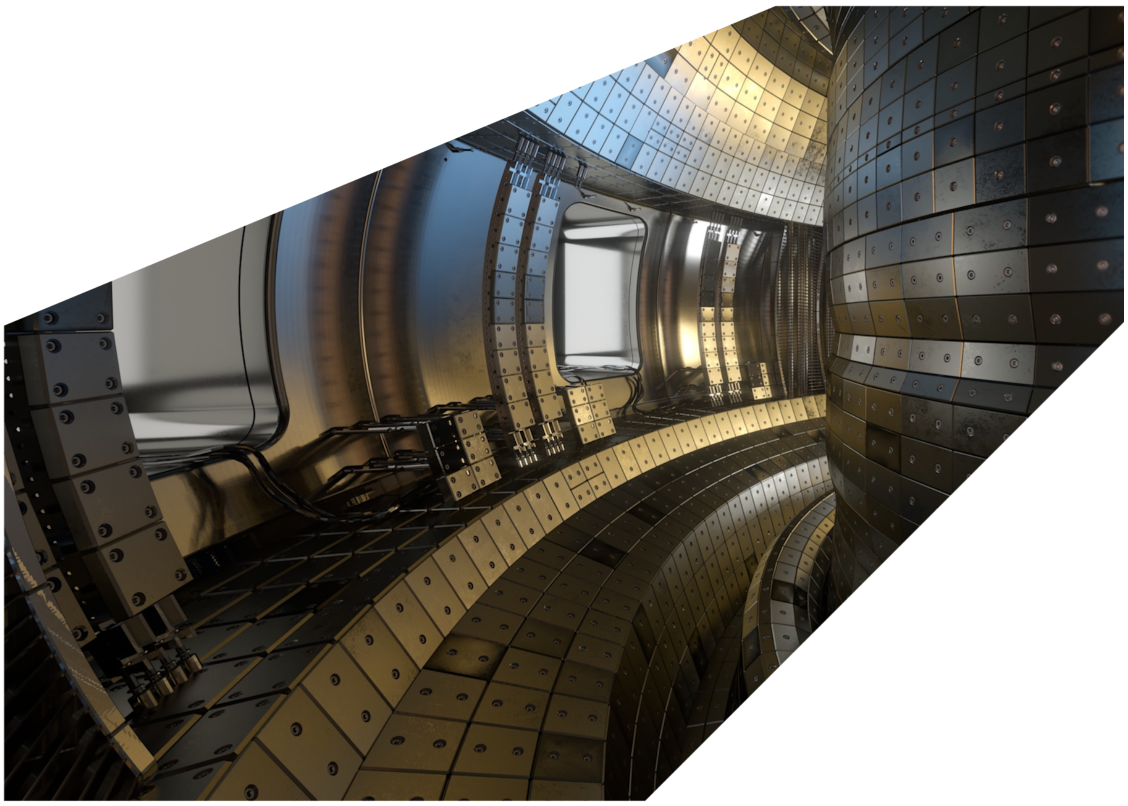
UK Atomic
Energy
Authority

# ExCALIBUR

Management of external research. Supports UQ Procurement

M5.1

**Abstract**
The report describes work for ExCALIBUR project NEPTUNE at Milestone 5.1. This report collates technical material used to inform the preparation of the call for procurement of support for Uncertainty Quantification (UQ). The work performed primarily consisted of collating material produced internally by UKAEA and by external grantees at UCL, plus hands-on exploration of sampling techniques, the construction of surrogates and the study of dimensionally reduced models. In addition, information from Exeter contractors gathered at NEPTUNE workshops is included. Besides paying for contractors, UKAEA is also sponsoring a PhD in UQ which is described in order to avoid duplication. The report has been slightly delayed so that it could be updated with material from a second NEPTUNE UQ workshop on 30 June, in order to facilitate assessment of bids for grants for the NEPTUNE 2021 call.



## UKAEA REFERENCE AND APPROVAL SHEET

| | | |
|---|---|---|
| | Client Reference: | |
| | UKAEA Reference: | CD/EXCALIBUR-FMS/0040 |
| | Issue: | 1.00 |
| | Date: | July 5, 2021 |
| Project Name: ExCALIBUR Fusion Modelling System | | |

| | Name and Department | Signature | Date |
|---|---|---|---|
| Prepared By: | Ed Threlfall<br>Joseph Parker<br>Will Saunders<br>Wayne Arter<br><br>BD | N/A<br>N/A<br>N/A<br>N/A | July 5, 2021<br>July 5, 2021<br>July 5, 2021<br>July 5, 2021 |
| Reviewed By: | Rob Akers<br><br>Advanced Computing Dept. Manager | | July 5, 2021 |
| Approved By: | Rob Akers<br><br>Advanced Computing Dept. Manager | | July 5, 2021 |

## 1 Introduction

This report collates technical material used to inform the preparation of the call for procurement of support for Uncertainty Quantification (UQ). The work performed primarily consisted of collating material produced internally [1, 2, 3, 4] and by external grantees at UCL [5, 6], plus hands-on exploration of sampling techniques, the construction of surrogates and the study of dimensionally reduced models. The production of this call reflects the importance attributed to UQ in project NEPTUNE, so that the software produced should be capable of producing 'actionable' results, that could form a significant input to multi-million pound procurements.

It seems worthwhile to start by recapitulating the project NEPTUNE requirements for UQ, insofar as they are currently understood. In particular, to note a clear distinction between the experiments that NEPTUNE code needs to model, and the outputs of the simulations themselves. Relevant experiments are almost entirely medium-sized and large tokamaks, where it seems that even the edge plasma has a temperature of around $10$ eV ($10^5$ K). The presence of magnetic fields of order $1-10$ T, and electric fields of around $10^4$ Vm$^{-1}$ implies a hostile environment, subject further to large electromagnetic transients not just at start-up and ramp-down of a discharge, but also caused by plasma instabilities such as ELMs and sawteeth. Most diagnostics struggle to achieve absolute accuracies of $10$ %, although they usually detect the direction of smaller changes reliably. Many are "one-of-a-kind" so require careful interpretation, and all signals may have been subject to filtering both at low and high frequencies, to remove 'spikes' and 'drift'. In comparison, signals from simulation are very 'clean', but the simulation may lack crucial realism, in that either important physical processes are not treated or if included, that the spatio-temporal discretisation may not be sufficiently fine to represent them accurately.

Turning to the expected usage of the NEPTUNE software, this covers both physics and engineering. Both physicist and engineers are potentially interested in everything computed by the software, which is of course likely to be an overwhelming amount of detail. A physicist is more likely to begin by formulating simpler models and use the software to determine their appropriateness and accuracy, an engineer probably more likely to use the output of simulation as a basis for producing simplified models. An experimental physicist may be satisfied with qualitative accuracy, provided it is obtained rapidly enough to help formulate the next experimental 'shot' (eg. by indicating that more or less gas input would be helpful), whereas a more theoretical counterpart may want to quantify the difference between model predictions and experiment sufficiently well to help produce a more refined model. An engineer seeking to design a reactor may be interested only in a relatively small number of quantities of interest (QoI) but which need to be determined more precisely, such as local maxima in time-averaged heat flux, whereas other engineers may see edge code as forming a part of a digital twin, to be used to help control operation of an actual tokamak device where accurate time-dependent modelling will also be critical. It is notable that most of the preceding requirements involve extrapolation, from model to new features of an existing experiment or, in the design case, to more extreme values of geometrical size, magnetic field etc. representing a fusion reactor relative to existing experiments such as JET.

For the above it is therefore as Smith [7] describes (see also the NEPTUNE report [4, § 1]), important that UQ involves more than just clever sampling to identify parameter sensitivities, and the standard definitions of ways of quantifying these such as Sobol indices [1]. In particular, the production of surrogate models is within scope of UQ. It is worth saying that in one sense even the most complex partial differential equation (PDE) models of plasma are simply surrogates for experimental ‘reality’. The most detailed and least controversial model of tokamak plasma consists in treating it as of order $10^{20}$ or more particles evolving under the influence of external and self-generated electromagnetic fields, but this is of course not tractable even at the Exascale [8], without very selective sampling of the particles. Currently, particle models are seen as at best struggling to capture edge plasma behaviour, with no certain likelihood of significant improvement, and in any event there is a separate calls that could cover research into particle sampling. The PDE models attempt to capture the key physics of the plasma particles, viz. their appearance, motion, interactions with other particles and eventual loss, as terms representing respectively source, advection, diffusion and sinks in continuum models. Unfortunately, long mean free paths (*mfp*s) mean that at least in some regions it is necessary explicitly to represent phase-velocity dependence of plasma density, and the resulting basic 6-D dependence of fields (5-D if gyro-averaged models are used) stretches current and foreseen computing facilities. In any event, above applications eg. preliminary design studies and operational control, require models capable of evaluation within seconds or less, ie. surrogates for the PDEs, expected to have lower spatial dimensionality (2-D or 1-D) or even as systems of ordinary differential equations (ODEs).

Unfortunately for application to NEPTUNE, most surrogates have been designed for use as interpolants. Nonetheless, interpolation is likely to be important at least within the software, eg. to connect different models, and the book by de Boor [9] has interesting material relating to the use of splines in this context. In particular, de Boor lists theorems relating to the optimality of spline polynomial interpolation, so that there might be wonder that any other form of interpolation is required. In fact, spline optimality is found only in the energy or least-squares norm, so there is anyway the possibility of fitting the data with polynomial functions that have under- and over-shooting oscillations between sample points (called nodes in spline jargon). Moreover, de Boor illustrates the potential for dramatic failings using what he refers to as the Titanium heat data set [9, Fig. XIII.1]. The data is a set of some $50$ apparently regularly spaced samples of a function which is practically flat except for one distinct ‘hump’, rising to four times the value of the ‘plateau’ over c. $15\,\%$ of the range. Since the interpolation is by a piecewise ($5^{th}$ order polynomial) the undershoot by greater than the maximum ‘hump’ value is in many respects more impressive than that exhibited in Boyd’s book [10, Fig. 4.3] as part of his demonstration of the Runge phenomenon, ie. that polynomial interpolation at evenly spaced points may diverge. In both books, the problem is identified as due to poor choice of sample points. de Boor [9, Fig. XIV.5] shows improved fitting if the number of samples in the plateau region of the *Ti* heat data is reduced, whereas Boyd points out that optimal $N^{th}$ order polynomial interpolation is achieved using sample points that are the roots of the Chebychev polynomial of degree $(N+1)$, that cluster as close together as $1/N^2$ near the end-points of the sample region. In any event, for many interpolation purposes, it seems that polynomial spline fitting with an order of three or so, is adequate *provided* the nodes are chosen suitably. The cost scales as the operation count of a banded-diagonal matrix solver, so $\mathcal{O}(N)$ where $N$ is the number of points to be interpolated per dimension. There are however strong proponents for the use of rational polynomial interpolation [11] and in particular radial basis function [12]. Wavelet bases [13] also have their adherents, and special conditions such as periodicity or infinite domain mean that respectively Fourier series and the Hermite basis are to be preferred. The Hermite basis is employed in the Polynomial Chaos Expansion (PCE), which is advocated by P. Coveney et al, see the NEPTUNE report [5].

Accurate spline fitting requires a sufficient amount of data to enable best location of the nodes, which, since they generally aim to use as few nodes as possible, is commonly unavailable in the output of highly expensive numerical calculations. This lack of data means also that interpolation using Neural Networks (NNs) is mostly unsuitable, and motivates the use of Gaussian Processes (GPs) for interpolation despite their expense, that $\mathcal{O}(N^3)$ of inverting a full matrix [2]. They come with the benefit of accompanying error bounds, and seem to be relatively stable when used for extrapolation [14]. They may be very effective in combination with a special basis produced by Principal Component Analysis (PCA), *aka* Proper Orthogonal Decomposition or POD, see the NEPTUNE report [6]. Moreover, S. Guillas [15] describes a number of techniques which outperform PCA, in particular favouring a reproducing kernel approach to estimating derivatives needed to optimise the basis. This is coupled with a sequential approach to selecting sample points. Indeed, choosing sample points appropriately is still a major issue for GPs, which P. Challenor (private communication) has explored extensively and favours 'Leave One Out' (LOO).

Nonetheless, given the general need for extrapolation, it is desirable to incorporate as much physical knowledge as possible into the surrogate model. The hopes are that the parameters of the surrogate are either invariant under the extrapolation, or that they have well-determined scaling properties, and that all important physical processes have been identified. The parameters of the surrogate may then be fitted using existing data from experiment and/or simulation as described above. The surrogate may need to be space and/or time dependent as required by the user and application. Of course, such surrogates can be very useful for interpolation also. Worthy of particular mention is Model Order Reduction [3], which for linear and some nonlinear problems, offers reliable estimates of error in interpolating Quantities of Interest (QoI), hence likely to be particularly useful for the design engineer.

The body of the current document consists of hands-on exploration of sampling techniques in Section 2, the construction of surrogates by Gaussian Processes in Section 3 and of surrogates consisting of dimensionally reduced models in Section 4. The implications for the call are summarised in Section 5.

## 2 BOUT++

BOUT++ [16, 17] is a framework for writing fluid and plasma simulations in curvilinear geometry. It is designed primarily to target reduced plasma fluid models, but it can evolve any number of partial differential equations, with equations appearing in a readable form via its domain specific language. BOUT++ is being used in Project NEPTUNE as the framework in which to build the fluid referent model proxyapp.

A team of BOUT++ developers attended the three VECMAtk hackathons in Spring 2021. The goal of the hackathon work was to create the software framework to enable BOUT++ to use VECMAtk, and to develop workflows for uncertainty quantification and the creation of surrogate models. In this report we focus on the development of the software infrastructure and the workflows for uncertainty quantification. The code produced at the hackathons is available in the repository [18].

### 2.1 Software Infrastructure

The VECMA toolkit wraps around user's software, providing a library of python functions for non-intrusive UQ. From VECMAtk's perspective, BOUT++ is a black box code to which it must provide input parameters and from which it must read output quantities of interest (QoIs). Therefore the workflow is written in Python, and is driven by calls to VECMAtk functions. VECMAtk selects the parameter values to be sampled, collects the QoI values, and calculates and plots statistical quantities such as Sobol indices and confidence intervals. The only necessary interaction between VECMA and BOUT++ is for VECMA to communicate input parameters to BOUT++ and to read back output values. This is achieved by providing an *encoder*, a routine that is capable of writing BOUT++ input files, and a *decoder*, a routine to parse BOUT++ output files. Both are actually classes providing additional functions, such as routines to check that BOUT++ simulations have finished. Examples of decoders and encoders used for BOUT++ are given in Appendices A and B respectively.

This framework structure makes it very easy to reuse the same workflow with different codes. BOUT++ could be replaced by any other code that produces the same QoIs from the same parameters simply by replacing the BOUT++ encoder/decoder with corresponding functions for that other code.

This pattern also makes it easy to use synthetic data instead of an actual code execution. Instead of calling the encoder/decoder, one simply calls a function to return the QoIs as a function of parameters. This is particularly useful in developing the workflow, such as testing the sampling routines or plotting functions, as executing BOUT++ is by far the most significant runtime cost in the workflow.

The encoders/decoders also provide the method for changing the QoIs for different physical cases. For example in our work with temperature $T(x, y, z, t)$ as the QoI, each of the different cases – fixed point over time $T(0, 0, 0, t)$, spatial range at the final time $T(x, 0, 0, t_{\mathrm{end}})$, or logarithm of temperature $\log[T(x, 0, 0, t_{\mathrm{end}})]$ – were each achieved by implementing a separate decoder.[1]

[1]A similar approach allowed us to work around a bug in the `chaospy` sampling routine (a library dependence of VECMAtk). The chaospy log-uniform probability distribution was broken, but the uniform distribution was functional.

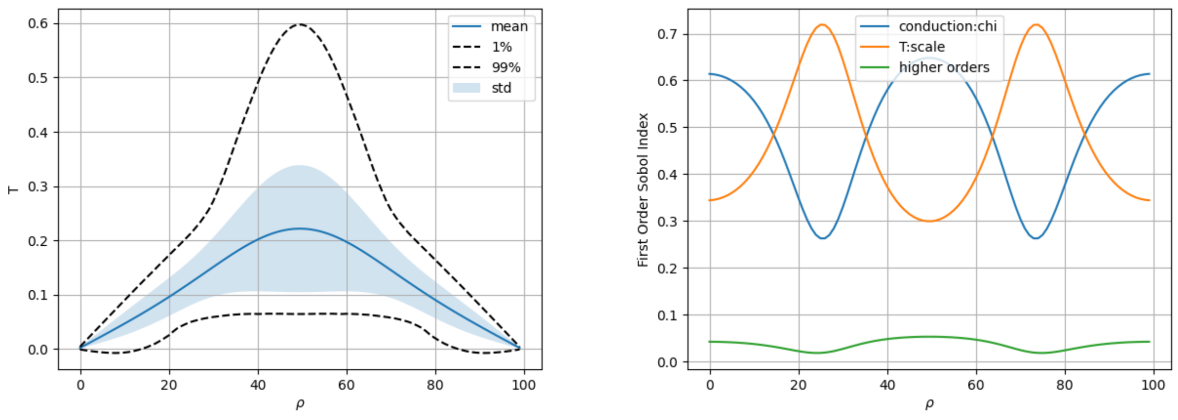


Figure 1: Mean and confidence intervals (left) and Sobol indices (right) plotted against $x$ array index for quantity of interest $T$.

## 2.2 Uncertainty Quantification

The first workflow considered introduced UQ into BOUT++ simulations. Instead of performing a single simulation to obtain a single result, the VECMAtk UQ framework performs an ensemble of simulations, varying chosen input parameters in a specified range. From this, VECMAtk fits a surrogate model: it finds the QoIs as a function of the input parameters. With this model it calculates statistics, such as the mean value of a QoI and corresponding confidence intervals, and the Sobol indices, which give a way of quantifying how much uncertainty in the QoIs is due the variation of each parameter.

One-dimensional heat conduction was used as an initial test problem,

$$\frac{\partial T}{\partial t} = \chi \frac{\partial^2 T}{\partial x^2}, \tag{1}$$

where $T$ is the temperature on the spatial domain $x \in [0, 1]$ with boundary conditions $T(0, t) = T(1, t) = 0$ and initial condition $T(x, 0) = T_0 \exp[-(x - 0.5)^2/0.04]$. Here, the thermal conductivity $\chi$ and the initial amplitude $T_0$ are uncertain parameters.

### 2.2.1 Polynomial chaos expansion

As a first example, we implemented a workflow for 1D heat conduction (1) varying $\chi \in (0.2, 4)$ and $T_0 \in (0.5, 1.5)$ using polynomial chaos sampling. The resulting statistics and Sobol indices for $T$ at the final time $T = 10$ is shown in Figure 1. The left figure shows the mean final temperature and its confidence intervals . The right figure shows the first Sobol indices, which describe how much of the uncertainty is due to each parameter. For example, at the centre of the domain, about

Therefore, in order to obtain a log-uniform distribution, we provided an encoder that wrote input parameter `p` to the BOUT++ input file as `variable=10**p` instead of `variable=p`.

$65\%$ of the uncertainty is due to varying $\chi$ (independently of $T_0$), about $30\%$ is due to varying $T_0$ (independently of $\chi$), and the remaining $5\%$ is due to varying both in a coupled fashion.

The statistics are qualitatively converged using only a 3rd-order polynomial chaos expansion which requires $(3+1)^2 = 16$ code evaluations in two dimensions. However the confidence intervals encompass negative values of temperature near the boundary, even though the temperature is non-negative and all simulation data provided is also non-negative.

One approach fixing this is to use $\log(T)$ as the QoI. The logarithm is a bijective map between the positive-valued temperature $T \in (0, \infty)$ and the real value $\log(T) \in (-\infty, \infty)$. Therefore this allows the same analysis routines to be performed on $\log(T)$ without concern about sign. The resulting confidence intervals now only allow positive temperatures. Moreover the Sobol indices are qualitatively the same, indicating that the transformed model ascribes uncertainty to the various parameters in a similar fashion.

However, considering higher-order polynomial chaos it becomes apparent that the negative values are an artefact of the data analysis routine. Near $T = 0$, the temperature is a steeply-varying function of the input parameters, and low-order polynomial fitting is simply inaccurate, predicting negative values. Increasing the order of the polynomial fit (by increasing the order of PCE and providing more simulation data) leads to this problem's no longer being observed.

### 2.2.2 Adaptive stochastic collocation

Using higher-order polynomial chaos is reasonable when varying a small number of parameters, but varying more parameters is beset by the "curse of dimensionality": the number of samples needed for $n$th-order PC with $d$ parameters is $(n+1)^d$. Therefore, we also implemented EasyVVUQ's sampling with adaptive stochastic collocation (ASC), which is based on the algorithm by Gerstner and Griebel [19]

In ASC, the grid of sampling points for each parameter is adaptively refined, beginning from a grid with only one sampling point for each parameter. Estimates are made for the reduction in the error that would result from adding one more sampling point in each parameter; the refinement is made to the parameter which would reduce the error the most. As a result, computing resources are directed where they are most effective, yielding results that are comparable with high-order PCE but which require many fewer code evaluations.

Implementing a new sampling algorithm (from a VECMA user's perspective) is a simple matter of changing the class on which sampling routines are called. However, adaptive stochastic collocation allows additional functionality which we may incorporate in our workflow. In particular, ASC uses a Clenshaw–Curtis collocation grid which has a nested quadrature grid (unlike in Gaussian quadrature, the grid for $(n+1)$th order accuracy contains all the points for $n$th order accuracy). This means we may increase the order of accuracy by adding one new code evaluation, rather than $(n+1)$ as would be required in PC. This allows us to use an iterative approach, continuing to refine the order in each parameters dimension until some convergence criteria is met. In our workflow, we used relative changes in the Sobol indices from one iteration to the next, but this criterion is not ideal as it is noisy and non-monotonic.

Finally, as a proof of concept we implemented uncertainty quantification for a two-dimensional simulation of plasma filament propagation,

$$\frac{\partial n}{\partial t} = -\{\phi, n\} + 2\frac{\rho_s}{R_c}\frac{\partial n}{\partial z} + D_n \nabla^2 n, \tag{2a}$$

$$\frac{\partial \Omega}{\partial t} = -\{\phi, \Omega\} + 2\frac{\rho_s}{R_c}\frac{\partial n}{\partial z} + \frac{D_\Omega}{n}\nabla^2 \Omega, \tag{2b}$$

$$\nabla^2 \phi = \Omega, \tag{2c}$$

in a two-dimensional box $(x, z)$ with Dirichlet boundary conditions in $x$ and periodic boundary conditions in $z$. In (2), $n$ is plasma density, $\Omega$ is vorticity, $\phi$ is electrostatic potential, $t$ is time, $\rho_s$ is the Bohm gyroradius, $R_c$ is the radius of curvatuve, $D_n$ and $D_\Omega$ are dissipation parameters, and $\{A, B\} = (\partial A/\partial x)(\partial B/\partial z) - (\partial A/\partial z)(\partial B/\partial x)$ is a Poisson bracket. We use the two-dimensional Laplacian $\nabla^2 \equiv \partial^2/\partial x^2 + \partial^2/\partial z^2$. Between each timestep, the vorticity equation (2c) must be solved for $\phi$ so its value can be used in (2a) and (2b) to advance $n$ and $\Omega$. We use $\phi$ at the current timestep as an initial guess for the iterative method (and $\phi = 0$ as the guess for the first time step).

Four input parameters were varied, the constant coefficients $\rho_s$, $R_c$, $D_n$ and $D_\Omega$. While still reasonably small, individual simulations require around 10 minutes on 16 cores on the University of York's cluster. Third order PCE required 1296 simulations, while comparable results using ASC only required 256 simulations.

# 3 Nektar++

Nektar++ [20, 21] is a framework for solving partial differential equations (PDEs) using the spectral / hp-element method. The software contains several different solvers targeting canonical problems in fluid mechanics. It is possible to extend the framework by adding new domain-specific solvers; this currently involves low-level coding as there is no DSL. Nektar++ is being used in Project NEPTUNE as the framework for the higher-order solution of fluid equations and currently is being extended to handle plasma fluids. Data is extracted from Nektar++ simulations by means of *filters* and written to text-based output files.

A small team from UKAEA attended the three VECMAtk hackathons in Spring 2021. The goal of the hackathon work was to create the software framework to enable Nektar++ to use VECMAtk, and to develop workflows for uncertainty quantification and the creation of surrogate models.

## 3.1 Software Infrastructure

The VECMA toolkit wraps around an individual Nektar++ solver to provide non-intrusive UQ (only non-coupled workflows are considered here). The toolkit interacts with Nektar++ via the inputs and outputs of the latter, through respectively an *encoder* and a *decoder*; these objects can be interchanged to enable different workflows (for example, the relevant quantities of interest (QoIs) are parsed in the decoder).

## 3.2 Uncertainty Quantification

The initial test problem used the incompressible Navier-Stokes solver of Nektar++ to simulate classical convection: a two-dimensional fluid-filled slot with vertical walls maintained at different temperatures; starting with an initial uniform temperature gradient from the hot side to the cold side, the system is time-evolved until a steady state is reached. Warmer fluid is subject to an upward buoyancy force, leading to a steady convective cell solution with fluid travelling up the hot side and down the cold one.

In dimensionless form, the equations for fluid velocity $\mathbf{u}$, temperature $T$ and pressure $p$ are

$$\frac{1}{Pr}\left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u}\right) = -\nabla p + Ra\,T\,\hat{\mathbf{y}} + \nabla^2\mathbf{u} \tag{3}$$

$$\left(\frac{\partial T}{\partial t} + \mathbf{u}\cdot\nabla T\right) = \nabla^2 T \tag{4}$$

$$\nabla\cdot\mathbf{u} = 0. \tag{5}$$

Here, the Rayleigh number $Ra$ and the Prandtl number $Pr$ are the governing parameters and are taken to be subject to uncertainty.

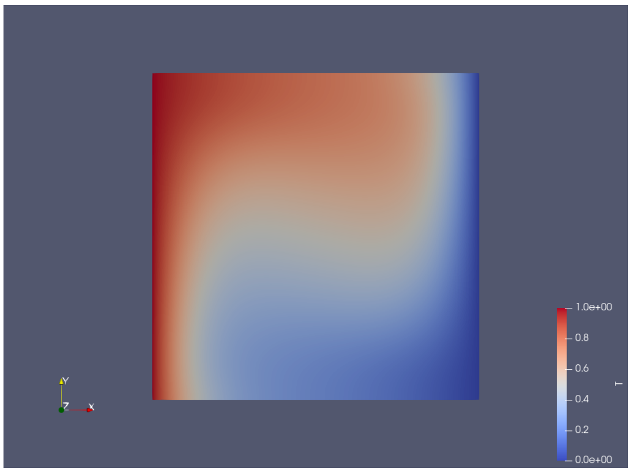


Figure 2: Steady-state colour map of dimensionless temperature for $Ra = 10^4$, $Pr = 1.0$. The left side is maintained at a temperature one unit higher than the right and a convective cell carries fluid in a clockwise circulation.

The Nektar++ simulations were designed so as to run quickly on a single desktop computer. Computational parameters governing accuracy were chosen to give plausible results without detailed analysis of convergence.

### 3.2.1 Polynomial chaos expansion

A square slot was used as a first example; the results follow closely the original investigation in [22]. An example temperature profile over the full domain can be see in Figure 2.

The QoI was the temperature profile $T(x)$ evaluated half-way up the slot, parsed via the decoder `SlicedHisDecoder` from data extracted using the Nektar++ `HistoryPoints` filter to record field values at chosen sample points. Note that the sample points cannot lie on the vertical boundaries, as the Dirichlet boundary condition enforces constant temperature values there, which causes the analysis to fail (a singular matrix error is reported by VECMA). $Pr$ was taken to be in the range $1 - 10$ (values typical for the gases and liquids commonly investigated in the literature), with a uniform distribution. $Ra$ was taken to be in the range $1.0 \times 10^4 - 3.2 \times 10^4$ (values consistent with a laminar convective steady state), with a log-uniform distribution.

The polynomial chaos expansion (PCE) produces a fit to the sampled QoI data using a sum over orthogonal polynomials in the input variables, where the orthogonality is defined using the input probability distributions as the measure and the fit is performed by either spectral projection or linear regression. The polynomial chaos analysis of EasyVVUQ provides the mean, variance, and standard deviation of the outputs and also the minimum, maximum, median and the $1^{st}$, $10^{th}$, $90^{th}$, and $99^{th}$ percentiles (a useful `supported_stats()` routine lists the available statistics). Some of these quantities are shown in Figure 3, which was generated using a fifth-order PCE. Also shown are the first Sobol indices, which show correctly that the uncertainty in the output is dominated by the sensitivity to $Ra$ in this case; in the regime in question, the effect of varying $Pr$ is small.

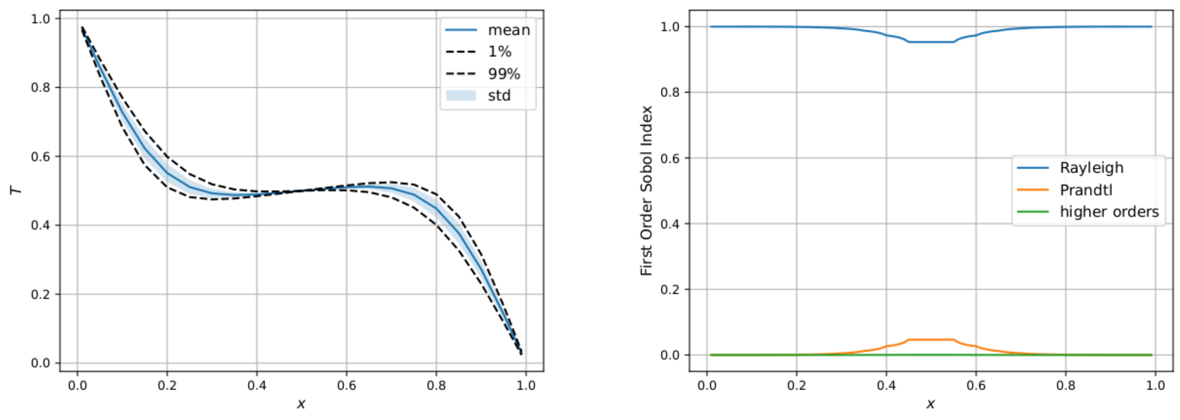


Figure 3: Mean and confidence intervals (left) and first Sobol indices (right) as a function of position for the central temperature profile $T(x)$ fitted by a polynomial chaos expansion. Note that the uncertainty tends to zero in regions where the temperature is either fixed (end points) or is constrained by symmetry (centre).

### 3.2.2 Stochastic collocation

Stochastic collocation (SC) creates a representation of the QoIs that is exact at the sample points; the solution is a linear combination of sample-point values multiplying Lagrange polynomials at those points. The stochastic collocation analysis of EasyVVUQ provides the mean, variance and standard deviation of the outputs. The set of available statistics is smaller than for the PCE, but the method is more efficient (as explained earlier in this report). The decoder used in the preceding section is switchable between PCE and SC.

## 3.3 Surrogate models

Note that, although it concerns surrogate models, the work in this subsection used EasyVVUQ and not EasySurrogate.

### 3.3.1 Polynomial chaos expansion and stochastic collocation

Either of the orthogonal polynomial fit constructed in the polynomial chaos expansion or the Lagrange polynomial fit constructed in the stochastic collocation method can serve as a computationally-inexpensive surrogate for the full model. As an example, PCE and SC surrogates for the Nusselt number $Nu$ as a function of $Ra$ and $Pr$ were constructed. $Nu$ is the dimensionless heat flux across the cavity, normalized to the $Ra = 0$ no-convection solution, calculated as

$$Nu = \frac{\int_0^H \partial_x T(x_0, y) + \partial_x T(x_1, y) dy}{\int_0^H \partial_x T(x_0, y) + \partial_x T(x_1, y) dy|_{Ra=0}} . \tag{6}$$

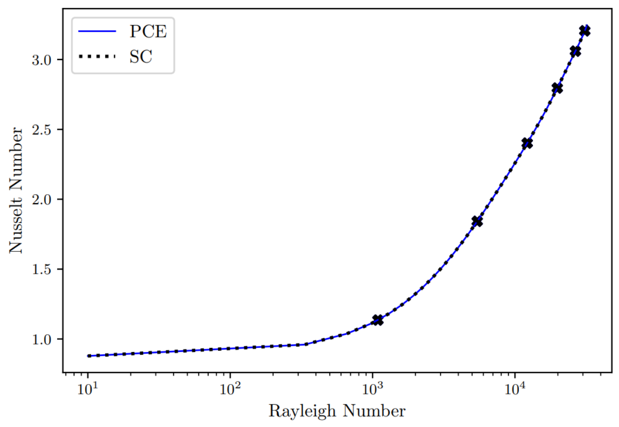


Figure 4: Polynomial chaos and stochastic collocation surrogates for Nusselt number as a function of Rayleigh number, for Prandtl number $1.0$.

where $\partial_x$ denotes partial derivative with respect to coordinate $x$.

The Nusselt number was extracted using a customized filter in Nektar++ and the output file was parsed by a decoder (`HeatFluxDecoder`) using a simple text processing routine. An illustration of the SC and PCE fits (both using fifth-order polynomial expansions) for the simple case of Nusselt number as a function of the Rayleigh number ($Pr = 1$) is shown in Figure 4. The Nusselt number is seen to follow a typical power-law curve.

### 3.3.2 Gaussian process surrogate

A different set-up, though again using the Nektar++ incompressible Navier-Stokes solver, was used as an initial exploration of the Gaussian process surrogate capability in EasyVVUQ; this was a slot $10$ units high and $1$ wide, operating at larger Rayleigh numbers where the flow exhibits a boundary-layer instability with waves moving up the hot wall and down the cold one (illustrated in Figure 5). This gave interesting time series for the maximum temperature near the affected parts of the walls (the lower part of the cold wall was used) and the position of the hottest point. These QoIs are of interest in fusion applications as they characterize the hot spots on the wall of a reactor containing hot plasma. The time series were extracted over a chosen time window using a customized filter in Nektar++; the simulation runs were performed once and the decoder (`HotSpotDecoder`) was given the ability to read stored simulation data in order to avoid repeating calculations. In view of the weak dependence on the parameter $Pr$, only the parameter $Ra$ was varied in this activity (values in the range $10^5 - 10^7$ were used). A kernel with zero mean and a Matérn covariance with parameters $\nu = 3/2$ and scale $10^7$ were used.

This investigation was an attempt at finding a surrogate for computations that are more demanding than the laminar convection ones in the preceding section and also concerned a system that exhibits deterministic chaos. The time series data used to train the Gaussian process is shown in Figure 5. Note that for $Ra$ sufficiently large for the instability to form, temperature maxima enter from the top of the monitored section and proceed downwards until usurped by the next wave. For smaller values of $Ra$, the system is quiescent and the time series eventually become constant.

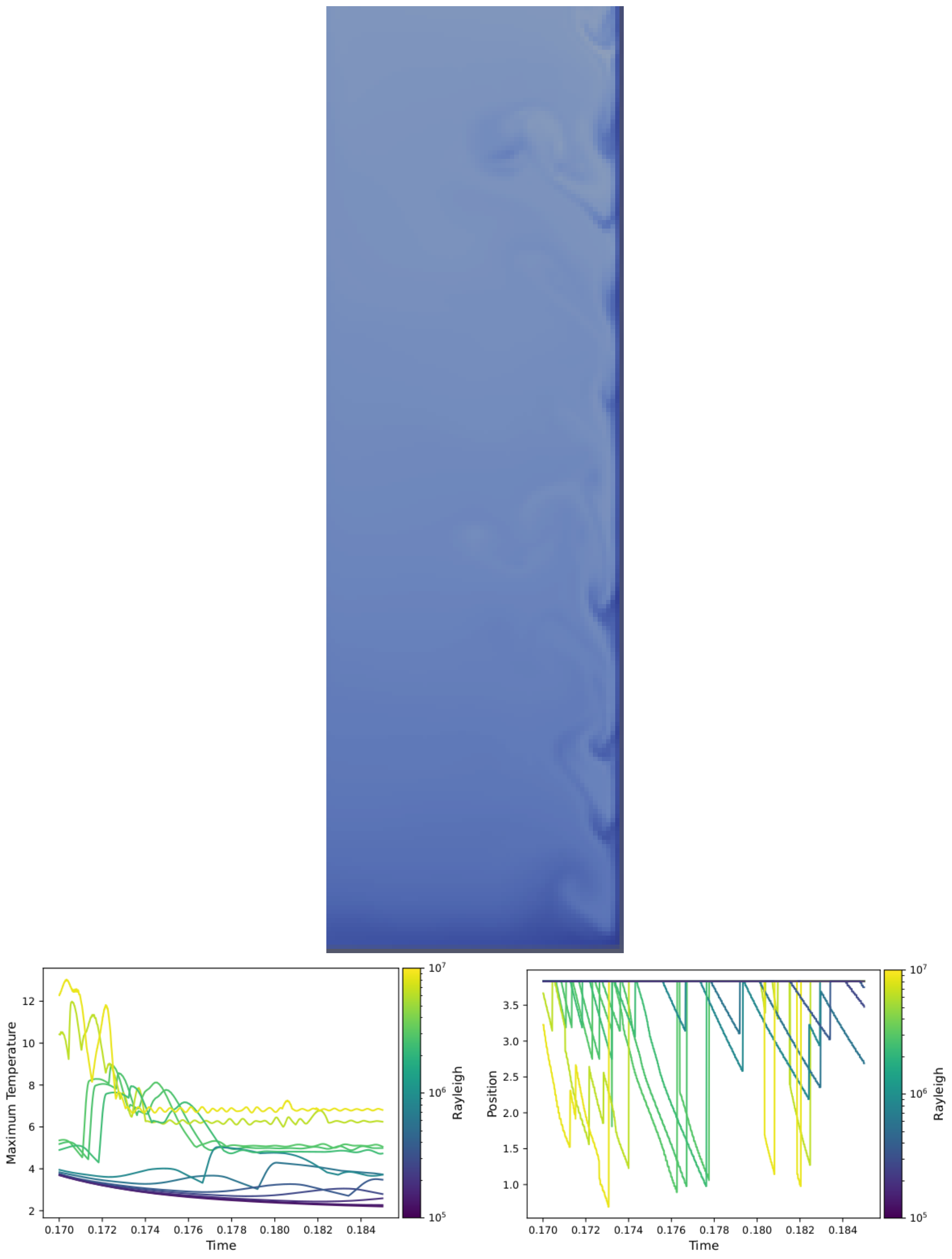


Figure 5: Boundary-layer instability at the cold wall for $Ra = 2.3 \times 10^8$ (top). The maximum temperature on this part of the wall and also the position of the hottest point are shown as time series (respectively left and right below).

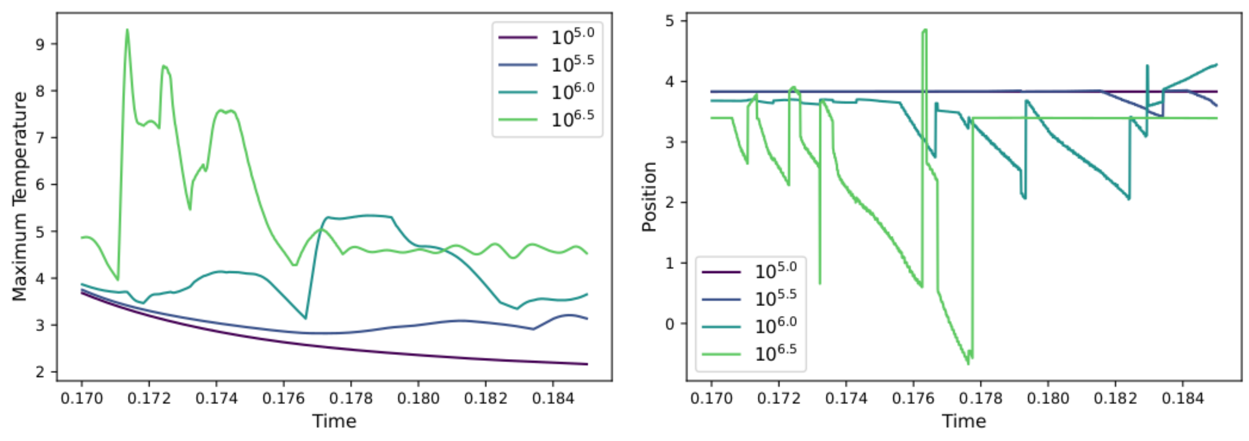


Figure 6: Output of the Gaussian process surrogates for the maximum temperature and the position of the hottest point evaluated at positions away from the sample parameter values.

Some sample evaluations of the Gaussian process surrogate away from the sample points can be seen in Figure 6. It can be seen that the surrogate correctly reproduces the more active nature of the time series for larger values of $Ra$.

# 4 Dimensionally reduced models

## 4.1 Boltzmann Equation

One suggestive example of the production of a surrogate is the reduction of the Boltzmann equation to the equations of fluid dynamics. The model is a combination of Lie derivative and solenoidal constraints

$$\begin{aligned} \frac{\partial f}{\partial t} + u_x \frac{\partial f}{\partial x} + u_y \frac{\partial f}{\partial y} &= S && (7) \\ \frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} &= 0 && (8) \end{aligned}$$

where $f(x, y, t)$ is the phase space density as a function of time $t$, $u_x$ is the flow in position space $x$, and $u_y$ is the flow in velocity space $y$. Collision and other source terms are represented as $S(x, y, t)$. For simplicity, it is assumed that position and velocity space are each 1-D. In more conventional notation where $v$ is the velocity space coordinate and $a$ the acceleration experienced by a particle, then $y \leftrightarrow v$, $u_x = v$ and $u_y = a$.

Integrating Equation (7) over velocity space, ie. forming $\int \cdot dy$ leads to

$$\frac{\partial n}{\partial t} + \frac{\partial}{\partial x}\int u_x f dy - \int \frac{\partial u_x}{\partial x} f dy + \frac{\partial}{\partial y}\int u_y f dy - \int \frac{\partial u_y}{\partial y} f dy = \int S dy \quad (9)$$

upon using

$$u_x \frac{\partial f}{\partial x} = \frac{\partial (u_x f)}{\partial x} - f \frac{\partial u_x}{\partial x} \quad (10)$$

and introducing

$$n = \int f dy \quad (11)$$

Using Equation (8) so that two terms cancel, Equation (9) becomes

$$\frac{\partial n}{\partial t} + \frac{\partial}{\partial x}\int u_x f dy + [u_y f]_{\pm y_b} = \int S dy \quad (12)$$

If $f$ and/or $u_y$ vanish(es) at the boundaries of the velocity domain $y = \pm y_b$, the third term is zero. Suppose there is a mean flow, ie. that $u_x$ may be written

$$u_x = U(x) + \tilde{u}(x, y) \quad (13)$$

such that

$$\int \tilde{u} f dy = 0 \quad (14)$$

and thus

$$\int u_x f dy = nU \quad (15)$$

then substituting Equation (13) in the second term of Equation (12) results in

$$\frac{\partial n}{\partial t} + \frac{\partial nU}{\partial x} = S_0 \quad (16)$$

which is recognised as the fluid dynamical equation of mass conservation with source $S_0(x) = \int S dy - [u_y f]_{\pm y_b}$.

To obtain the momentum conservation equation from Equation (7), form $\int \cdot u_x dy$ and rearrange $f$-derivative terms, leading to

$$\frac{\partial nU}{\partial t} + \frac{\partial}{\partial x}\int u_x^2 f dy - \int \frac{\partial u_x^2}{\partial x} f dy + \frac{\partial}{\partial y}\int u_x u_y f dy - \int \frac{\partial (u_x u_y)}{\partial y} f dy = \int S u_x dy \tag{17}$$

Using

$$\frac{\partial (u_x u_y)}{\partial y} = u_x \frac{\partial u_y}{\partial y} + \frac{\partial u_x}{\partial y} u_y = -u_x \frac{\partial u_x}{\partial x} + \frac{\partial u_x}{\partial y} u_y \tag{18}$$

(the second manipulation using Equation (8)), Equation (17) gives

$$\frac{\partial nU}{\partial t} + \frac{\partial}{\partial x}\int u_x^2 f dy - \int u_x \frac{\partial u_x}{\partial x} f dy + [u_x u_y f]_{\pm y_b} - \int u_y \frac{\partial u_x}{\partial y} f dy = \int S u_x dy \tag{19}$$

Writing $u_x = U(x,t) + \tilde{u}$, the second term expands to

$$\frac{\partial}{\partial x}(U^2 \int f dy) + 2\frac{\partial}{\partial x}(U \int \tilde{u} f dy) + \frac{\partial}{\partial x}\int \tilde{u}^2 f dy \tag{20}$$

which since $\int \tilde{u} f dy = 0$, reduces to

$$\frac{\partial (nU^2)}{\partial x} + \frac{\partial}{\partial x}\int \tilde{u}^2 f dy \tag{21}$$

The third term in Equation (19) is zero provided

$$\frac{\partial u_x}{\partial x} = 0 \tag{22}$$

Indeed if $u_x = y$, then upon introducing

$$p = \int \tilde{u}^2 f dy \tag{23}$$

and

$$G = \int u_y f dy = \int a f dy \tag{24}$$

there results the fluid dynamical equation of momentum conservation with momentum source $S_1(x) = \int S u_x dy - [u_x u_y f]_{\pm y_b}$, namely:

$$\frac{\partial nU}{\partial t} + \frac{\partial nU^2}{\partial x} = -\frac{\partial p}{\partial x} + G + S_1 \tag{25}$$

It may be helpful to note that the third term in Equation (19) expands to

$$-(nU\frac{\partial U}{\partial x} + U\int \frac{\partial \tilde{u}}{\partial x} f dy + \int \tilde{u}\frac{\partial \tilde{u}}{\partial x} f dy) \tag{26}$$

where Equation (14) has been used to zero one term.

## 4.2 Lie Derivative Surrogate

In the absence of sources Equation (7) may be written for a flow $\mathbf{u}$ in any number of dimensions as

$$\frac{\partial f}{\partial t} + \mathbf{u}.\nabla f = 0 \tag{27}$$

To proceed to derive a widely used class of surrogate for models involving Lie derivatives, it is helpful to run over a small amount of mathematics. The streamlines of a vector field such as $\mathbf{u}$ by definition satisfy

$$\frac{d\mathbf{x}}{ds} = \mathbf{u}(\mathbf{x}) \tag{28}$$

where $\mathbf{x}(s)$ is the streamline parameterised by $s$, and where often $s$ is chosen to be arc-length along the curve. Hence substituting Equation (28) in $\mathbf{u}.\nabla f$ gives the Lie derivative as $df/ds$ which illustrates its coordinate independence, since $s$ is an almost arbitrary label. Further, substituting Equation (28) in Equation (27) gives

$$\frac{\partial f}{\partial t} + \frac{\partial f}{\partial s} = 0 \tag{29}$$

where now $f(s,t)$. Equation (29) is well-known to have the exact solution $f = F(t-s)$ for arbitrary scalar functions of a single variable $F$. In realistic applications, $S \neq 0$, but the property that $f$ is a function only of arclength (and time) survives, so that the physics of sources and sinks may be dealt with by a surrogate which has only one space dimension supposing the streamlines are known. Moreover, in the plasma context, the main flow may often be directed along lines of magnetic field $\mathbf{B}$, so that exactly the same statements could be made replacing streamline with fieldline.

Now, Equation (25) can be expressed in Lie derivative form by eliminating $\frac{\partial n}{\partial t}$ using Equation (16), and division by $n$, giving

$$\frac{\partial U}{\partial t} + U\frac{\partial U}{\partial x} = -\frac{1}{n}\left(\frac{\partial p}{\partial x} + G + S_1 - US_0\right) \tag{30}$$

Thus provided $\mathbf{u}$ is expressed in Cartesians, each component separately obeys an equation of the form Equation (27) and similar remarks apply. There is also [23] a (pseudo-)vector Lie formulation for $\mathbf{B}$ and the vorticity $\nabla \times \mathbf{u}$ which can lead to similar simplifications and potential surrogates.

## 4.3 Surrogates for advection

Equations(7) and (8) combine to give an equation identical to that for mass conservation in 2-D, namely

$$\frac{\partial f}{\partial t} + \frac{\partial (u_x f)}{\partial x} + \frac{\partial (u_y f)}{\partial y} = S \tag{31}$$

where now $f(x,y,t)$ is the fluid density as a function of time $t$, and $u_x$ and $u_y$ are components of the flow $\mathbf{u}$. Source terms may still be represented as $S(x,y,t)$, collision terms may be represented as a further contribution to $S(x,y,t)$, of form

$$S_d = \frac{\partial}{\partial x}\left(\kappa_{xx}\frac{\partial f}{\partial x} + \kappa_{xy}\frac{\partial f}{\partial y}\right) + \frac{\partial}{\partial y}\left(\kappa_{yx}\frac{\partial f}{\partial x} + \kappa_{yy}\frac{\partial f}{\partial y}\right) \tag{32}$$

The above 2-D expressions should be sufficient for understanding how to deal with many situations involving advection, diffusion and sources, which are more generally formulated as

$$\frac{\partial f}{\partial t} + \nabla \cdot (\mathbf{u} f - \kappa \nabla f) = S \tag{33}$$

for tensor diffusion coefficient $\kappa$.

Analogous to Section 4.1, integrating Equation (31) over coordinate $y$, ie. forming $\int \cdot dy$ leads to

$$\frac{\partial \int f dy}{\partial t} + \frac{\partial}{\partial x} \int u_x f dy + [u_y f]_{y=\pm y_b} = \int S dy \tag{34}$$

There is then a simplification to a 1-D surrogate upon supposing that as in Section 4.1 $u_x$ may be written

$$u_x = U(x) + \tilde{u}(x, y) \tag{35}$$

and introducing as before

$$n = \int f dy \tag{36}$$

to give without further approximation, the equation (cf. Equation (16)

$$\frac{\partial n}{\partial t} + \frac{\partial nU}{\partial x} = S_0 \tag{37}$$

where $S_0$ includes the difference in mass fluxes at the boundaries $y = \pm y_b$. However, if instead of Equation (35) is assumed

$$u_x = V(y) + \tilde{v}(x, y) \tag{38}$$

then it is necessary to assume a weak dependence of $f$ on $y$ to give an equivalent of Equation (37) that is necessarily approximate as

$$\frac{\partial n}{\partial t} + \frac{1}{2y_b} \int V dy \cdot \frac{\partial n}{\partial x} = S_0 \tag{39}$$

The production of a so-called 0-D surrogate is achieved by further integrating over coordinate $x$, ie. forming $\int \cdot dx$ of Equation (34):

$$\frac{d \int \int f dx dy}{dt} + \left[ \int u_x f dy \right]_{x=\pm x_b} + \left[ \int u_y f dx \right]_{y=\pm y_b} = \int \int S dx dy \tag{40}$$

This is evidently an ODE for the total mass $n_{tot}(t) = \int \int f dx dy$ in terms of the total source $S_{tot}(t) = \int \int S dx dy$ provided the boundary fluxes are known. Note that diffusion may be explicitly included in the above expressions Equations(34) and (40) on making the replacements

$$u_x f \quad \longrightarrow \quad u_x f - \kappa_{xx} \frac{\partial f}{\partial x} - \kappa_{xy} \frac{\partial f}{\partial y} \tag{41}$$

$$u_y f \quad \longrightarrow \quad u_y f - \kappa_{yx} \frac{\partial f}{\partial x} - \kappa_{yy} \frac{\partial f}{\partial y} \tag{42}$$

Again analogously to Section 4.1, form $\int \cdot \psi dy$ for an arbitrary scalar $\psi$ and rearrange $f$-derivative terms, leading to

$$\frac{\partial n\Psi}{\partial t} + \frac{\partial}{\partial x}\int u_x \psi f dy - \int u_x \frac{\partial \psi}{\partial x} f dy + [\psi u_y f]_{\pm y_b} - \int u_y \frac{\partial \psi}{\partial y} f dy = \int S\psi dy \tag{43}$$

where now also is introduced $\Psi$ such that

$$\psi = \Psi(x) + \tilde{\psi}(x, y) \tag{44}$$

and thus

$$\int \psi f dy = n\Psi \tag{45}$$

Further expanding $u_x$ and $\psi$ in Equation (43), there results, using similar manipulations to Section 4.1, but without assuming $u_x = y$ that

$$\frac{\partial n\Psi}{\partial t} + \frac{\partial nU\Psi}{\partial x} = -\frac{\partial p_\psi}{\partial x} + G_\psi + S_{1\psi} \tag{46}$$

where

$$p_\psi = \int \tilde{u}\tilde{\psi} f dy, \quad S_{1\psi} = \int S\psi dy - [\psi u_y f]_{\pm y_b} \tag{47}$$

and exploiting Equation (26)

$$G_\psi = nU\frac{\partial \Psi}{\partial x} + \int u_y \frac{\partial \tilde{\psi}}{\partial y} f dy + U\int \frac{\partial \tilde{\psi}}{\partial x} f dy + \int \tilde{u}\frac{\partial \tilde{\psi}}{\partial x} f dy \tag{48}$$

As before, using the equation of mass conservation simplifies the dynamics, resulting in

$$n\frac{\partial \Psi}{\partial t} - U\int \frac{\partial \tilde{\psi}}{\partial x} f dy = -\frac{\partial p_\psi}{\partial x} + H_\psi + S_{1\psi} - \Psi S_0 \tag{49}$$

where

$$H_\psi = \int u_y \frac{\partial \tilde{\psi}}{\partial y} f dy + \int \tilde{u}\frac{\partial \tilde{\psi}}{\partial x} f dy \tag{50}$$

As might be expected from Section 4.1, taking $\psi = u_x$ achieves only minor further simplification.

## 5 Summary

ExCALIBUR project NEPTUNE work on UQ to-date (including two UQ workshops organised by each of the two UCL grantees) has already helped established a significant number of points relevant to the design of the NEPTUNE software. The points may be dealt with under the two main headings of sampling and surrogate production, however there is a strong suggestion that sampling and the production of a surrogate should operate sequentially, meaning that the surrogate informs the choice of sample points, which in turn modifies the surrogate &c.

For sampling, there are recommendations from external NEPTUNE reports [5] that adaptive stochastic collocation (ASC) is to be preferred, supported by the study in Section 2.2.2 above. The ASC method is expected to be resilient in the face of the "curse of dimensionality", although not completely resistant [1]. It seems natural to accompany most sampling by polynomial interpolation when the spectral/hp element approach is used to represent spatial dependences in terms of $p^{th}$ order polynomials.

Section 3.3.1 above confirms that surrogates should incorporate "appropriate" physical knowledge, such as the need for say Nusselt numbers not to be less than unity. Section 3 confirms that turbulence, even when arising in the context of a 2-D fluid model, is hard. The work shows that there remain questions concerning adequate numerical resolution, probably tied up with the existence of relatively long timescale transients. In such circumstances, the additional expense of GP is likely to be worthwhile, and to become even more so with the higher dimensional models. Given the obvious need to couple models of different dimensionality, the linked GP approach to producing surrogates described external NEPTUNE report [6] (see also ref [24]) appears worth pursuing.

There is also a strong suggestion from the discussion [1] that active subspace methods be considered to help produce surrogates. Clearly UQ is an area where there is scope for the development of significant new techniques, indicating that it is worth supporting a post-doc if only in order to keep abreast of the latest research. Since, there is lacking a convincing demonstration at scale of the proposed techniques, a new grant could usefully direct a post-doc to explore practical details of implementation. These could also include the use of PCA *aka* POD in application to time-dependent problems, as described in the closing chapters of the book by Brunton and Kutz [25, § 11,12]. Probably there is no particular worry regarding PCA itself as it seems to be so widely used. Application to time-dependent problems is already being pursued by a UKAEA sponsored research student at Leeds (Alasdair Roy). One technique for identifying parameters which is known to scale because of its use in weather and climate studies is Data Assimilation (DA), discussed briefly in internal project NEPTUNE report [3, § 3], specifically by means of the Ensemble Kalman Filter (EnKF). EnKF does however rely on having good starting estimates for the parameters, now discoverable by an optimisation front-end in the UKAEA sponsored EnKF-Python software [26], and also at least crude estimates of the uncertainties in the parameters. A. Roy is exploring the possibility of using algorithms for SINDy - Sparse Identification of Nonlinear Dynamics [25, § 7.3] - to help provide the information required by DA. (Exa)scalability will need ultimately to be considered for use in NEPTUNE. There has also been consideration of the use of adjoints, see discussion in NEPTUNE report [27, § 2.4.1] of their usefulness and economy for design, and discussion in NEPTUNE report [3, § 3.2] of their use in calculating functional derivatives. Again, there is the issue of scalability to consider.

However, so as not to be too prescriptive, there are other aspects of UQ that would withstand further examination, such the utility of other sparse approximation techniques, eg. Proper generalized decomposition (PGD) described in ref [28] as example of use of tensor train ideas and already mentioned in previous call documents, along with the Discrete Empirical Interpolation Method (DEIM). In particular, a method would be useful for selecting automatically amongst the various possible techniques, having regard to application and resources available.

## Acknowledgement

*The support of the UK Meteorological Office and Strategic Priorities Fund is acknowledged.*

## References

[1] E. Threlfall and W. Arter. Assessment of which UQ methods are required to make NEPTUNE software actionable. Technical Report CD/EXCALIBUR-FMS/0035-M2.4.1, UKAEA, 2021.

[2] W. Arter and E. Threlfall. Selection of techniques for Uncertainty Quantification (in preparation). Technical Report CD/EXCALIBUR-FMS/00XX-M2.4.2, UKAEA, 2021.

[3] E. Threlfall and W. Arter. Select techniques for MOR (Model Order Reduction). Technical Report CD/EXCALIBUR-FMS/0031-M2.5.1, UKAEA, 2021.

[4] W. Arter, E. Threlfall, and J. Parker. Report on user layer design for Uncertainty Quantification. Technical Report CD/EXCALIBUR-FMS/0024-M3.1.3, UKAEA, 2020.

[5] M. Vassaux, W. Edeling, and P. V. Coveney. Review of methods and toolkits for uncertainty quantification of single and coupled-model applications. Technical Report 2047352_1-TN-01-4, UKAEA Project Neptune, 2021.

[6] D. Ming and S. Guillas. Report on suitability and potential of ROM to fusion models : A Non-intrusive ROM for Solvers with High-dimensional Outputs. Technical Report 2047352_2-TN-01, UKAEA Project Neptune, 2021.

[7] R.C. Smith. *Uncertainty Quantification: Theory, Implementation, and Applications*. SIAM, 2014.

[8] W. Arter. Numerical simulation of magnetic fusion plasmas. *Reports on Progress in Physics*, 58:1–59, 1995. http://dx.doi.org/10.1088/0034-4885/58/1/001.

[9] C. de Boor. *A practical guide to splines*. Springer, New York, 1978.

[10] J.P. Boyd. *Chebyshev and Fourier Spectral Methods*. Dover Publications, Mineola, NY, 2001. http://www-personal.umich.edu/ jpboyd/BOOK_Spectral2000.html.

[11] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis, 3rd Edition*. Springer, 2002.

[12] B. Fornberg and N. Flyer. *A Primer on Radial Basis Functions with Applications to the Geosciences*. SIAM, 2015.

[13] M. Farge and K. Schneider. Wavelet transforms and their applications to mhd and plasma turbulence: a review. *Journal of Plasma Physics*, 81(6):435810602, 2015.

[14] P. Challenor. Private communication. Technical Report 2049XXX-TN-01, UKAEA Project Neptune, 2021.

[15] X. Liu and S. Guillas. Dimension reduction for Gaussian process emulation: An application to the influence of bathymetry on tsunami heights. *SIAM/ASA Journal on Uncertainty Quantification*, 5(1):787–812, 2017.

[16] B. Dudson, P. Hill, D. Dickinson, J. Parker, A. Allen, G. Breyiannia, J. Brown, L. Easy, S. Farley, B. Friedman, E. Grinaker, O. Izacard, I. Joseph, M. Kim, M. Leconte, J. Leddy, M. Liten, C. Ma, J. Madsen, D. Meyerson, P. Naylor, S. Myers, J. Omotani, T. Rhee, J. Sauppe, K. Savage, H. Seto, D. Schwrer, B. Shanahan, M. Thomas, S. Tiwari, M. Umansky, N. Walkden, L. Wang, Z. Wang, P. Xi, T. Xia, X. Xu, H. Zhang, A. Bokshi, H. Muhammed, M. Estarellas, and F. Riva. BOUT++ v4.3.2, March 2020.

[17] B.D. Dudson. BOUT++ website. `https://boutproject.github.io/`, 2020. Accessed: June 2020.

[18] P. A. Hill, J. T. Parker, D. Dickinson, and B. D. Dudson. BOUT++ VECMA hackaton repository. `https://github.com/boutproject/VECMA-hackathon`, 2021. Accessed: June 2021.

[19] Thomas Gerstner and Michael Griebel. Dimension–adaptive tensor–product quadrature. *Computing*, 71(1):65–87, 2003.

[20] D. Moxey, C.D. Cantwell, Y. Bao, A. Cassinelli, G. Castiglioni, S. Chun, E. Juda, E. Kazemi, K. Lackhove, J. Marcon, et al. Nektar++: enhancing the capability and application of high-fidelity spectral/hp element methods. *Computer Physics Communications*, 249:107110, 2020.

[21] D. Moxey et al. Nektar++ website. `https://www.nektar.info`, 2020. Accessed: June 2020.

[22] J.W. Elder. Numerical experiments with free convection in a vertical slot. *Journal of Fluid Mechanics*, 24:823–843, 1965.

[23] W. Arter. Potential Vorticity Formulation of Compressible Magnetohydrodynamics. *Physical Review Letters*, 110(1):015004, 2013. `http://dx.doi.org/10.1103/PhysRevLett.110.015004`.

[24] D. Ming and S. Guillas. Linked Gaussian Process Emulation for Systems of Computer Models using Matérn Kernels and Adaptive Design. *arXiv preprint arXiv:1912.09468*, 2021.

[25] S.L. Brunton and J.N. Kutz. *Data-driven science and engineering: Machine learning, dynamical systems, and control.* CUP, 2019.

[26] W. Arter, A. Osojnik, C. Cartis, G. Madho, C. Jones, and S. Tobias. Data assimilation approach to analysing systems of ordinary differential equations. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS), Florence, Italy*, number 8351751, 27-30 May 2018. http://dx.doi.org/10.1109/ISCAS.2018.8351751.

[27] W. Arter, L. Anton, and D. Samaddar. Year One Summary Report. Technical Report CD/EXCALIBUR-FMS/0011-1.00-M1.2.1, UKAEA, 2020.

[28] F. Chinesta, R. Keunings, and A. Leygue. *The Proper Generalized Decomposition for Advanced Numerical Simulations: A Primer*. Springer Briefs in Applied Science and Technology, 2014.

## A Example BOUT++ decoder

```
class BaseBOUTDecoder(BaseDecoder, decoder_name="bout++-base"):
    def __init_subclass__(cls):
        super().__init_subclass__(cls.__name__)

    def __init__(self, target_filename=None):
        """
        Parameters
        ==========
        target_filename: str or None
            Filename or glob to read in (default: "BOUT.dmp.*.nc")
        """
        self.target_filename = target_filename or "BOUT.dmp.*.nc"

    @staticmethod
    def _get_output_path(run_info=None, outfile=None):
        """
        Get the path the run directory, and optionally the file outfile
        """
        if run_info is None:
            raise RuntimeError("Passed 'None' to 'run_info'")

        run_path = run_info.get("run_dir", "data")

        if not os.path.isdir(run_path):
            raise RuntimeError(f"Run directory does not exist: {run_path}")

        return os.path.join(run_path, outfile)

    def sim_complete(self, run_info=None):
        """Return True if the simulation has finished"""
        settings_filename = self._get_output_path(run_info, "BOUT.settings")

        if not os.path.isfile(settings_filename):
            return False

        # Check for normal, clean finish
        settings_file = BoutOptionsFile(settings_filename)

        return "run:finished" in settings_file

    def get_outputs(self, run_info):
        """Read the BOUT++ outputs into an xarray dataframe"""
        data_files = self._get_output_path(run_info, self.target_filename)
        return open_boutdataset(data_files, info=False)

    def get_restart_dict(self):
        """Serialise the class to a dict of JSON serialisable variables"""
        # This is possibly too magic... get the variable names from
        # the __init__ method
        init_vars = inspect.signature(self.__init__).parameters.keys()
        # Assuming that the __init__ does `self.var = var`, we can
        # just return a dict of everything
        try:
            return {var: getattr(self, var) for var in init_vars}
        except AttributeError as e:
            # Let's try to give a helpful error message if this does go wrong
            match = re.search(r"no attribute '(.*)'", str(e))
            if match is None:
                # Oops, something else
                raise e
            varname = match.group(1)
            raise AttributeError(
                f"'{varname}' doesn't seem to be a member of this instance.\nDid y
            )

    @staticmethod
    def element_version():
        return "0.1.0"


class SimpleBOUTDecoder(BaseBOUTDecoder):
    """Just collects individual variables at the last timestep"""

    def __init__(self, target_filename=None, variables=None):
        """
        Parameters
        ==========
        variables: iterable or None
            Iterable of variables to collect from the output. If None, return ever
        """
        super().__init__(target_filename=target_filename)

        # TODO: check it's an iterable or otherwise sensible type?
        self.variables = variables

    def parse_sim_output(self, run_info=None, *args, **kwargs):
        df = self.get_outputs(run_info)
```

```python

        return {
            variable: flatten_dataframe_for_JSON(df[variable][-1, ...])
            for variable in self.variables
        }

    @staticmethod
    def element_version():
        return "0.1.0"
```

## B Example BOUT++ encoder

```python
class BOUTEncoder(BaseEncoder, encoder_name="bout++"):
    def __init__(self, template_input=None):
        """Read an existing BOUT.inp file to use as a template.
        If no input is given, an empty set of options will be created
        Example
        -------
        from boutvecma.encoder import BOUTEncoder
        encoder = BOUTEncoder("data/BOUT.inp")
        """
        if template_input:
            self._options = BoutOptionsFile(template_input)
        else:
            self._options = BoutOptions()

        self.template_input = template_input

    def encode(self, params=None, target_dir=""):
        """Create a BOUT.inp file in target_dir with modified parameters.
        Sub-sections are specified using colons in the params keys
        Example
        -------
        encoder.encode({"section:key":42}, target_dir="data")
        modifies key so that the file "data/BOUT.inp" contains
        [section]
        key = 42
        """
        if params:
            options = deepcopy(self._options)

            for key, value in params.items():
                options[key] = value
        else:
            options = self._options

        # Note: Here options could be BoutOptions or BoutOptionsFile
        # so can't just use options.write
        with open(os.path.join(target_dir, "BOUT.inp"), "w") as f:
            f.write(str(options))

    def element_version(self):
        return "0.1"

    def get_restart_dict(self):
        return {"template_input": self.template_input}
```