directly for single simulations, or as part of a scan over multiple parameter values as part of ensembles of runs for uncertainty quantification or calibration.

11.1 Forward uncertainty quantification - `neso_pce_ensemble` task

The `neso_pce_ensemble` task allows running an ensemble of NESO simulations for performing a PCE of one or more (functional) outputs of the solver, building on the support for PCE in EasyVVUQ and *chaospy* (Feinberg and Langtangen 2015). The results of the PCE ensemble can then be analysed to perform forward uncertainty quantification on the outputs and sensitivity analyses.

For example the shell command

```
fabsim archer2 neso_pce_ensemble:\
    config=2Din3D-hw,solver=H3LAPD,\
    nodes=4,processes=512,\
    polynomial_order=3,variant=pseudo-spectral,\
    HW_alpha=0:2,Te_eV=5:20
```

would schedule an ensemble of `(polynomial_order + 1)**n_parameters` $= (3+1)^2 = 16$ jobs on ARCHER2 running H3LAPD solver instances with parameter values corresponding to quadrature nodes over a uniform distribution on the two-dimensional parameter space spanned by `HW_alpha` $\in [0, 2]$ and `Te_eV` $\in [5, 20]$, with the specific PCE variant here being *pseudo-spectral projection*. Each ARCHER2 jobs would be scheduled to run on 512 MPI ranks (processes) across 4 nodes. Once this command is executed, we can use `fabsim archer2 job_stat` to check on the job statuses and `fabsim archer2 fetch_results` to pull the job results back to our local machine once the jobs have completed.

A separate `neso_pce_analysis` task is provided for analysing the outputs of a previous PCE ensemble task run. For example if the output of the `fetch_results` task indicated the results had been pulled locally to the directory `/path/to/results` we could analyse the outputs of the previous `neso_pce_ensemble` task on our local system by running

```
fabsim localhost neso_pce_analysis:config=2Din3D-hw,results_dir=/path/to/results
```

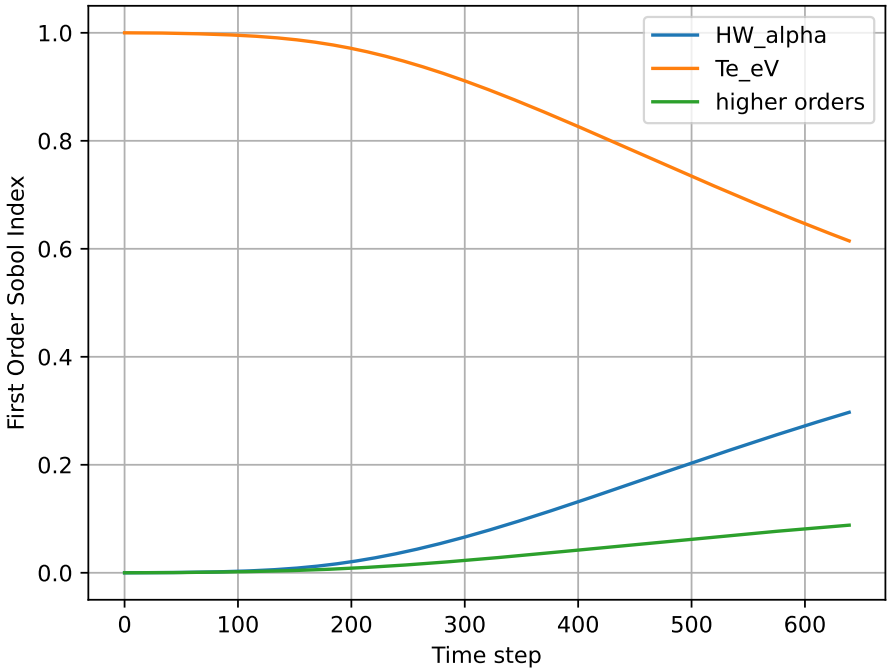


Figure 37: Estimated first order Sobol indices of $E(t)$ outputted by H3LAPD solver run using `2Din3D-hw` example configuration with respect to the parameters `HW_alpha` and `Te_eV`.

with then outputting a pickle file containing a serialized version of the EasyVVUQ PCE analysis results object corresponding to the ensemble. This can then be used to compute various statistics of the model outputs, construct a surrogate model or perform sensitivity analyses of the outputs with respect to the parameters. Example of output plots produced using the PCE analysis results object for a PCE ensemble run executed as shown above on ARCHER2 are shown in Figures 37 and 38.

11.2 Calibrating model parameters - `neso_vbmc` task

The `neso_vbmc` task allows accelerated Bayesian calibration of the parameters of NESO solvers using *variational Bayesian Monte Carlo* (VBMC) (Acerbi 2018), specifically using the Python implementation in PyVBMC (Huggins et al. 2023). The VBMC algorithm is an approximate Bayesian inference method for efficient estimation of the posterior on the parameters of computationally demanding simulator models given observations of the model outputs. Compared to alternative approximate inference approaches such asMCMC, it allows robust estimation of the posterior over moderately dimensioned parameter spaces (up to around 10) with typically far fewer model evaluations (often in the low hundreds rather than tens of thousands typical of MCMC methods) and treating the model as a 'black-box' - for example not requiring the ability to compute derivatives with respect to the model parameters. These characteristics make it well suited the calibration of NESO simulations, which are expensive to evaluate, have no gradient information available and (currently) have a relatively small number of parameters we may wish to calibrate.

VBMC combines variational inference, an efficient approach for fitting an approximation to target posterior distribution from a prescribed parametric family, with Bayesian quadrature, an approach for estimating the value of integrals where the integrand is expensive to evaluate by computing a Gaussian process surrogate. It employs a iterative sequential design strategy, alternating between: actively sampling new parameter values to evaluate the log posterior density (model) at based on a current Gaussian process model for the log posterior, balancing exploration of areas of the parameter space where we are uncertain about the log posterior density with exploiting of regions containing high posterior mass; updating the Gaussian process surrogate for the log density based on the newly

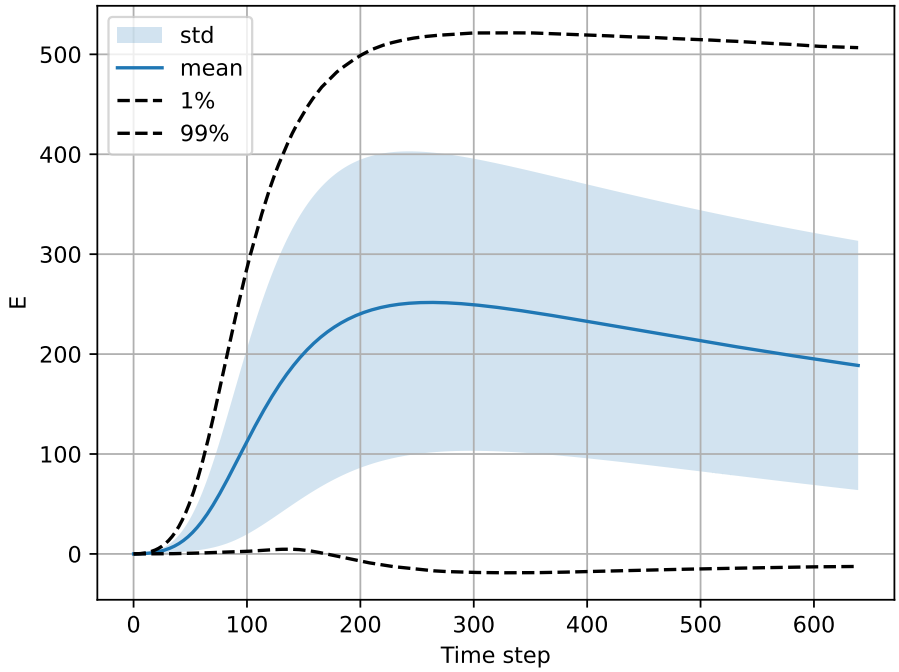


Figure 38: Estimated moments of energy trajectory $E(t)$ outputted by H3LAPD solver run using `2Din3D-hw` example configuration with a uniform distribution over the parameters `HW_alpha` and `Te_eV`.

acquired log posterior density values; updating the variational approximation to the target posterior using the Gaussian process surrogate.

The `neso_vbmc` task currently supports calibration of the `Electrostatic2D3V` solver, with a pre-defined function for extracting the values of the simulated electrostatic potential field along a line at the final simulation time from the HDF5 files output by the solver provided. Integration with additional solvers would require identification of suitable outputs to calibrate against and implementation of a similar function to extract the necessary values from the solver outputs. The task can be run locally by specifying the `localhost` machine in the `fabsim` command or on a remote system such as ARCHER2 (specified using `archer2` machine argument). The following command

```
fabsim archer2 neso_vbmc:\
    config=two_stream,solver=Electrostatic2D3V,\
    processes=16,reference_field_file=observations.txt,\
    particle_initial_velocity=0:2:0.5:1.5,\
    particle_charge_density=0:200:50:150,\
    particle_number_density=0:200:50:150
```

would run PyVBMC to estimate the posterior on the parameters of the Electrostatic2D3V solver `particle_initial_velocity`, `particle_charge_density` and `particle_number_density` given a set of (noisy) observations of the final electrostatic potential field along a line recorded in a data file `observations.txt`. The colon delimited lists after the parameter names specify the bounds of the parameters (potentially infinite) and 'plausible' bounds specifying a range over which it is believed likely most of the posterior mass lies. The task here would sequentially schedule jobs running the solver with the actively sampled parameter values on ARCHER2 (with the example command here specifying to run each job on 16 MPI ranks), waiting for the completion of the job and computation of the corresponding log posterior density value before scheduling a new run at the next actively sampled parameter set.

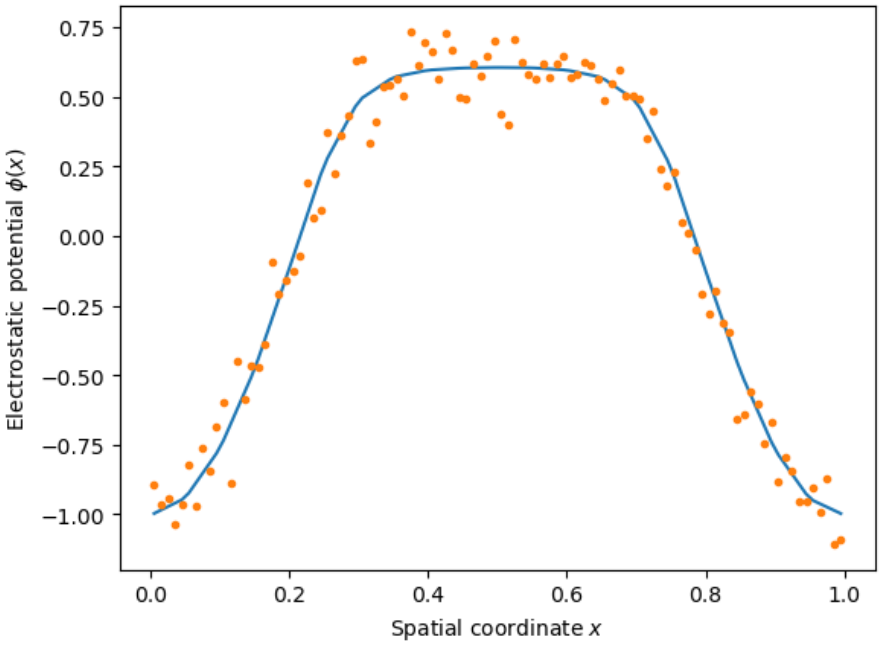


Figure 39: Simulated noisy observations (orange points) and true underlying values (blue curve), for electrostatic potential field along line at final timestep of a simulation using NESO `Electrostatic2D3V` solver with `two_stream` example configuration.

Once PyVBMC has identified certain convergence criteria have been met, the final variational posterior approximation is returned to allow use in subsequent analyses. The `neso_vbmc` task saves a serialization of this variational posterior object to a pickle file.

Figure 39 shows an example set of simulated observations of the final electrostatic potential field along a line at 100 regularly spaced points, generated using parameters `particle_initial_velocity` = 0.96, `particle_charge_density` = 99, and `particle_number_density` = 103, with independent Gaussian observation noise with standard deviation 0.1.

Figure 40 shows the posterior estimate on the parameters parameters `particle_initial_velocity`, `particle_charge_density` and `particle_number_density` of the `Electrostatic2D3V` NESO solver with `two_stream` example configuration, produced by PyVBMC for the simulated observations in Figure 39. The parameters bounds were set as specified in the `neso_vbmc` task command example above, with a uniform distribution assumed over the parameter space. In this case the VBMC algorithm converged after evaluation of the model at 130 different set of parameter values, with the true parameter values used to simulate the observations contained within the bulk of the estimated posterior mass as expected. With a relatively limited number of model evaluations the VBMC algorithm here has been able to capture a plausible estimate of the posterior distribution.

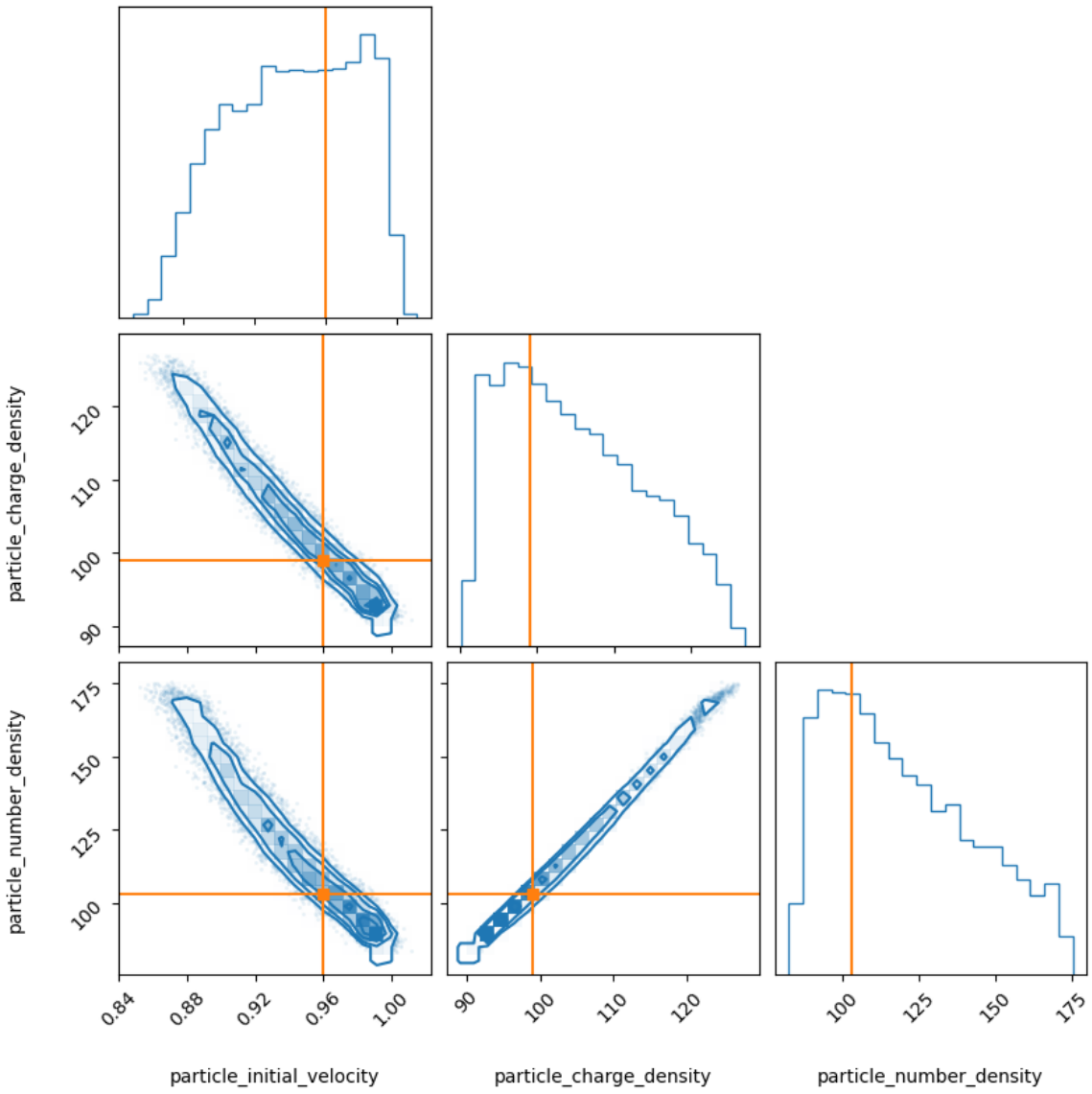


Figure 40: Estimated posterior on the three parameters `particle_initial_velocity`, `particle_charge_density` and `particle_number_density` (other parameters fixed) of the NESO `Electrostatic2D3V` solver with the `two_stream` example configuration, given the simulated observations show in Figure 39, estimated using PyVBMC using 130 evaluations of the model. Posterior density estimate is shown in blue and true parameter values used to generate the observations in orange.

12 Additional Bayesian calibration tools

In addition to the FabNESO integration with PyVBMC, we also developed a basic Python package `nesopy` , available in the repository `https://github.com/UCL/neso-calibration`, which provides a simple interface for executing NESO solvers with specified parameter values and extracting the generated outputs, to allow easy integration with existing calibration tools such as PyVBMC. The NESO solvers used can either be built natively on the local system or within a Docker container, with `nesopy` providing wrappers for both options. A notebook illustrating how to use the `nesopy` Python wrapper to perform calibration of a NESO simulation with PyVBMC is provided in the `neso-calibration` repository under `examples/two_stream/pyvbmc_calibration_example.ipynb`.

A Python implementation of an alternative calibration approach exploiting Gaussian process emulation of the posterior density was also developed during AQUIFER and is available at `https://github.com/UCL/calibr`, with the initial intention of exploring this as an alternative to PyVBMC for calibration of NESO simulations, though time constraints meant that this work was not completed. `calibr` is a Python implementation of the algorithm described in *Parallel Gaussian process surrogate Bayesian inference with noisy likelihood evaluations* (Järvenpää et al. 2021). It is designed to allow estimation of the posterior distribution on the unknown parameters of expensive to evaluate simulator models given observed data, using a batch sequential design strategy which iterates fitting a Gaussian process emulator to a set of evaluations of the (unnormalized) posterior density for the model and using the emulator to identify a new batch of model parameters at which to evaluate the posterior density which minimize a measure of the expected uncertainty in the emulation of the posterior density.

The posterior density can be evaluated at the parameter values in each batch in parallel, providing the opportunity for speeding up calibration runs on multi-core and multi-node high performance computing systems. The acquisition functions used to choose new parameter values to evaluate are implemented using the high-performance numerical computing framework *JAX*, with the gradient-based optimization of these acquisition functions exploiting JAX's support for automatic differentiation.

13 Software implementation and HPC deployment

Note than Archer2 cycles were provided to UKAEA, and to two of the non-UCL NEPTUNE grantees from the SEAVEA allocation.

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 *data assimilation* (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.

13.1 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 10 times 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. Right now, we've been allocated 54 000 *computing unit*s (CUs) hours on ARCHER2. One CU indicate a

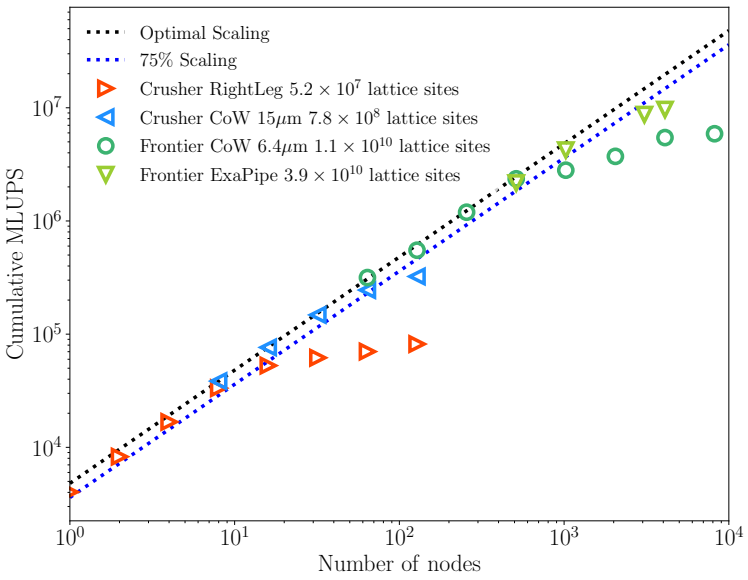


Figure 41: Performance scaling plot of HemeLB Mazzeo and Coveney 2008 deploys on Frontier and Crusher.

node with 128 processor cores running for one hour. 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.

13.1.1 HemeLB: Performance benchmark on Frontier

HemeLB-GPU[5] has undergone extensive testing on both the Frontier and Crusher machines. The performance tests conducted on these machines provide valuable insights into the scalability of HemeLB. Figure 29 illustrates the strong scaling performance test results of HemeLB-GPU (pressure boundary conditions) using different vascular models on both Frontier and Crusher. Three vascular models were tested: the right leg vascular model and the circle of Willis vascular model at resolutions of 15 μm and 6.4 μm, respectively. These models consist of lattice sites ranging from $5.2x10^7$ to $1.1x10^10$ and ran for 10,000 time-steps. To exploit the potential for exascale computing here, we have further tested pipe flow scaling with $3.9x10^10$ lattice sites. In the optimal scaling region, HemeLB demonstrated a single-node performance of approximately 4000 *million lattice cell updates per second* (MLUPS). This performance exhibited linear strong scaling with the number of nodes until degradation sets in due to the problem size. We have also demonstrated continued strong scaling behaviour on up to 87% of the full Frontier system (8192 nodes).

13.1.2 HemeLB: Validation of turbulent model using large-eddy simulation techniques

To validate the channel flow simulation, a high-resolution stenotic channel flow simulation is established. As depicted in Fig. 42, the dimensions of the stenotic channel flow are configured as $L_x \times L_y \times L_z$, where $L_x = 266\delta$, $L_y = 22\delta$, and $L_z = 2\delta$ correspond to the streamwise, spanwise, and vertical directions, respectively. Here, δ denotes the turbulent boundary layer thickness, set at $\delta = 0.0045m$. The lattice resolution is set to $100\mu m$, resulting in a simulation domain composed of approximately 1.0×10^9 lattice cells.

In Fig. 43, the experimental data begins from $y^+ = 10$, whereas the y^+ for the first cell near the wall in the *lattice Boltzmann method* (LBM)–*large-eddy simulation* (LES) simulation is approximately $y^+ = 1.5$. Although there is a minor deviation in the first cell, the LBM–LES simulation aligns well with both experimental and *direct numerical simulation* (DNS) references. For $y^+ > 30$, the LBM–LES results deviate slightly from the DNS data but remain in close agreement with the experimental results.

[5] `https://github.com/UCL-CCS/HemePure-GPU`

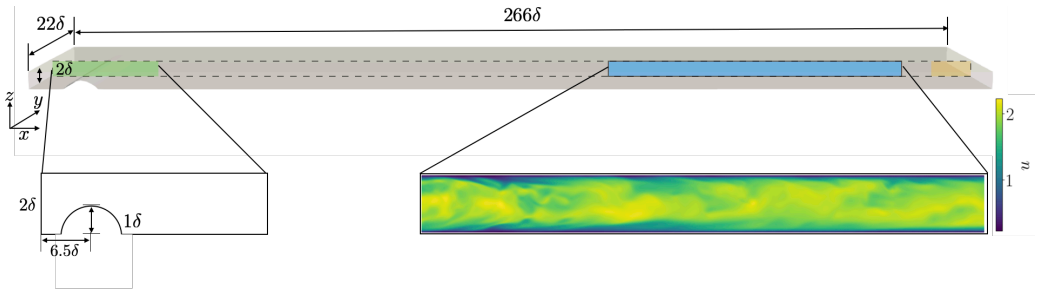


Figure 42: Illustration of a stenotic channel flow simulation setup. The green zone demonstrates the semi-cylindrical obstacle's configuration. The blue zone shows a snapshot where the channel flow has evolved into a fully developed turbulent velocity profile. The orange zone represents the sponge zone, designed to absorb reflective waves emanating from the outlet.

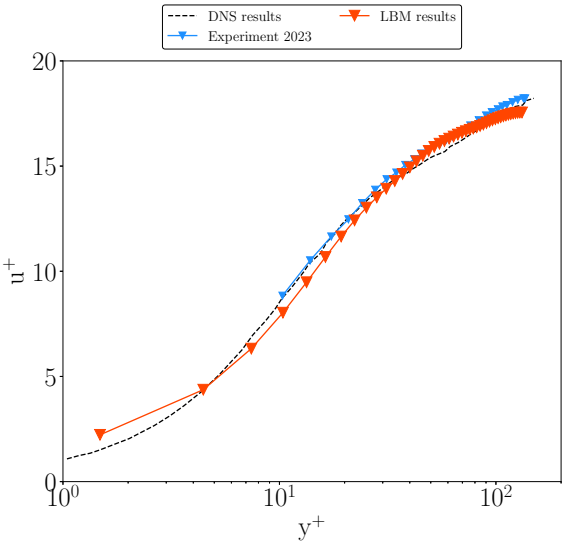


Figure 43: Representation of u^+ as a function of y^+, where the black dotted line illustrates the DNS reference data Kim, Moin, and Moser 1987. The blue triangle line depicts the PIV experiment data from Ding et al.Ding et al. 2021. The red triangle line represents the data obtained from the LBM–LES simulation.

Overall, the LBM–LES implementation demonstrates good concordance with both experimental and DNS simulations. Furthermore, we checked the dimensionless *root mean square* (RMS) for two velocity components on streamwise and vertical direction. We didn't include the spanwise direction due to the comparison with the *particle image velocimetry* (PIV) experiment. $u^{+\prime}_{rms}$, $v^{+\prime}_{rms}$ are the dimensionless RMS velocity components that are normalized with the shear velocity u_τ:

$$u^{+\prime}_{rms} = \frac{\sqrt{(u(\mathbf{x}) - \langle u \rangle)^2}}{u_\tau}. \tag{13}$$

Spatial averaging for RMS is performed only during post-processing due to the high resolution of the simulation. In Figure 43, the red dots represent the LBM–LES simulation results, while the blue dots correspond to the PIV experimental reference, and the black lines denote the DNS reference. Both experimental and LBM–LES results successfully capture the peak value of $u^{+\prime}_{rms}$ compared with the DNS data. Beyond the peak, both LBM–LES and experimental results gradually deviate from the DNS data, which may be attributed to statistical issues and the confinement of the channel flow. Regarding the vertical velocity component $u^{+\prime}_{rms}$, LBM–LES results align well with the experimental data but are slightly lower than the DNS results for $y^+ > 20$.

Considering both Figure 43 and Figure 44 it is evident that the LBM–LES implementation aligns closely with experimental and DNS results, demonstrating its capability to capture turbulent statistical quantities accurately. The comparison reveals that the LBM–LES method effectively replicates the

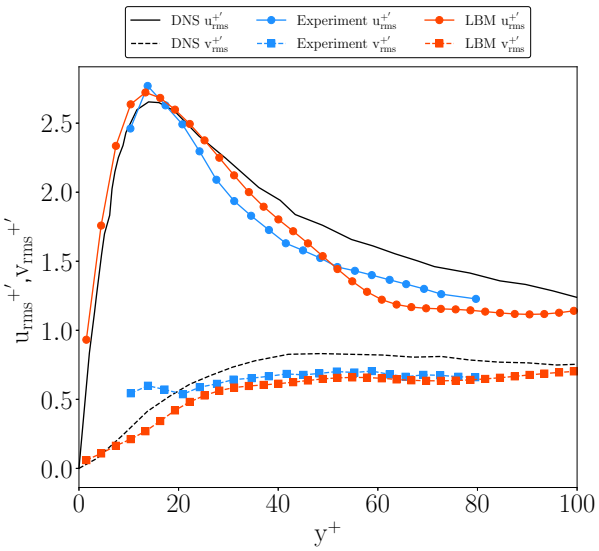


Figure 44: Presentation of $u_{rms}{}^{+\prime}$ and $v_{rms}{}^{+\prime}$ as functions of y^+, with the black and black dotted lines representing the DNS reference data Kim, Moin, and Moser 1987. The blue round and square lines correspond to $u_{rms}{}^{+\prime}$ and $v_{rms}{}^{+\prime}$ from the experiment Ding et al. 2021, respectively. Additionally, the red round and square lines depict the LBM simulation outcomes for $u_{rms}{}^{+\prime}$ and $v_{rms}{}^{+\prime}$, respectively.

key features of turbulent flows, confirming its reliability in modeling complex flow dynamics. This alignment shows the potential of LBM–LES in contributing valuable insights into the understanding of turbulence, particularly in the context of fluid dynamics simulations.

Acronyms

ANAET axissymmetric non-axissymmetric extended

AQUIFER advanced quantification of uncertainties in fusion modelling at the exascale with model order reduction

CDF cumulative distribution function

CL contour localization

CU computing unit

DA data assimilation

DAS deep active subspace

DGP deep Gaussian process

DNS direct numerical simulation

EPSRC Engineering and Physical Sciences Research Council

ESMACS enhanced sampling of molecular dynamics with approximation of continuum solvent

ESS effective sample size

GKDR gradient-based kernel dimension reduction

GP Gaussian process

HDF5 hierarchical data format, version 5

HGP heteroskedastic Gaussian process

HPC high performance computing

HTML hypertext markup language

KAS-GP kernelized active subspace Gaussian process

LBM lattice Boltzmann method

LES large-eddy simulation

LGP linked Gaussian process

LHD latin hypercube design

LSE level set estimation

MCMC Markov chain Monte Carlo

MD molecular dynamics

MLUPS million lattice cell updates per second

MOGP multi-output Gaussian process

MPI message passing interface

NEPTUNE neutrals and plasma turbulence numerics for exascale

NESO NEPTUNE exploratory software

NRMSE normalized root mean square error

ODE ordinary differential equation

OPE outer product emulator

PCA principal components analysis

PCE polynomial chaos expansion

PDE partial differential equation

PI principal investigator

PIV particle image velocimetry

POD proper orthogonal decomposition

QOI quantity of interest

RADDISH real-time advanced data assimilation for digital simulation of numerical twins on HPC

RBC Rayleigh–Bénard convection

RKHS reproducing kernel Hilbert space

RMS root mean square

RMSE root mean square error

SA sensitivity analysis

SC stochastic collocation

SDE stochastic differential equation

SEAVEA software environment for actionable and VVUQ-evaluated exascale applications

SHM simple harmonic motion

SPDE stochastic partial differential equation

SSC simplex stochastic collocation

TBR Tritium breeding ratio

TIES thermodynamic integration with enhanced sampling

UCL University College London

UKAEA United Kingdom Atomic Energy Association

UQ uncertainty quantification

VBMC variational Bayesian Monte Carlo

VVUQ verification, validation and uncertainty quantification

XML extensible markup language

References

Acerbi, Luigi (2018). "Variational Bayesian Monte Carlo". In: *Advances in Neural Information Processing Systems* 31.

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.

Beck, Joakim and Serge Guillas (2016). "Sequential design with mutual information for computer experiments (MICE): Emulation of a tsunami model". In: *SIAM/ASA Journal on Uncertainty Quantification* 4.1, pp. 739–766.

Beskos, Alexandros, Dan Crisan, and Ajay Jasra (2014). "On the stability of sequential Monte Carlo methods in high dimensions". In: *The Annals of Applied Probability* 24.4, pp. 1396–1445. DOI: 10.1214/13-AAP951. URL: https://doi.org/10.1214/13-AAP951.

Beskos, Alexandros, Dan Crisan, Ajay Jasra, et al. (2017). "A stable particle filter for a class of high-dimensional state-space models". In: *Advances in Applied Probability* 49.1, pp. 24–48.

Booth, Annie S, S Ashwin Renganathan, and Robert B Gramacy (2023). "Contour Location for Reliability in Airfoil Simulation Experiments using Deep Gaussian Processes". In: *arXiv preprint arXiv:2308.04420*.

Boyd, John P (1992). "Defeating the Runge phenomenon for equispaced polynomial interpolation via Tikhonov regularization". In: *Applied mathematics letters* 5.6, pp. 57–59.

Bunch, Pete and Simon Godsill (2016). "Approximations of the optimal importance density using Gaussian particle flow importance sampling". In: *Journal of the American Statistical Association* 111.514, pp. 748–762.

Cohn, David, Zoubin Ghahramani, and Michael Jordan (1994). "Active learning with statistical models". In: *Advances in neural information processing systems* 7.

Cole, D Austin et al. (2023). "Entropy-based adaptive design for contour finding and estimating reliability". In: *Journal of Quality Technology* 55.1, pp. 43–60.

Ding, Guanghui et al. (2021). "Transitional pulsatile flows with stenosis in a two-dimensional channel". In: *Physics of Fluids* 33.3.

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.

Dullin, Holger R et al. (2007). "Extended phase diagram of the Lorenz model". In: *International Journal of Bifurcation and Chaos* 17.09, pp. 3013–3033.

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.

Evensen, G. (2006). *Data Assimilation: The Ensemble Kalman Filter*. Springer Berlin Heidelberg. ISBN: 9783540383017. URL: https://books.google.co.uk/books?id=VJ2oOecHhOYC.

Evensen, Geir (1994). "Sequential data assimilation with a nonlinear quasi-geostrophic model using Monte Carlo methods to forecast error statistics". In: *Journal of Geophysical Research: Oceans* 99.C5, pp. 10143–10162.

Evensen, Geir, Femke C Vossepoel, and Peter Jan van Leeuwen (2022). "Weak Constraint 4DVar". In: *Data Assimilation Fundamentals: A Unified Formulation of the State and Parameter Estimation Problem*. Springer, pp. 49–61.

Farchi, Alban and Marc Bocquet (2018). "Comparison of local particle filters and new implementations". In: *Nonlinear Processes in Geophysics* 25.4, pp. 765–807.

Fearnhead, Paul and Hans R Künsch (2018). "Particle filters and data assimilation". In: *Annual Review of Statistics and Its Application* 5, pp. 421–449.

Feinberg, Jonathan and Hans Petter Langtangen (2015). "Chaospy: An open source tool for designing methods of uncertainty quantification". In: *Journal of Computational Science* 11, pp. 46–57. ISSN: 1877-7503. DOI: https://doi.org/10.1016/j.jocs.2015.08.008. URL: https://www.sciencedirect.com/science/article/pii/S1877750315300119.

Frei, Marco and Hans R Künsch (2013). "Bridging the ensemble Kalman and particle filters". In: *Biometrika* 100.4, pp. 781–800.

Gardner, Timothy S, Charles R Cantor, and James J Collins (2000). "Construction of a genetic toggle switch in Escherichia coli". In: *Nature* 403.6767, pp. 339–342.

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.

Gotovos, Alkis et al. (2013). "Active Learning for Level Set Estimation". In: *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*. AAAI Press.

Graepel, Thore (2003). "Solving noisy linear operator equations by Gaussian processes: Application to ordinary and partial differential equations". In: *ICML*. Vol. 3, pp. 234–241.

Graham, Matthew M and Alexandre H Thiery (2019). "A scalable optimal-transport based local particle filter". In: *arXiv preprint arXiv:1906.00507*.

Gramacy, Robert B (2020). *Surrogates: Gaussian process modeling, design, and optimization for the applied sciences*. CRC press.

Gramacy, Robert B and Herbert K H Lee (2008). "Bayesian treed Gaussian process models with an application to computer modeling". In: *Journal of the American Statistical Association* 103.483, pp. 1119–1130.

Groen, Derek et al. (2023). "FabSim3: An automation toolkit for verified simulations using high performance computing". In: *Computer Physics Communications* 283, p. 108596.

Huggins, Bobby et al. (2023). "PyVBMC: Efficient Bayesian inference in Python". In: *Journal of Open Source Software* 8.86, p. 5428. DOI: 10.21105/joss.05428. URL: https://doi.org/10.21105/joss.05428.

Järvenpää, Marko et al. (2021). "Parallel Gaussian Process Surrogate Bayesian Inference with Noisy Likelihood Evaluations". In: *Bayesian Analysis* 16.1, pp. 147–178. DOI: 10.1214/20-BA1200. URL: https://doi.org/10.1214/20-BA1200.

Kalman, R. E. (Mar. 1960). "A New Approach to Linear Filtering and Prediction Problems". In: *Journal of Basic Engineering* 82.1, pp. 35–45. ISSN: 0021-9223. DOI: 10.1115/1.3662552. eprint: https://asmedigitalcollection.asme.org/fluidsengineering/article-pdf/82/1/35/5518977/35_1.pdf. URL: https://doi.org/10.1115/1.3662552.

Kim, John, Parviz Moin, and Robert Moser (1987). "Turbulence statistics in fully developed channel flow at low Reynolds number". In: *Journal of fluid mechanics* 177, pp. 133–166.

Lee, John C and Norman J McCormick (2011). *Risk and safety analysis of nuclear systems*. John Wiley & Sons.

Lindgren, Finn, Håvard Rue, and Johan Lindström (2011). "An explicit link between Gaussian fields and Gaussian Markov random fields: the stochastic partial differential equation approach". In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 73.4, pp. 423–498.

Ma, Tian and Shouhong Wang (2005). *Bifurcation theory and applications*. Vol. 53. World Scientific.

Mandel, Jan (2006). *Efficient implementation of the ensemble Kalman filter*. Tech. rep. 231. Center for Computational Mathematics, University of Colorado at Denver and Health Sciences Center.

Mazzeo, Marco D and Peter V Coveney (2008). "HemeLB: A high performance parallel lattice-Boltzmann code for large scale fluid flow in complex geometries". In: *Computer Physics Communications* 178.12, pp. 894–914.

McHutchon, Andrew (2015). "Nonlinear modelling and control using Gaussian processes". PhD thesis. University of Cambridge.

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 (2023). "Deep gaussian process emulation using stochastic imputation". In: *Technometrics* 65.2, pp. 150–161.

Oakley, Jeremy (2004). "Estimating percentiles of uncertain computer code outputs". In: *Journal of the Royal Statistical Society Series C: Applied Statistics* 53.1, pp. 83–93.

Rabier, Florence and Zhiquan Liu (2003). "Variational data assimilation: theory and overview". In: *Proc. ECMWF Seminar on Recent Developments in Data Assimilation for Atmosphere and Ocean, Reading, UK, September 8–12*, pp. 29–43.

Rasmussen, Carl Edward, Christopher KI Williams, et al. (2006). *Gaussian processes for machine learning*. Vol. 1. Springer.

Rathgeber, Florian et al. (2016). "Firedrake: automating the finite element method by composing abstractions". In: *ACM Transactions on Mathematical Software (TOMS)* 43.3, pp. 1–27.

Rebeschini, Patrick and Ramon van Handel (2015). "Can local particle filters beat the curse of dimensionality?" In: *The Annals of Applied Probability* 25.5, pp. 2809–2866. DOI: 10.1214/14-AAP1061. URL: https://doi.org/10.1214/14-AAP1061.

Richardson, Robin A. et al. (Apr. 2020). "EasyVVUQ: A Library for Verification, Validation and Uncertainty Quantification in High Performance Computing". In: *Journal of Open Research Software*. DOI: 10.5334/jors.303.

Roth, Michael et al. (2017). "The Ensemble Kalman filter: a signal processing perspective". In: *EURASIP Journal on Advances in Signal Processing* 2017, pp. 1–16.

Rybin, Mikhail V et al. (2015). "Phase diagram for the transition from photonic crystals to dielectric metamaterials". In: *Nature communications* 6.1, p. 10102.

Santner, Thomas J et al. (2003). *The design and analysis of computer experiments*. Vol. 1. Springer.

Scheuerer, Michael (2010). "A comparison of models and methods for spatial interpolation in statistics and numerical analysis". In.

Slivinski, Laura and Chris Snyder (2016). "Exploring practical estimates of the ensemble size necessary for particle filters". In: *Monthly Weather Review* 144.3, pp. 861–875.

Snyder, Chris (2011). "Particle filters, the "optimal" proposal and high-dimensional systems". In: *Proceedings of the ECMWF Seminar on Data Assimilation for atmosphere and ocean*. Citeseer, pp. 1–10.

Solak, Ercan et al. (2002). "Derivative observations in Gaussian process models of dynamic systems". In: *Advances in neural information processing systems* 15.

Wendland, Holger (2004). *Scattered data approximation*. Vol. 17. Cambridge university press.

Whittle, Peter (1954). "On stationary processes in the plane". In: *Biometrika*, pp. 434–449.

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.