ADVANCED THEORY AND SIMULATIONS
www.advtheorysimul.com

Building Confidence in Simulation: Applications of EasyVVUQ

*David W. Wright, Robin A. Richardson, Wouter Edeling, Jalal Lakhlili, Robert C. Sinclair, Vytautas Jancauskas, Diana Suleimenova, Bartosz Bosak, Michal Kulczewski, Tomasz Piontek, Piotr Kopta, Irina Chirca, Hamid Arabnejad, Onnie O. Luk, Olivier Hoenen, Jan Węglarz, Daan Crommelin, Derek Groen, and Peter V. Coveney**

Validation, verification, and uncertainty quantification (VVUQ) of simulation workflows are essential for building trust in simulation results, and their increased use in decision-making processes. The EasyVVUQ Python library is designed to facilitate implementation of advanced VVUQ techniques in new or existing workflows, with a particular focus on high-performance computing, middleware agnosticism, and multiscale modeling. Here, the application of EasyVVUQ to five very diverse application areas is demonstrated: materials properties, ocean circulation modeling, fusion reactors, forced human migration, and urban air quality prediction.

1. Introduction

In order for the results of computational science to become widely accepted components of decision making processes, such as in medicine and industry, it is essential that we quantify the trust one can have in the model in question. Confidence can only be gained by ensuring not only that simulation codes are solving the correct governing equations (validation), but they are solving them correctly (verification) and we have a comprehensive estimate of the uncertainties in the result uncertainty quantification).[1,2] Collectively the processes involved in evaluating our level of trust in the results obtained from models are known as VVUQ. While the need for rigorous model assessment is widely acknowledged, it is far from being universally implemented within the scientific literature. The reasons for this are wide ranging, but include lack of specialist knowledge of VVUQ techniques and, until recently, the difficulty in obtaining sufficient computational power to perform the necessary sampling in large scale simulations.

We have recently developed EasyVVUQ,[3] a package designed to help leverage recent advances in the scale of computational resources to make state of the art VVUQ algorithms available and accessible to a wide range of computational scientists. EasyVVUQ is a component of the VECMA open source toolkit (http://www.vecma-toolkit.eu), which provides tools to facilitate the use of VVUQ techniques in multiscale, multiphysics applications.[4]

In order to enable straightforward computations of EasyVVUQ scenarios on HPC resources, the tool has been designed to work with a variety of middleware technologies, such as FabSim3[5] or

Dr. D. W. Wright, Dr. R. A. Richardson, Dr. R. C. Sinclair, I. Chirca, Prof. P. V. Coveney
Centre for Computational Science
Department of Chemistry
University College London
London WC1H 0AJ, UK
E-mail: p.v.coveney@ucl.ac.uk
Dr. W. Edeling, Prof. D. Crommelin
Centrum Wiskunde & Informatica
Science Park 123, Amsterdam 1098 XG, The Netherlands
Dr. J. Lakhlili, Dr. O. O. Luk, Dr. O. Hoenen
Max-Planck Institute for Plasma Physics, Garching
Boltzmannstraße 2, Garching bei München 85748, Germany
The ORCID identification number(s) for the author(s) of this article can be found under https://doi.org/10.1002/adts.201900246

DOI: 10.1002/adts.201900246

B. Bosak, M. Kulczewski, T. Piontek, P. Kopta
Poznań Supercomputing and Networking Center
ul. Jana Pawła II 10, Poznań 61-139, Poland
Dr. D. Suleimenova, Dr. H. Arabnejad, Dr. D. Groen
Brunel University London
Uxbridge UB8 3PH, UK
Prof. D. Crommelin
Korteweg-de Vries Institute
University of Amsterdam
Science Park 105-107, Amsterdam 1098 XG, The Netherlands
Prof. P. V. Coveney
Informatics Institute
University of Amsterdam
Amsterdam 1090 GH, Netherlands
Dr. V. Jancauskas
Leibniz Supercomputing Centre
Boltzmannstraße 1, Garching bei München 85748, Germany
Prof. J. Węglarz
Institute of Computing Science
Poznan University of Technology
Piotrowo 2, Poznań 60-965, Poland

QCG.[6] The integration with pilot job mechanisms, in particular with QCG-PilotJob[7] and Dask JobQueue, allowed us to bypass limitations of regular queuing systems related to the scheduling of workloads composed of a very large number of relatively short tasks.

In this paper, we showcase the use of EasyVVUQ in a selection of applications chosen to have highly varied computational and VVUQ requirements. The examples come from a wide range of domains; materials science, climatology, fusion physics, forced population displacement, and environmental science. All of the examples come from active research projects and were chosen to highlight the range of capabilities of EasyVVUQ:

1. Materials—a simple parameter sweep performed using a computationally expensive molecular dynamics simulation;
2. Ocean circulation— estimation of Sobol sensitivity indices using stochastic collocation in a differential equation based model;
3. Fusion—estimation of Sobol sensitivity indices using the polynomial chaos expansion in a multiscale simulation workflow;
4. Forced migration—estimation of Sobol sensitivity indices in an agent based model;
5. Environmental—estimating uncertainties using stochastic collocation in a model forecasting urban air quality.

2. EasyVVUQ

EasyVVUQ is a Python library, developed within the VECMA project, designed to simplify the implementation of VVUQ workflows in new or existing applications. The library is designed around a breakdown of such workflows into four distinct stages; sampling, simulation execution, result collation (or aggregation), and analysis. In the sampling stage, the uncertainty on the inputs of the model are defined, for instance, by specifying independent probability density functions $p(\xi_i)$ for each model parameter ξ_i. This leads to a sampling plan, that is, a collection of points in the input space $\boldsymbol{\xi}$ where the model must be executed. This execution stage is deemed beyond the remit of the package (it can be handled for instance by Dask JobQueue, FabSim3,[5] QCG-PilotJob[8], RADICAL Cybertools,[9] etc.) but EasyVVUQ does provide some functionality to address it. The final collation and analysis stages, which are handled by EasyVVUQ, deal with post processing the simulation outcomes into mean predictions, uncertainty estimates, and sensitivity measures.

A common object, the Campaign, contains information on the application being analyzed alongside the runs mandated by the sampling algorithm being employed, and is used to transfer information between each stage. All applications outlined below share a similar Campaign creation step, up until the point where a specific sampler and input uncertainties are selected. This general procedure consists of creating an EasyVVUQ Campaign object, defining the parameter space and code outputs, and selecting an encoder, decoder, and collation element. The following code can be used as a generic template for all applications we consider (up to sampler selection), where variables indicated by $< \cdot >$ have to be replaced with application-specific values.

```
# import EasyVVUQ
import easyvvuq as uq

# Set up a fresh EasyVVUQ campaign
my_campaign = uq.Campaign(name="<campaign_name>", work_dir=<"path_work_directory">)

# Define parameter space for the uncertain parameters (2 in this case)
params = {
    "param_1": {
        "type": "<type_param_1>",   #e.g. "float"
        "min": <min_param_1>,
        "max": <max_param_1>,
        "default": <default_value_param_1>},
    "param_2": {
        "type": "<type_param_2>",
        "min": <min_param_2>,
        "max": <max_param_2>,
        "default": <default_value_param_2>},
    "out_file": {
        "type": "string",
        "default": "<outfile_filename>"}}

```

```
    # The name of the output file, and the columns containing
    # the quantities of interest (qois)
    output_filename = params["out_file"]["default"]
    output_columns = ["<qoi_1>", "<qoi_2>", "<qoi_3>", "<qoi_4>"]

    # Create an encoder, decoder and collation element
    encoder = uq.encoders.GenericEncoder(
        template_fname= "<path_to_input_template>",
        delimiter='$',
        target_filename="<input_filename>")
    decoder = uq.decoders.SimpleCSV(target_filename=output_filename,
                                    output_columns=output_columns,
                                    header=0)
    collater = uq.collate.AggregateSamples()

    # Add the app (automatically set as current app)
    my_campaign.add_app(name="<app_name>",
                        params=params,
                        encoder=encoder,
                        decoder=decoder,
                        collater=collater)
```

Most such variables are self explanatory, hence we only highlight

- `"<path_to_input_template>"`: This is the path to the input template of a particular application. Essentially, this is just the standard input file that the application uses, except that the value of uncertain variables (those in `params`), must be flagged by a delimiter (`$` in this case, e.g., `param_1=$param_1`), such that they will change their values for each sample.
- `"<input_filename>"`: This is the file name that will be given to each realisation of the input template.

Note also that the parameter space definition (in Listing 1) has optional specification of the type and minimum/maximum allowed values. EasyVVUQ's Cerberus dependency uses this information to apply verification of input variables such as type, range, and conditional checks.[10] EasyVVUQ additionally provides version checking for the library (and each of the component VVUQ elements) so that the user is made aware when a given element they have been using in the past may now have a new algorithm/behavior. This functionality, along with detailed logging of element application and "fail-early" checks, is intended to aid the user in verifying that a VVUQ workflow is doing what was intended.

For more information on the various EasyVVUQ elements, we refer to the software release publication.[3] The following sections detail the use of EasyVVUQ as applied to a variety of different application domains. All can be considered to have gone through the Campaign creation process as described above, hence we will not repeat this setup code. Only the assignment of input distributions, sampler selection, and post-processing will be described for each example application. Relevant information, code, and output data for the following example applications may be found in Supporting Information.[11]

3. Goals

We intend, through the five following example sections, to demonstrate how EasyVVUQ can be used to augment existing applications with VVUQ features or capabilities, notably:

A) In a non-intrusive manner (all solvers may be used as "black boxes", with no changes to their internals). This applies to all five example applications.
B) Favouring consistency and interoperability between approaches (a particular UQ approach may be painlessly swapped for another due to EasyVVUQ's standard interface for VVUQ elements). In this work, we demonstrate a basic parameter sweep (Section 5), stochastic collocation (Sections 6, 8, and 9), and polynomial chaos expansion (Section 7), showing a similar pattern of application.
C) Combining VVUQ elements together into single elements (to create complex) behavior easily using small, existing parts). This is demonstrated with Encoders in the Fusion (Section 7) and UrbanAir (Section 9) applications.
D) Allowing execution of generated runs in any order, using any desired middleware (of particular importance to HPC applications, where job submission and execution patterns are key to performance and highly dependent on the computing resources). This design principle is demonstrated by the mix of execution methods used in the five example applications, ranging from local or manual execution through to dynamic pilot job schedulers.

The focus is not on the scientific results of each section, but on the consistency of the approach when applied to different techniques, for different solvers from different scientific domains. EasyVVUQ seeks to abstract out both the underlying model (with its application specific inputs and execution needs) and the

implementation of VVUQ (particularly UQ) algorithms. These algorithms, which may be custom implementations in EasyVVUQ or sourced from existing libraries such as chaospy or SAlib, all interact via standardized interfaces, such that the user should not have to worry about the provenance of the underlying implementation, but rather about connecting the operations together (or swapping them for others).

4. Background in UQ Techniques

This section gives a brief background in the various UQ techniques which are used in the example applications.

4.1. Stochastic Collocation

Once an input distribution is defined, the output quantities of interest (QoIs) become random variables. The stochastic collocation (SC) method creates a polynomial approximation of a quantity of interest q in the stochastic space $\boldsymbol{\xi} \in \mathbb{R}^d$ via the following expansion:

$$q(\boldsymbol{\xi}) \approx \sum_{j=1}^{N_p} q_j(\boldsymbol{\xi}_j) L(\boldsymbol{\xi}) \tag{1}$$

Here, the stochastic space $\boldsymbol{\xi}$ is the space of the uncertain code input parameters, for which independent, user-specified, probability density functions (pdfs) must be provided: $\xi_i \sim p(\xi_i)$, $i = 1, \cdots, d$. Furthermore, q_j are the code samples which are computed on a structured multi-dimensional grid, and N_p is the total number of collocation points, that is, the total number of code evaluations. The samples q_j are interpolated to an arbitrary point within the stochastic space $\boldsymbol{\xi}$ by means of Lagrange interpolation polynomials $L(\boldsymbol{\xi})$. For interpolation in multiple dimensions ($d > 1$), $L(\boldsymbol{\xi})$ is built as a tensor product of 1D Lagrange polynomials. The SC method, and similarly the polynomial chaos expansion (PCE) method (described briefly in the next section), are well-known and we refer to ref. [12] for more details on these techniques. Suffice it to say that the tensor product construction yields an exponential increase in N_p with the number of uncertain variables d and the chosen polynomial order, an example of the familiar "curse of dimensionality." However, for moderate values of d, the SC and PCE methods can display exponential convergence with N_p, thereby outperforming Monte Carlo sampling.[12]

There are three main uses of the SC expansion (1). First, the N_p code samples q_j can be used to estimate the first two moments of q in the stochastic space, giving a mean prediction and an estimate of the output uncertainty due to the prescribed distributions on the inputs. Second, (1) acts as a computationally inexpensive surrogate model for the code. Using the Lagrange polynomials, the code samples q_j (evaluated at a specific parameter values $\boldsymbol{\xi}_j$), can be interpolated to an unsampled location $\boldsymbol{\xi}$. Finally, the SC expansion is amenable to variance-based global sensitivity analysis. Estimates of the well-known Sobol sensitivity indices can be obtained from Equation (1) as a post processing step, (which is outlined in Section 4.3).

4.2. Polynomial Chaos

The PCE method is an expansion technique that is closely related to SC method presented in Section 4.1. Whereas in SC we build Lagrange interpolation functions for known coefficients, in PCE we estimate coefficients for known orthogonal polynomial basis functions. Here, we can approximate the quantity of interest q with the following expansion:

$$q(\boldsymbol{\xi}) \approx \sum_{j=1}^{N_p} c_j P_j(\boldsymbol{\xi}) \tag{2}$$

In this equation, $\boldsymbol{\xi}$, c_j, and N_p are the uncertain parameter, expansion coefficients, and number of expansion factors,[13,14] respectively. The polynomials P_j are chosen such that they are orthogonal to the input distributions, which differ from the SC expansion in Equation (1).

To compute the c_j coefficients, two variants have been implemented: spectral projection and linear regression. In the spectral projection variant, we project the response against each basis function (composed of the polynomials set (P_j)) and we exploit their orthogonality properties to extract each coefficient. In the linear regression variant (also known as point collocation), we use a least squares method that minimizes a normed difference between the PC expansion and the output for a set of samples; the coefficients c_j are then the solution of the resulted linear system.[12]

By using the PCE method, like the SC method, we can obtain the statistical moments (mean, standard deviation, variance, and $(100 - x)^{\text{th}}$ percentile) of the quantities of interest, and we can also provide a global sensitivity analysis in the form of Sobol indices (which is outlined in the next section).

4.3. Sobol Indices

Sobol indices are variance-based sensitivity measures of a function $q(\boldsymbol{\xi})$ with respect to its inputs $\boldsymbol{\xi} \in \mathbb{R}^d$.[15] As in the case of the SC method, an independent probability density function $p(\xi_i)$ is assigned to each input ξ_j, which makes this a global method. Local sensitivity methods, on the other hand, measure the sensitivity of q at some point $\boldsymbol{\xi}_0$ in the domain, and are uninformative away from this point. Another advantage of global methods is that they can capture the sensitivity due to higher-order interactions (several parameters changing at once).

Sobol indices are derived from the analysis of variance (ANOVA) decomposition of $q(\boldsymbol{\xi})$. This decomposes q into a sum of basis functions of increasing input dimension, which in long forms reads as:

$$\begin{aligned} q(\boldsymbol{\xi}) = {} & q_\emptyset + q_1(\xi_1) + \cdots + q_d(\xi_d) + q_{12}(\xi_1, \xi_2) + q_{13}(\xi_1, \xi_3) \\ & + \cdots + q_{d-1\,d}(\xi_{d-1}, \xi_d) + \cdots + q_{1\ldots d}(\xi_1, \cdots, \xi_d) \end{aligned} \tag{3}$$

A more concise notation is

$$q(\boldsymbol{\xi}) = \sum_{u \subseteq \mathcal{F}} q_u \tag{4}$$

where u is a multi-index and $\mathcal{F}$ is the power set of $\mathcal{U} := \{1, 2, \cdots, d\}$. Let us define ξ_u as $\xi_u := \{\xi_i | \forall i \in u\}$, that is, the set of all inputs with an index in u. Furthermore, u' is the complement of u, that is, $u \cup u' := \mathcal{U}$ and $u \cap u' = \emptyset$.

In the ANOVA decomposition, the basis functions q_u satisfy the following properties:

$$\int q_u(\xi_u) \mathrm{d}p(\xi_u) = 0, \quad \text{if } u \neq \emptyset$$

$$\int q_u(\xi_u) q_v(\xi_v) \mathrm{d}p(\xi_u \cup \xi_v) = 0, \quad \text{if } u \neq v \tag{5}$$

that is, they have zero mean and are orthogonal when integrated over the distributions. These properties hold when the basis functions are defined as

$$q_\emptyset = \int q(\boldsymbol{\xi}) \mathrm{d}p(\boldsymbol{\xi})$$

$$q_u = \int q(\boldsymbol{\xi}) \mathrm{d}p(\xi_{u'}) - \sum_{w \subset u} q_w(\xi_w) \tag{6}$$

It is perhaps more clear to write this in terms of conditional expectations:

$$\begin{aligned} q_\emptyset &= \mathbb{E}[q] \\ q_i &= \mathbb{E}[q \mid \xi_i] - q_\emptyset \\ q_{ij} &= \mathbb{E}[q \mid \xi_i, \xi_j] - q_i - q_j - q_\emptyset \\ &\cdots \end{aligned} \tag{7}$$

Hence, $q_\emptyset$ represents the mean of $q(\boldsymbol{\xi})$, and the q_i basis functions represent the effect of varying a single parameter ξ_i, minus the mean. Basis functions such as q_{ij} capture the effect of changing ξ_i and ξ_j simultaneously, minus all lower-order interactions, etc.

Therefore, the variances of these basis functions are the sensitivity measures we aim to approximate. Since the q_u have a zero mean, these are defined as

$$D_u := \mathbb{V}\mathrm{ar}[q_u] = \int q_u^2 \mathrm{d}p(\xi_u), \quad u \neq \emptyset \tag{8}$$

Using the orthogonality property of the basis functions, (8) can be rewritten as

$$D_u = \int \left(\int q(\boldsymbol{\xi}) \mathrm{d}p(\xi_{u'}) \right)^2 \mathrm{d}p(\xi_u) - \sum_{w \subset u} D_w \tag{9}$$

Expression (9) allows us to compute all D_u in increasing order, if we can compute the first integral on the right-hand side. The authors of ref. [16] developed a method to approximate this integral using the SC expansion (1) for $q(\boldsymbol{\xi})$, and similar techniques exist for the PCE method.[17] It is out of the scope of the current paper to go into detail, and we refer the interested reader to refs. [16, 17] for the mathematical details. Essentially, once all the code samples are obtained, the Sobol indices, which are defined as

$$S_u := \frac{D_u}{D} \tag{10}$$

can be approximated in a post-processing step. Here, $D := \mathbb{V}\mathrm{ar}[q] = \sum_{u \subseteq \mathcal{U}} D_u$.[15] Note that all D_u are positive, and that the sum of all possible S_u equals 1. Each D_u measures the amount of variance in the output q that can be attributed to the parameter combination indexed by u.

5. Example 1—Materials

5.1. Application Outline

Molecular dynamics (MD) simulations are often used to investigate the properties of materials,[18] including as part of multiscale material prediction applications.[19] Here, we take a well understood soft-matter system and study how calculations of its Young's modulus (stiffness) using MD can vary with the system size and starting configuration.

The system under consideration (**Figure 1**) is an epoxy resin—epoxy tetraglycidyl methylene dianiline (TGMDA) cured with polyetheramine (PEA) in a 1:1 ratio. Epoxies are thermosetting polymers. Small reactant monomer molecules have several reactive sites which create strong covalent bonds between several other molecules, forming a dense network of crosslinks. The resulting polymer network is very strong and epoxies are widely used in manufacturing, in the aerospace industry, as adhesives, and as multipurpose insulators.

MD simulations in the condensed phase are almost always periodic, which means that we only simulate a comparatively small simulation cell to approximate the bulk properties. The size of this simulation cell has many implications for computational cost and, more importantly, the scientific results it furnishes. Finite size effects, self-interaction across periodic boundary boundaries, and thermal fluctuations in small systems can all affect the simulation's outcome. We measure the Young's modulus (YM) of an epoxy system by measuring the pressure exerted along one axis before and after a small strain.

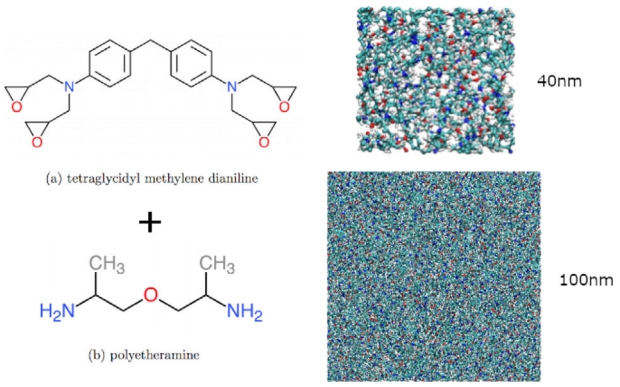


Figure 1. We test the effect of simulation size on the Young's modulus of an epoxy resin made from a 1:1 mixture of the monomers shown here.

5.2. VVUQ Algorithm

Since the instantaneous pressure of a molecular dynamics simulation can fluctuate by several GPa, it is necessary to average this value over a long sample period to measure the change in pressure due to an applied strain. The YM could also be affected by starting velocities of the atoms in the system, and the configuration of the epoxy network. To measure the system size dependence of all of these potential sources of variance, we design an EasyVVUQ Campaign that will take samples across each variable. Then, bootstrap analyses will measure their effect on the YM. A closer look at the variance due to each variable will show which is the most significant.

5.3. Execution Pattern

This application makes use of the BasicSweep sampler in EasyVVUQ, which recursively carries out a parameter sweep across the range of allowed values specified for each input variable. The system size is limited by computational cost at the high end, and the system stability at the lower. In this case, we know these approximate limits beforehand, so choose the specific range we want to sample using this method. The sampler is set up like this:

```
    sweep = {
        "box_size": [30, 40, 60, 80, 100, 150],
        "structure_no": list(range(1, 11)),
        "replica_no": list(range(1, 11)),
    }
    sampler = uq.sampling.BasicSweep(sweep=sweep)
```

In the above, we create a sampling element to sample across 6 simulation sizes, build 10 epoxy networks (structures) at each size, then measure the YM for each network starting from 10 different snapshots. These numbers are somewhat arbitrary, and more parameters could be swept depending on availability of computational resources.

Building the epoxy networks is done with an in-house developed script,[20] used in ref. [21] Simulating the epoxy network is accomplished using LAMMPS.[22] The execution of the system building procedure and measurement simulations are submitted on a remote computing resource. The "restart campaign" functionality of EasyVVUQ is required here, as the sampling and analysis stages were performed in separate python scripts. This ability to restart a campaign from a different script is useful in cases where, for example, the runs are expected to take a long time on a remote computing resource, and the user cannot or does not wish to have an EasyVVUQ script running locally, waiting for such jobs to finish.

Each replica generates three values for the YM, measured by separately straining along each principle axis of the polymer simulation. So that we can treat each of these values as equivalent measurements, we change the results pandas DataFrame format to have one row for each value; all YM values are in one column which makes some analysis more straightforward. This mode may be set using

```
campaign.set_collated_dataframe_format('one_row_per_var').
```

5.4. Results and Analysis

The system can be characterized by simply employing a boostrap analysis of the campaign.

```
bootstrap = uq.analysis.EnsembleBoot(groupby=["box_size"], qoi_cols=["Value"])
campaign.apply_analysis(bootstrap)
bootstrap_results = campaign.get_last_analysis()
```

The results of the above analysis are shown in **Figure 2**, along with a histograms of all measured YMs associated with each box size. We can clearly see that the average YM is independent of simulation size, above 4 nm. There is approximately a 20% increase in YM for a system size of 3 nm. We can safely say that for this system the characteristic length is therefore less than 4 nm.

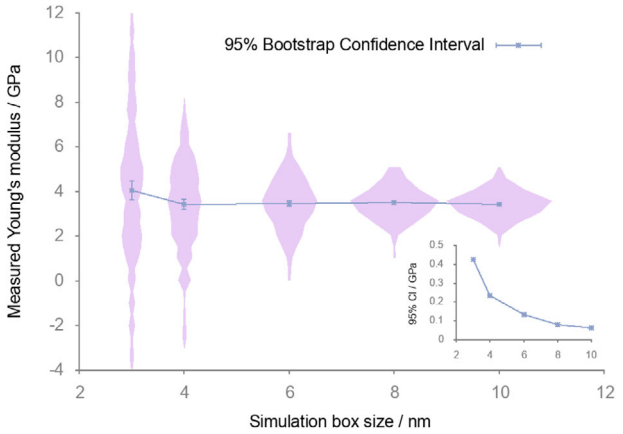


Figure 2. Young's modulus of an epoxy resin measured with different simulation sizes. Each point is the average of 300 simulations, which make up the pink histograms for each box size. The 95% bootstrap confidence interval for each simulation size is shown clearly in the bottom right insert.

We would like to know if the structure of an epoxy network has a significant bearing on the YM of a system, that is, if there is a large variation in the expected YM given an epoxy network. We approach this with the law of total variance

$$\mathrm{Var}(\mathrm{YM}) = \mathrm{Var}[\mathrm{E}(\mathrm{YM}|\#)] + \mathrm{E}[\mathrm{Var}(\mathrm{YM}|\#)] \tag{11}$$

where "#" is used to denote a specific network of cross-links. We can calculate the first and third terms of this law (the total variance in YM, and the expected variance given a specific structure) by some straightforward manipulation of the campaign results DataFrame.

```
# Retrieve the results DataFrame so we can manipulate here
results = campaign.get_collation_result()[["box_size","structure_no","Value"]]

total_var                = results.groupby("box_size").var()["Value"]
var_per_structure        = results.groupby(["box_size","structure_no"]).var()
expected_var_given_struct = var_per_structure.groupby("box_size").mean()["Value"]
var_due_to_structure     = total_var - expected_var_given_struct
```

Detailed results are shown in Supporting Information;[11] however, the analysis shows that $\mathrm{Var}[\mathrm{E}(\mathrm{YM}|\#)] \ll \mathrm{E}[\mathrm{Var}(\mathrm{YM}|\#)]$. Therefore, the epoxy network structure has no significant effect on the YM. The variance is due to the inefficient sampling of MD. We studied low strains (0.5%) because epoxies are often brittle above these strains, but simulating further (into plastic deformation) could resolve the dependence on the network structure. Chaotic dynamical systems may manifest a pathology of IEEE floating point arithmetic which was hitherto unknown,[23] providing a potentially interesting overlap between uncertainty quantification and verification in affected systems.

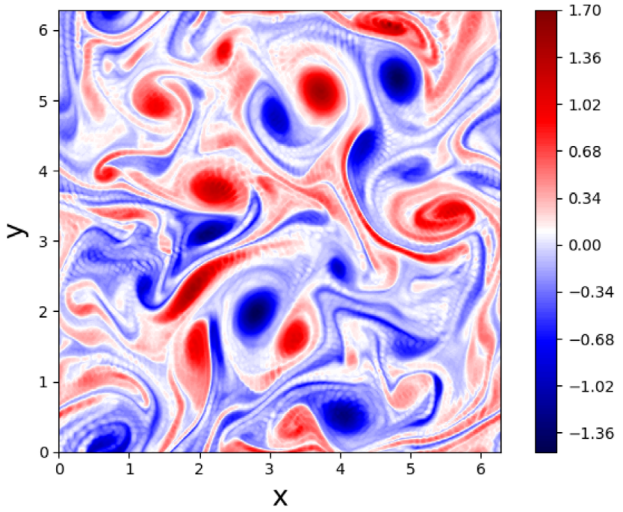


Figure 3. A snapshot of vorticity contours from Equation (12) with fully periodic boundary conditions, solved on a numerical grid of 256×256 points.

6. Example 2—Ocean Circulation

6.1. Application Outline

In this section, we consider the forced-dissipative vorticity equations for 2D incompressible flow (as described in Verkley et al.[24]), used as a simplified study for the general circulation in the oceans. The governing equations are

$$\frac{\partial \omega}{\partial t} + J(\Psi, \omega) = \nu \nabla^2 \omega + \mu(F - \omega)$$
$$\nabla^2 \Psi = \omega \tag{12}$$

Here, ω is the vertical component of the vorticity, defined from the curl of the velocity field $\mathbf{V}$ as $\omega := \mathbf{e}_3 \cdot \nabla \times \mathbf{V}$, where $\mathbf{e}_3 := (0, 0, 1)^T$. The stream function Ψ relates to the horizontal velocity components by the well-known relations $u = -\partial\Psi/\partial y$ and $v = \partial\Psi/\partial x$. The non-linear advection term is defined as

$$J(\psi, \omega) = \frac{\partial \Psi}{\partial x}\frac{\partial \omega}{\partial y} - \frac{\partial \Psi}{\partial y}\frac{\partial \omega}{\partial x} \tag{13}$$

This system generates flow fields such as those shown in **Figure 3**, which depicts a snapshot of the vorticity ω.

As in ref. [24], the forcing term is chosen as the single Fourier mode $F = 2^{3/2}\cos(5x)\cos(5y)$. The system is fully periodic in the x and y directions over a period of $2\pi L$, where L is a user-specified length scale, chosen as the Earth's radius ($L = 6.371 \times 10^6[m]$). The inverse of the earth's angular velocity Ω^{-1} is chosen as a time scale, where $\Omega = 7.292 \times 10^{-5}[s^{-1}]$. Thus, a simulation time period of a single "day" can now be expressed as $24 \times 60^2 \times \Omega \approx 6.3$ non-dimensional time units. Given these chosen length and time scales, we non-dimensionalize (12) and solve by using a spectral method with the second-order accurate AB/BDI2 time-stepping scheme.[25]

The viscosity ν and the forcing term coefficient μ are tunable parameters, and are typically set to a value such that the build up of grid-scale noise at the smallest resolved scale is prevented. In our example code, their values are computed such that a Fourier mode at this scale is exponentially damped with a user-specified e-folding time scale, that is, a time scale over which a decay of 63 % occurs ($1 - e^{-1}$). This leads to the following expressions for ν and μ:

$$\nu = \frac{1}{24 \cdot 60^2}\,\frac{1}{\Omega}\,\frac{1}{K^2}\,\frac{1}{\xi_1} \quad \text{and} \quad \mu = \frac{1}{24 \cdot 60^2}\,\frac{1}{\Omega}\,\frac{1}{\xi_2} \tag{14}$$

Here, K is the highest resolved wave number in our spectral method, which is fixed at 85. More important for our current discussion are ξ_1 and ξ_2, that is, the aforementioned damping time scales (expressed in days), which we treat as uncertain. We use EasyVVUQ to estimate the effect of this uncertainty on certain measures derived from the solution of (12). Our focus will be on the (time-dependent) energy E and enstrophy Z, defined as

$$E(t) := \frac{1}{2}\left(\frac{1}{2\pi}\right)^2 \int_0^{2\pi}\int_0^{2\pi} \mathbf{V}\cdot\mathbf{V}\mathrm{d}x\mathrm{d}y \quad \text{and}$$

$$Z(t) := \frac{1}{2}\left(\frac{1}{2\pi}\right)^2 \int_0^{2\pi}\int_0^{2\pi} \omega^2\mathrm{d}x\mathrm{d}y \tag{15}$$

Specifically, we are interested in the time-averaged statistical moments of the energy E and enstrophy Z; for example, our quantities of interest q take the form of

$$q = \int_{T_0}^{T} E(t)\mathrm{d}t =: \overline{E} \quad \text{or} \quad q = \int_{T_0}^{T} \left(E(t) - \overline{E}\right)^2 \mathrm{d}t \tag{16}$$

and likewise for the enstrophy. The integration interval $[T_0, T]$ will be defined later.

6.2. VVUQ Algorithm

For this particular problem, we will use the stochastic collocation method, as outlined in Section 4.1. In addition to the statistical moments of the aforementioned energy and enstrophy, the Sobol sensitivity indices of our damping time scales will serve as our QoIs as well. Specific implementation details are given next.

6.3. Execution Pattern

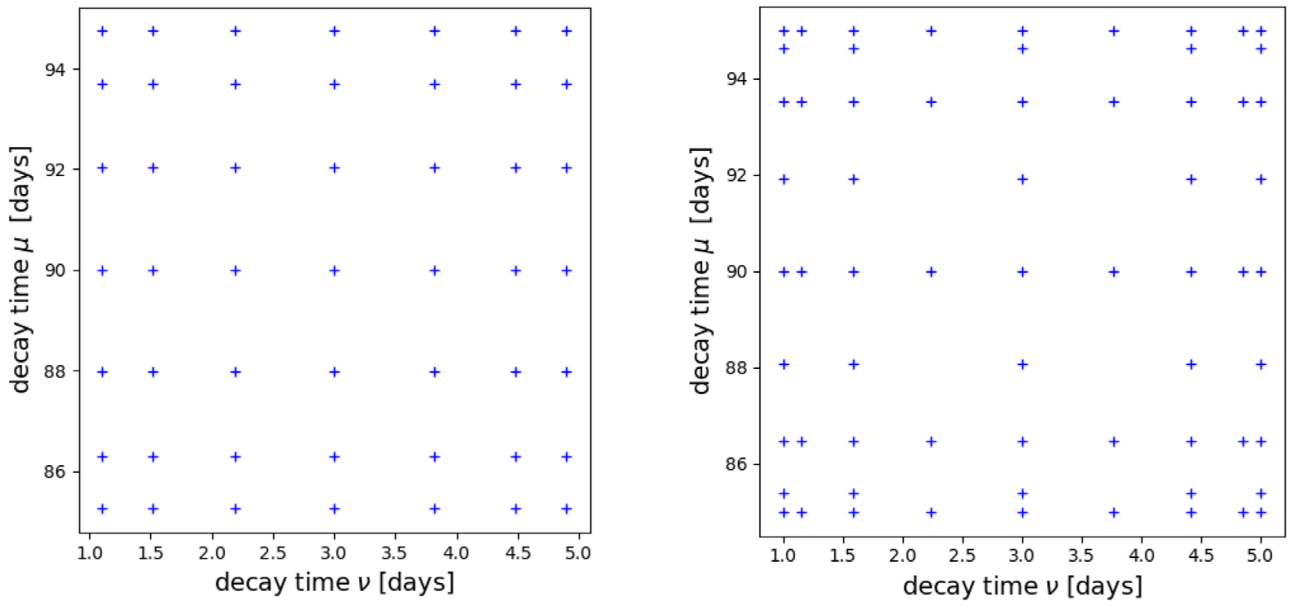
EasyVVUQ is designed to work with the Chaospy library,[26] for the specification of the input distributions. We will assume the following distributions for the uncertain decay times associated with ν and μ:

```
import chaospy as cp
vary_ocean = {"decay_time_nu": cp.Uniform(1.0, 5.0),
              "decay_time_mu": cp.Uniform(85.0, 95.0)}
```

That is, we assume that the viscous term ($\nu\nabla^2\omega$) in Equation (12) has a uniformly distributed uncertain decay time at the smallest retained scale between 1 and 5 days, whereas our forcing term is damped somewhere between 85 and 95 days. We then select the stochastic collocation sampler via

```
my_sampler = uq.sampling.SCSampler(vary=vary_ocean, polynomial_order=6),
```

By selecting a polynomial order of 6, a seven-point quadrature rule for each uncertain dimension is created. Hence, since we have two uncertain variables, we obtain a tensor grid of 49 points in the stochastic space, see **Figure 4**a. At each point, we have to evaluate the code solving (12). Instead of directly creating a full tensor product of the seven-point 1D quadrature rule, we can also construct a sparse grid (see Figure 4b), which uses a linear combination of tensor products of quadrature rules of different orders.[12] By using carefully chosen 1D quadrature rules, many points in the different tensor products will coincide, leading to a more efficient sampling plan in high dimensions. To switch to a sparse grid, one might use



(a) A full tensor grid. (b) A sparse grid.

Figure 4. Two stochastic collocation grids generated by EasyVVUQ. Each symbol is a point in the stochastic space at which the code solving Equation (12) must be evaluated.

```
my_sampler = uq.sampling.SCSampler(vary=vary_ocean, quadrature_rule="C",
                polynomial_order=3, sparse=True, growth=True)
```

Here, `quadrature_rule="C"` denotes the use of 1D Clenshaw–Curtis quadrature rules, which are a common choice in sparse grid constructions. Furthermore, `growth=True` selects an exponential growth rule, which ensures that the Clenshaw–Curtis rules are "nested" such that a quadrature rule of the next order contains all points of the previous order, leading to the aforementioned more efficient sampling plan in high dimensions. However, since we just have two uncertain variables here, we will use the full tensor product construction. Depending on the spatial resolution of the computational grid (in our case, we employ a 2D grid of 256×256 points), the cost of sampling Equation (12) at all collocation points $\boldsymbol{\xi}_j$ can be high. Moreover, since we are interested in the time-averaged statistics as in Equation (16), we must run each sample until convergence in these statistics can be safely demonstrated. We use FabSim3[5] to facilitate the execution of these samples in parallel on the Eagle supercomputer at the Poznan Supercomputing and Networking Center (PSNC). A FabSim3 plugin "FabUQCampaign" has been created to execute the ensemble run of EasyVVUQ samples on a remote resource, with minimal change in the code that is executed on the localhost. For a tutorial on the setup of FabUQCampaign, see ref. [27].

```
import fabsim3_cmd_api as fab
fab.run_uq_ensemble(my_campaign.campaign_dir, "ocean", machine="eagle_vecma")
```

The `fab` module is a wrapper around FabSim3 command-line instructions, such that these can be executed from within Python. Furthermore, `machine` specifies the name of the remote HPC resource (the PSNC Eagle cluster in our case), and `campaign_dir` is the directory containing the EasyVVUQ campaign. Finally, `"ocean"` is the name of the script which executes a single run of our model (12); see the tutorial[27] for more details. Once the ensemble run has completed, the results can be retrieved through:

```
fab.get_uq_samples(my_campaign.campaign_dir, machine="eagle_vecma")
```

If one wishes to run a (small) local ensemble for testing or debugging purposes, specifying `machine="localhost"` will make sure that everything is executed locally. Note that FabSim3 is not the only available execution interface between EasyVVUQ and HPC clusters. EasyVVUQ-QCGPilotJob is a lightweight integration code that simplifies usage of EasyVVUQ with a QCG-PilotJob execution engine; see ref. [7] for a tutorial.

6.4. Results and Analysis

In our example, $d = 2$ (ν and μ), and our quantities of interest are time-averaged moments of Equation (16) of the energy and enstrophy. For each sample of the ensemble run, Equation (12) is simulated for 11 years, and the last 10 years are used to compute the time-averaged E and Z moments. To perform the post-processing analysis of these samples, an `SCAnalysis` object is created:

```
sc_analysis = uq.analysis.SCAnalysis(sampler=my_sampler,
                                     qoi_cols=output_columns)
my_campaign.apply_analysis(sc_analysis)
results = my_campaign.get_last_analysis()
```

The `results` dictionary contains the statistical moments and the Sobol indices of the quantities of interest, the latter of which are given below for this particular case:

```
========================
Sobol indices E_mean
S(nu) = 0.6649
S(mu) = 0.3256
S(nu, mu) = 0.0096
========================
Sobol indices Z_mean
S(nu) = 0.7073
S(mu) = 0.2823
S(nu, mu) = 0.0103
========================
Sobol indices E_stdev
S(nu) = 0.4661
S(mu) = 0.0881
S(nu, mu) = 0.4458
========================
Sobol indices Z_stdev
S(nu) = 0.4971
S(mu) = 0.0775
S(nu, mu) = 0.4254
```

A value close to one means that this variable, or combination of variables, explains most of the variance in the selected output. Clearly, ν is the more influential parameter for both the time-averaged energy E and enstrophy Z. However, for the corresponding standard deviations (stdevs), μ does play an important role in the second-order Sobol index, indicating a significant interaction between ν and μ for these QoI.

In order to use the SC expansion as a surrogate model for the real code, we can draw random samples from Equation (1) via

```
#evaluate the energy surrogate at xi
sc_analysis.surrogate("E", xi)
```

Here, `xi` is an array containing a random sample from the input distributions of ν and μ. The surrogate is far cheaper than the original model, such that we can use it to evaluate the output probability density function via a kernel-density estimate (KDE). **Figure 5** shows the KDE of E, evaluated using 5×10^4 samples from (1).

7. Example 3—Fusion

7.1. Application Outline

Thermonuclear fusion is potentially a solution to the provision of base load electricity, which is carbon free and not subject to geopolitical problems. Understanding the mechanisms of heat

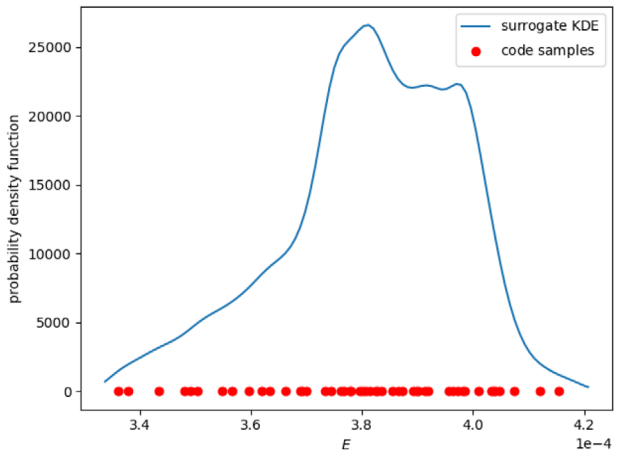


Figure 5. The kernel-density estimate of the time-averaged energy E, computed from 50 000 samples of a SC surrogate of polynomial order 6. The 49 code samples used to build the surrogate are also shown.

and particle transport in hot fusion plasma is one of the keys to obtain a cost-efficient reaction in the fusion devices. Our present understanding of the problem is that turbulence at small scales is responsible for much of this transport, but the profiles of temperature and density evolve over much larger scales.

A wide standardization effort toward integrated modeling[28] for fusion plasmas has allowed us to build modular applications in the form of a workflow. The code-to-code coupling is done via standardized data-objects[29] (referred to hereafter as CPO files), while specific parameters are stored in XML. This setup allows users to swap codes with others of different complexity. Based on this effort, a multi-scale application is developed to study the turbulence effects on plasma transport at larger scales.[30] However, much remains to be done on the validation of such simulations as well as on the control of their uncertainties. In this work, we present an early validation pattern we uncovered by extracting and comparing experimental and UQ simulation output distributions. In our application, these uncertainties originate from applied heating sources (extrinsic) and/or from the noisy, chaotic nature of the turbulence (intrinsic). We focus here on quantifying extrinsic uncertainties for the heating source as well as boundary conditions for both electron and ion temperatures. The heat source (energy per unit time) for each species is a Gaussian function with respect to the radial (or toroidal flux) coordinate ρ_{tor}, and it is characterized by its amplitude, width, and position. The boundary conditions refer to the initial temperatures at the plasma edge for both species; the edge is positioned at the maximum ρ_{tor} value, or at normalized $\rho_{tor} = 1.0$.

7.2. VVUQ Algorithm

The EasyVVUQ library provides both quasi-Monte Carlo (QMC) and polynomial Chaos expansion (PCE) methods (described in Section 4.2) that we can select from to conduct UQ and SA in the multi-scale fusion workflow.[31] In the work we present here, the PCE method was selected because it can carry out the calculations much faster than the QMC method. However, this is only valid if the number of uncertain parameters remain relatively low.

7.3. Execution Pattern

Similar to the ocean circulation example, we specify the input distributions using chaospy through EasyVVUQ. In addition, to fully benefit from the standardized interface for each code within our multi-scale workflow, we extended the EasyVVUQ base encoder with a new domain specific CPOEncoder (for boundary conditions of electron and ion temperature profiles) and a generic XMLEncoder (for electron and ion heating sources approximated by the amplitude, position and width of a Gaussian function). These format-bound encoders allow us to update real data and parameter files without having to create a template, which in turn gives us more flexibility. Since we are interested in uncertainties driven by both the heating sources and the boundary conditions for electrons and ions temperatures, we need to combine these two encoders with the MultiEncoder provided by EasyVVUQ. Therefore, the encoder creation from listing 1 is modified with the following snippet of code:

```
# Extended encoder for XML template
encoder_xml = XMLEncoder(template_filename="<path_to_input_xml_template>",
                         target_filename="<input_xml_file>",
                         common_dir=common_dir,
                         uncertain_params=uncertain_params)
# Extended encoder for CPO template
encoder_cpo = CPOEncoder(template_filename="<path_to_input_cpo_template>",
                         target_filename="<input_cpo_file>",
                         common_dir=common_dir,
                         uncertain_params=uncertain_params)
# Encoder to be used by Campaign object
encoder = uq.encoders.MultiEncoder(encoder_cpo, encoder_xml)
```

- `common_dir` is a folder that contains all required input files.
- `uncertain_params` is a python dictionary specified by the user, and it contains the list of parameters with their probability distributions types followed by the chaospy glossary.

In addition, the new encoders have a specific function that provides two dictionaries containing the names and types of all parameters to be varied and their corresponding distributions.

```
# Get parameters: vary = vary_1 + vary_2 and params = params_1 + params_2
params_1, vary_1 = encoder_xml.draw_app_params()
params_2, vary_2 = encoder_cpo.draw_app_params()
```

We set up the PCE sampler using a polynomial of order 4 to ensure good accuracy:

```
# Create the sampler
sampler = uq.sampling.PCESampler(vary=vary, polynomial_order=4)
```

The output of the application is composed of several CPO format files, so the same kind of modification is done for the creation of the decoder, which uses our domain-specific CPODecoder.

```
# Create a specific decoder for CPO output files
decoder = CPODecoder(target_filename="<output_cpo_file>",
                     output_columns=<quantities_of_interest_list>)
```

Finally, to generate all samples needed for the analysis, we can either call the function `ExecuteLocalexecute` provided directly by EasyVVUQ or resort to a wrapper enabling the execution using the QCG-PilotJob mechanism.[7,8]

7.4. Results and Analysis

To perform a post-processing analysis on the generated samples, we use the `PCEAnalysis` object from EasyVVUQ. For the results, as in the ocean circulation example, we use `analysis_results`, the output dictionary of the campaign object's `get_last_analysis()` method, in which the statistical moments, and the Sobol indices of the Quantities of Interest are stored.

The current version of the fusion workflow uses an analytical turbulence code, with four uncertain parameters (amplitude, width, position of heating source, and boundary condition). We assumed each of these parameters has a normal distribution in the range of ±20% around its original value, and as the number of samples is determined by the uncertain parameter number and polynomial degree in the PCE method, the number of runs required for this example is 1296.

The uncertainty quantification of the fusion workflow is shown in **Figures 6** and **7**. The quantities of interest are the electron and ion temperature profiles, spanning from the radial position of plasma core ($\rho_{tor} = 0$) to the edge (normalized $\rho_{tor} = 1.0$). The standard deviation indicates that the ion temperature varies weakly since the uncertainties are carried by the electrons sources. The sensitivity analysis reveals that the variance in the electron and ion temperatures is mainly due to the uncertainty from three parameters: the position and amplitude parameters of the sources at core region of the plasma and, as expected, boundary condition parameter at the edge region. The parameter width has no direct effect on the variance of the two quantities, so according to ref. [32], this parameter can be neglected and then the number of samples can be reduced while keeping the same variance behavior.

In addition to uncertainty quantification, the fusion application performs validation on the simulation results by comparing the distribution of the QoI to the distribution from experimental measurements (first results are shown in the **Figure 8**). Specifically, we create the `ValidationSimilarity` object to determine the similarity between two distribution functions:

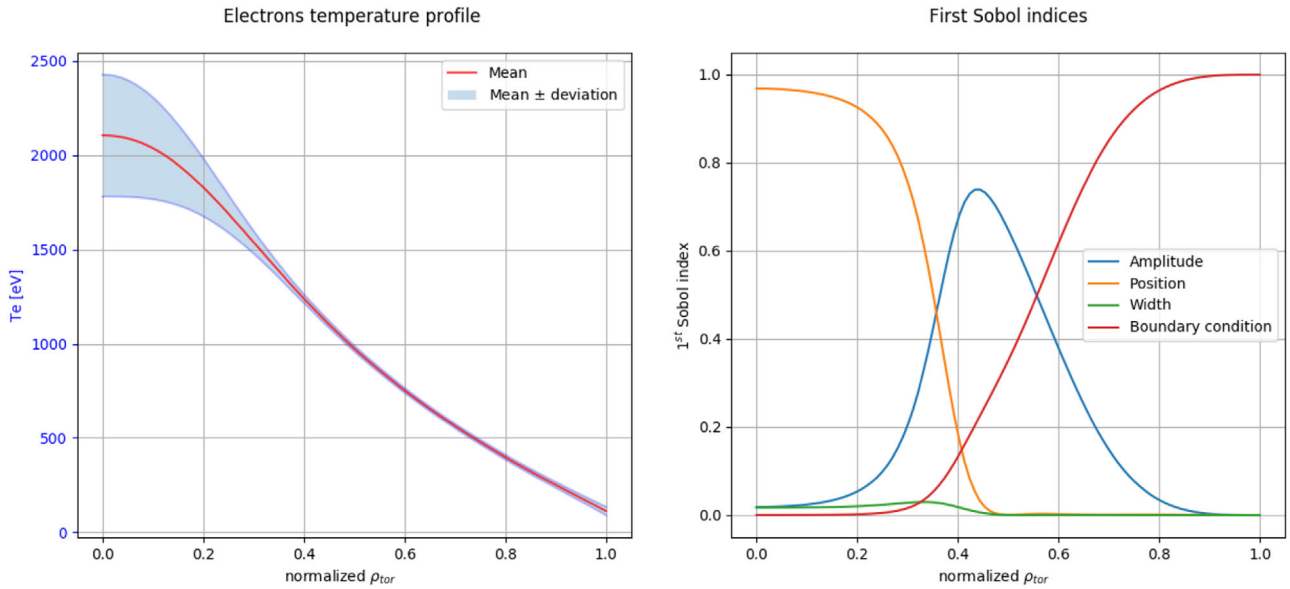


Figure 6. Descriptive statistics and sensitivity analysis of UQ example for the electron temperature.

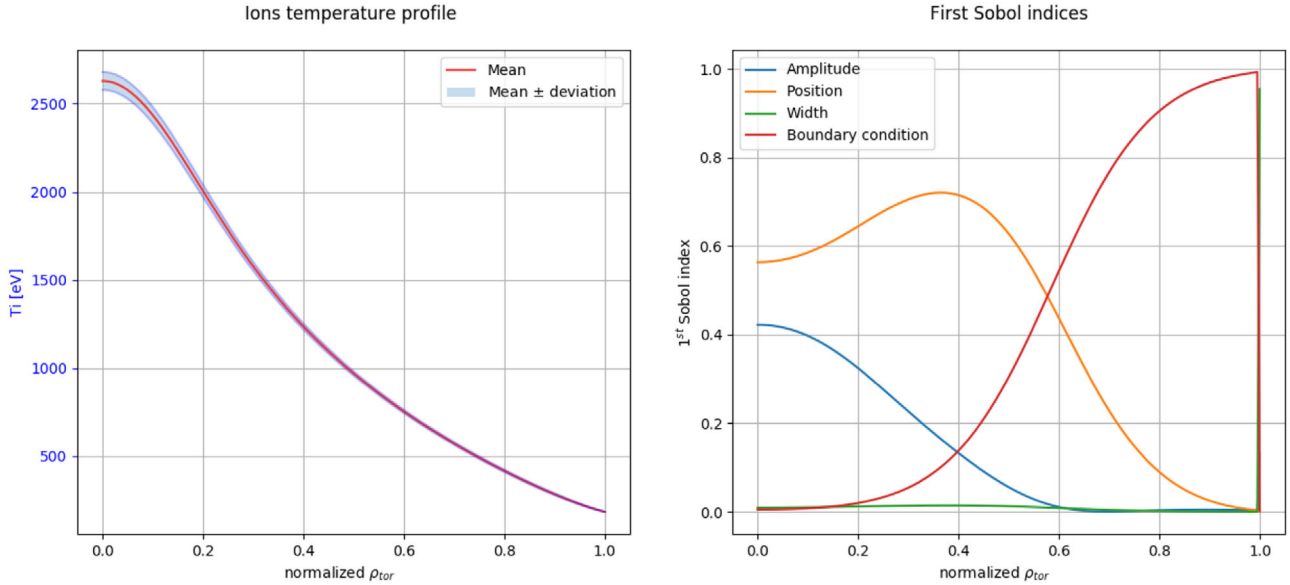


Figure 7. Descriptive statistics and sensitivity analysis of UQ example for the ion temperature.

```
# Create a specific Validation object
validation = uq.comparison.ValidationSimilarity()
validation_results = validation.compare(dataframe1=<exp_dist_values>,
                                        dataframe2=<sim_dist_values>)
```

- `exp_dist_values` is a list of approximated distributions of the given experimental samples. The samples obtained from fusion experiments contain the mean, lower, and upper threshold values; these lower and upper threshold values do not necessarily equal to each other. Therefore, we treat each sample as a two-piece normal distribution.

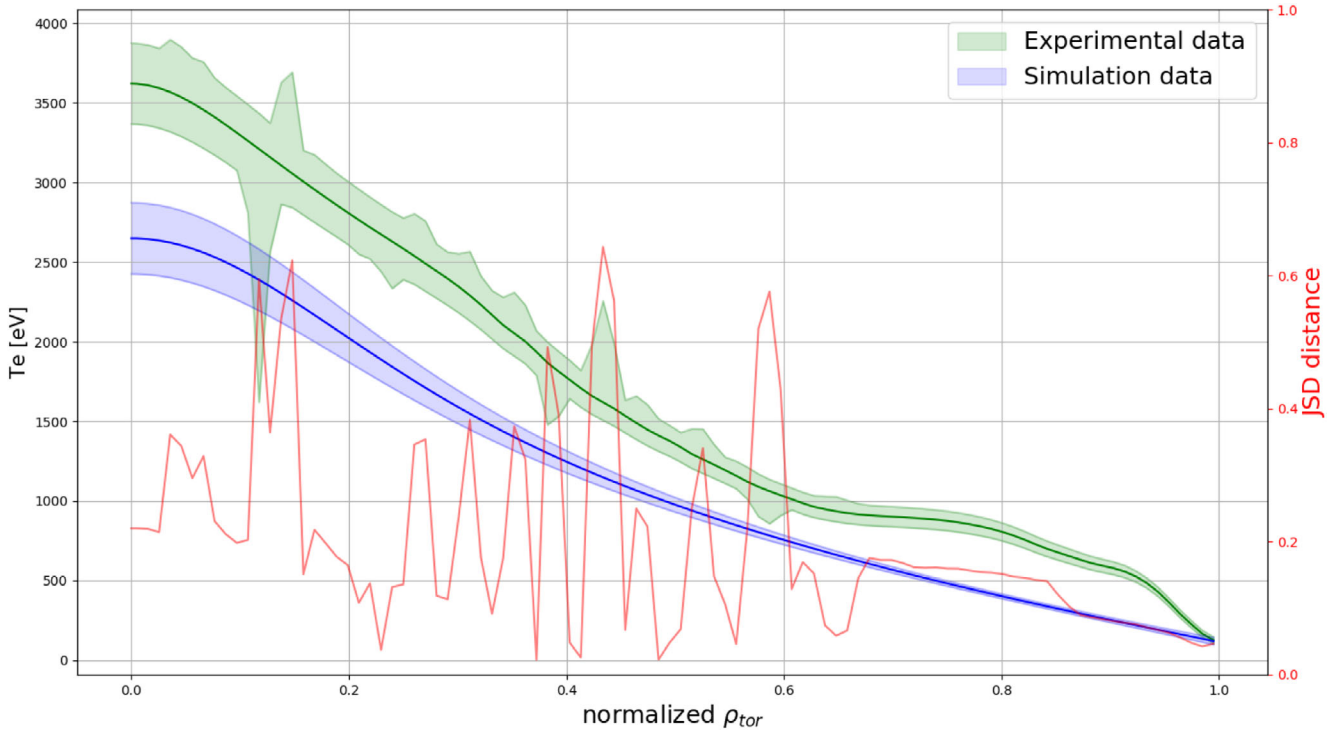


Figure 8. Validation using comparison between experimental and simulation data for electron temperatures. On the left hand-side axis, the expected values and standard deviations, and on the right hand-side axis, the Jensen–Shannon distance measuring the similarity between distributions with respect to the normalized toroidal flux coordinate ρ_{tor}.

- `sim_dist_values` is a list of output distributions given by Analysis results: `analysis_results[''output_distributions'']["QoI"]`. In EasyVVUQ, we use a function that constructs a kernel density estimator (KDE) for each polynomial by sampling it.

For the similarity measure, we use the Jensen–Shannon distance (JSD), which is a symmetrized and smoothed version of the Kullback–Leibler divergence.[33,34] It is defined by

$$JSD(P,Q) = \frac{1}{2}\sum_{i=1}^{Ns}\left(P_i \log(\frac{P_i}{M_i}) + Q_i \log(\frac{Q_i}{M_i})\right) \quad (17)$$

Here, Ns is the number of samples, P and Q are defined as two discrete probability distributions, and $M = \frac{1}{2}(P+Q)$. As presented in Figure 8, Jensen–Shannon distance takes values in the range $[0,1]$. The values closer to 0 indicate a smaller "distance" between the two distributions and therefore a stronger similarity. Two other measures based on the Hellinger and Wasserstein distances[33] are also available in EasyVVUQ. These measures were also tested on the current example, and they give equivalent results as the Jensen–Shannon distances.

8. Example 4—Forced Migration

8.1. Application Outline

Forecasting forced displacement is of considerable importance since 70.8 million people are today being forcibly displaced worldwide, a record level.[35] It is also challenging as many forced population data sets are small and incomplete, and data sources have too little information.[36] Nevertheless, forced population predictions are essential to save the lives of such migrants, to investigate the effects of policy decisions and to help complete incomplete data collections on forced population movements.

Through the use of computational approach, namely the FLEE agent-based simulation code, we predict the distribution of forced population arrivals to potential destinations as governments and NGOs can efficiently allocate humanitarian resources and provide protection to vulnerable people.[37] We represent forcibly displaced people as individual agents, combining simple rulesets for individuals to allow complex movement patterns to emerge and validate simulation results against real data. We are also able to systematically explore the possible impact of policy decisions using the FabSim3-based FabFlee toolkit while accounting for the sensitivity to a subset of parameters and assumptions in the model, such as the probability of migrants making specific moves. In **Figure 9**, we present a simulation instance of the Mali conflict, in which a number of insurgent groups began a fight for the independence of the Azawad region resulting in an increasing number of forcibly displaced people since January 16, 2012.[36,37]

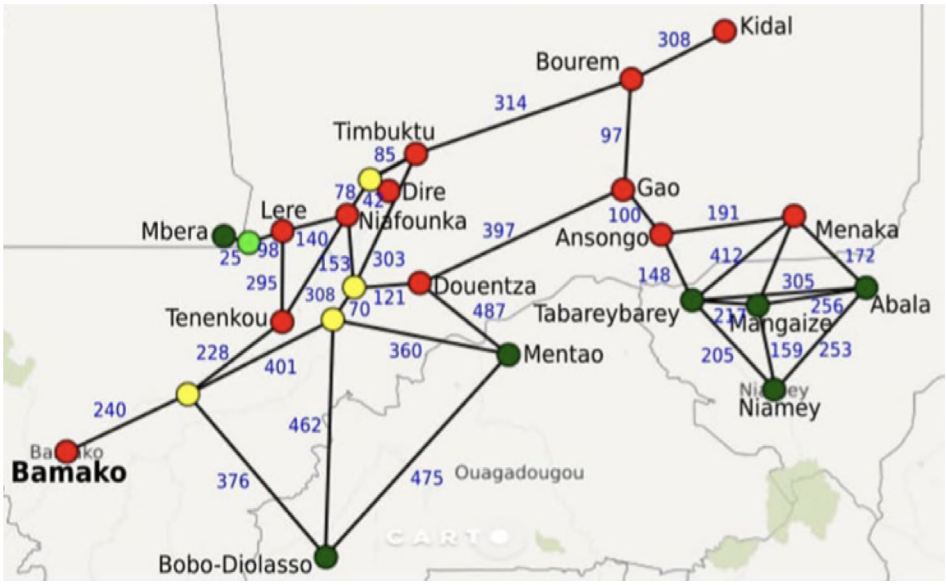


Figure 9. Overview of geographic network model for Mali, which includes conflict zones (red circles), camps (dark green circles), forwarding hub (light green circle), and other major towns (yellow circles) interconnected with straight-lines that represent roads and their length in kilometers with adjacent blue numbers.

8.2. VVUQ Algorithm

FabFlee uses the EasyVVUQ library to facilitate VVUQ for simulation analysis. It allows us to automate parameter exploration analysis and explore essential one-at-a-time input uncertainty quantification. Importantly, uncertainty quantification and sensitivity analysis are required in multiscale migration studies to understand in what regimes and scenarios our simulation approach performs well. FabSim3, EasyVVUQ, QCG-PilotJob, and other QCG components can be combined in a variety of ways, enabling users to combine their added values while retaining a limited deployment footprint. As previously mentioned, EasyVVUQ can use FabSim3 to facilitate automated execution. Users can convert their EasyVVUQ campaigns to FabSim3 ensembles using a one-line command, and the FabSim3 output is ordered such that it can be directly moved to EasyVVUQ for further decoding and analysis.

8.3. Execution Pattern

We use similar approach as described in the ocean circulation example for sensitivity analysis of forced migration application. In particular, we analyze the probability of parameters when agents move from their current location to a different one on a given day. These probabilities depend on the type of locations, namely conflict zone, camp, or other location, where agents reside.[37] We adjust these parameters to understand the importance of our assumptions in regard to the validation results. In **Figure 10**, we provide the overall workflow of forced displacement application for sensitivity analysis.

To provide the input distributions, for instance, we specify a uniformly distributed move chance probabilities for camps between 0.0001 and 1.0, as well as for conflict locations between 0.1 and 1.0 as illustrated below.

```
import chaospy as cp
vary = {"camp_move_chance": cp.Uniform(0.0001, 1.0),
        "conflict_move_chance": cp.Uniform(0.1, 1.0)}
```

Then, we set up the stochastic collocation (SC) sampler using

```
# Create the sampler
my_sampler = uq.sampling.SCSampler(vary=vary, polynomial_order=3)
```

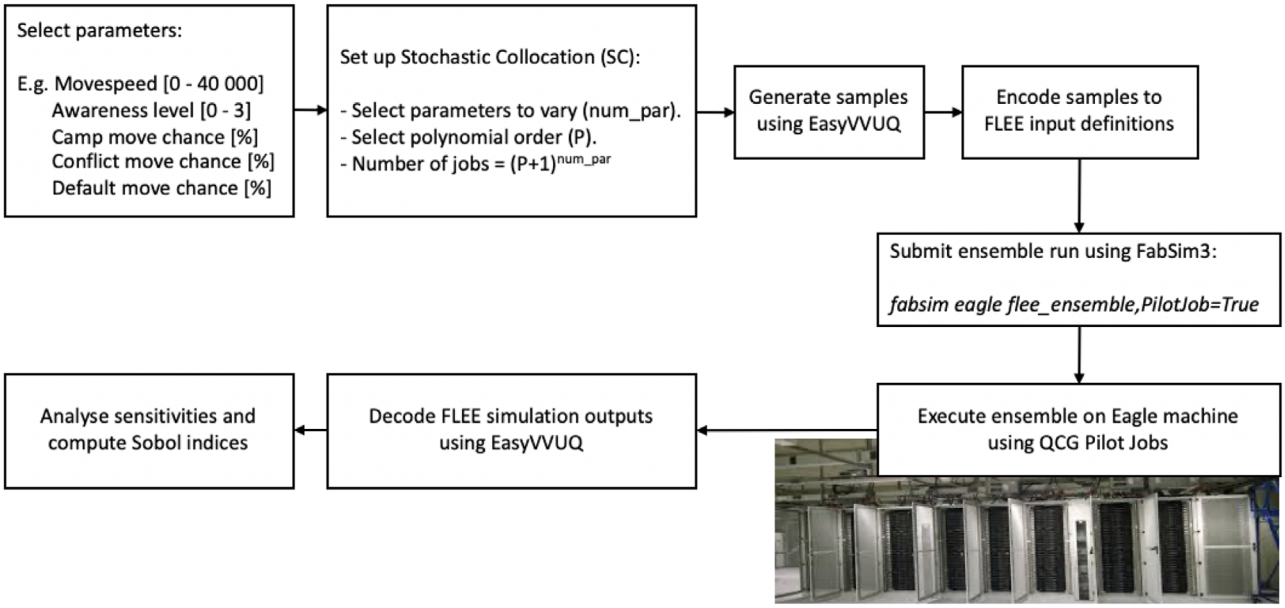


Figure 10. Overview of the FLEE workflow, where we use FabSim3 in conjunction with EasyVVUQ and QCG Pilot Job Manager.

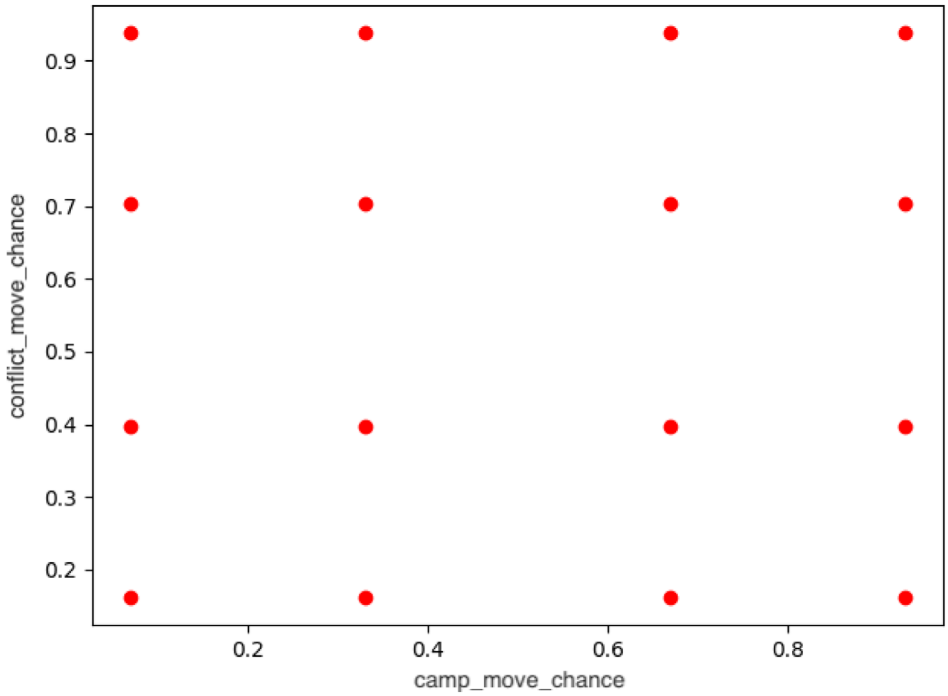


Figure 11. A stochastic collocation grid generated by EasyVVUQ for migration application parameters.

where a polynomial order is 3 in this instance. In turn, it creates a four-point quadrature rule for each move chance parameter (see **Figure 11**). EasyVVUQ encodes the generated samples to FLEE input definitions for specific conflict simulations and submits all ensemble runs for execution using FabSim3 to Eagle machine where QCG-PilotJobs schedule submitted ensemble runs and pre-reserved resources.

8.4. Results and Analysis

We apply the same SC-based Sobol index method[16] as in the ocean circulation example above to the Mali conflict, and obtain the results illustrated in **Table 1** for two parameters. We draw our own distinction on the Sobol indices by accepting parameters with values below 0.05 while identifying parameters with higher

Table 1. FabFlee and EasyVVUQ input parameter exploration results for multiscale migration application, where we vary two parameters.

Parameters	Sobol indices
camp_move_chance (0)	0.9497191
conflict_move_chance (1)	0.04978418
Combination of parameters (0, 1)	0.00049672

values as sensitive to output results. The camp_move_chance parameter is more sensitive in our model compared to the other parameter, namely conflict_move_chance, since camps are primary destination locations for forcibly displaced people fleeing from conflict locations. We also find that our models are not sensitive to the combination of these parameters.

9. Example 5—UrbanAir

9.1. Application Outline

The UrbanAir application concerns the modeling and forecasting of the concentration and dispersion of pollutants. It is a 3D multiscale model that combines a numerical weather prediction (NWP) model, running at larger scale (e.g., mesoscale), with a city-scale geophysical flow solver for accurate prediction of contaminant transportation through the street corridors, over buildings and obstacles.

The NWP model is based on the community Weather Research and Forecasting (WRF) model,[38] while the city-scale problem is solved using the EULAG model.[39] EULAG is a numerical solver for all-scale geophysical flows, with many proven scenarios, for example, flows around buildings[40] with comparison against wind tunnel experiments.[41] The coupling between WRF and EULAG model has been evaluated in ref. [42]. Typically, an emergency response situation requires fast and accurate tools. However, the use of more complex and expensive models is dictated by the need for accurate prediction of peak concentrations and plume temporal evolution.

With increased model resolution, small-scale flow characteristics are becoming more essential for prediction, and general urban parameterization coming from the NWP model is not enough. The WRF output is used as the initial and lateral boundary conditions for the EULAG simulation, along with terrain data (terrain elevation, road network, buildings shapes, and height) and emission data. **Figure 12** presents the general workflow of the application. The IMB approach is used in EULAG to explicitly resolve complex building structures, accounting for different urban aerodynamic features, such as channeling, vertical mixing, and street-level flow. The pollutant dispersion is simulated using passive tracer equations.

The NWP model may be supplemented with an additional chemistry module, to simulate chemical transportation and mixing over larger scales.[43,44]

In order to accurately simulate at small scales (grid resolutions up to 1 metre), HPC resources are required. EULAG is proven to scale up to thousands of CPU cores to support such resolution and to decrease overall time-to-solution.[45] The key problem in providing accurate forecasts is the lack of complete, well-known emission sources. Contaminants—such as NO, NO_2, PM2.5 (particulate matter under 2.5 μm in size), and PM10 in particular—are emitted by point sources (e.g., industrial chimneys), line sources (e.g., road transportation), and area

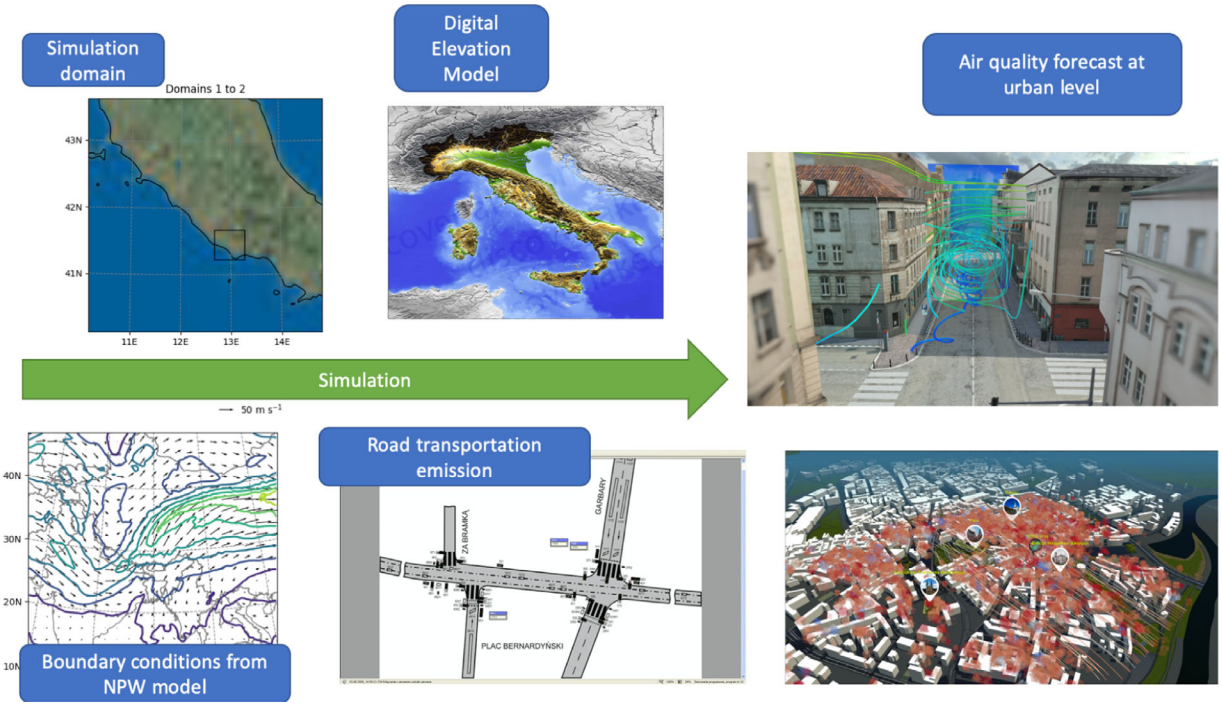


Figure 12. High-level UrbanAir application workflow.

sources (e.g., heat appliances). The uncertainty comes from unknown emission details. Taking road transportation as an example, there is a set of parameters that need to be estimated: these include the ratio of cars using gasoline to diesel fuel, fuel usage, emission index, percentage of cars that cold-started, and so on. Through the use of computational ensemble simulations, we can address these issues using statistical data, such as by combining the number of cars passing a given road section within 1 h with previously estimated parameter values.

9.2. VVUQ Algorithm

In order to assess the influence of unknowns in the emission sources, we have designed an EasyVVUQ campaign that samples across each of the input variable. It allows us to assess input uncertainty quantification and sensitivity analysis, though we concentrate on the former at the moment. The uncertainty may additionally stem from weather boundary conditions, heights of buildings, etc. To facilitate uncertainty quantification for this computationally demanding application, The QCG-PilotJob is used to choreograph the execution of the ensembles.

9.3. Execution Pattern

Currently, we focus on quantifying uncertainty coming from parameters related to NO_2 emission attributed to road transportation. The simulations require input data regarding, for example, NO_2 index from gasoline engines, fuel usage, density, and ratio of gas to diesel cars. The input distribution is specified using chaospy via EasyVVUQ. Here, we focus only on parameters related to petrol-powered vehicle, while a similar setup is needed for diesel vehicle analysis.

```
vary_gas_no2 = {
 "gas_engine": cp.Uniform(0.1, 0.9),
 "gas_usage": cp.Uniform(3.0, 13.0),
 "gas_density": cp.Uniform(0.001, 0.9),
 "gas_no2_index": cp.Uniform(0.01, 0.1),
}
```

Next we set up different samplers for different input parameters we want to be sampled.

```
emis_encoder = EmisEncoder(
    template_fname=jobdir + '/' + TEMPLATE,
    delimiter='$',
    target_filename=ENCODED_FILENAME)

wind_encoder = uq.encoders.GenericEncoder(
    template_fname = jobdir + '/' + WIND_TEMPLATE,
    delimiter='$',
    target_filename='wind.dat')

scalars_encoder = uq.encoders.GenericEncoder(
    template_fname = jobdir + '/' + SCALARS_TEMPLATE,
    delimiter='$',
    target_filename='scalars.dat')

fort13_encoder = uq.encoders.GenericEncoder(
    template_fname = jobdir + '/' + FORT13_TEMPLATE,
    delimiter='$',
    target_filename='fort.13')
```

A custom encoder is used, EmisEncoder, whose goal is to use the values of the sampled parameters as components to calculate the correct value of road transportation emissions.

```
#gas_engine - number of gasoline cars
no2_gas_emis = gas_engine * vehicles
#length - road length
#gas_usage - fuel usage per 100km
no2_gas_emis *= length * gas_usage
no2_gas_emis /= 100
#gas_density - gasoline density
#gas_no2_index - gasoline NO2 emission index
no2_gas_emis *= gas_density * gas_no2_index
#and make the emission per second
no2_gas_emis /= 3600
```

We setup a stochastic collocation sampler

```
my_sampler = uq.sampling.SCSampler(vary=vary_gas_no2,polynomial_order=1)
```

and use the multiencoder for our campaign.

```
encoders = uq.encoders.MultiEncoder(
    emis_encoder,
    wind_encoder,
    scalars_encoder,
    fort13_encoder)
```

To facilitate running ensembles, each of which requires hundreds of cores, we use an integrator between QCG-PilotJob and EasyVVUQ called EasyVVUQ-QCGPJ.[7,8]

```
qcgpjexec = easypj.Executor()
#use 32 nodes equipped with 24 cores each
qcgpjexec.create_manager(dir=my_campaign.campaign_dir, resources='32:24')
#encoding will use 1 core, one for each of the generated ensemble
qcgpjexec.add_task(Task(
    TaskType.ENCODING,
    TaskRequirements(cores=Resources(exact=1))
))
#each ensemble will run on 16 nodes
qcgpjexec.add_task(Task(
    TaskType.EXECUTION,
    TaskRequirements(nodes=Resources(exact=16),cores=Resources(exact=24)),
    application='mpirun.sh 384'
    ))

#the execution patterns is to generate ensembles, then to run simulation
#there should be 2 ensembles simulation in parallel since one requires
#16 nodes, and we requested 32 nodes for the whole campaign
qcgpjexec.run(
    campaign=my_campaign,
    submit_order=SubmitOrder.PHASE_ORIENTED)
```

9.4. Results and Analysis

In this simulation example, a $2 \times 2 \times 2$ metre grid resolution has been used, and the same resolution has been applied to the output results, which contain NO_2 concentration for each given point in 3D space. The output is then transformed into x*y columns with z NO_2 values, that is, for each point in 2D-space, there is a list of NO_2 concentration at different heights. Such data is then processed and analyzed using the SCAnalsysis object from EasyVVUQ.

```
my_analysis = uq.analysis.SCAnalysis(sampler=my_sampler,
qoi_cols=["vertical_NO2_concentrations_at_some_point_in_2D_space"])
my_campaign.apply_analysis(my_analysis)
results = my_campaign.get_last_analysis()
stats = results['statistical_moments']
        ['vertical_NO2_concentrations_at_some_point_in_2D_space']
```

The goal of the analysis is to provide us with the mean concentration (and associated uncertainty) for the whole domain at different heights, and to study how the final result may vary due to the incomplete emissions data. While at present the analysis is performed for a given point in 2D space for different heights above street level, future analyses will concern the entire 3D space.

The uncertainty quantification of the UrbanAir workflow for an arbitrary point in 2D-space is shown in **Figure 13**. Since the NO_2 concentrations is attributed to road transportation, it tends to decrease with increasing height above road level. Note that the interpolation of NO_2 concentration in between every 2 m is here due only to the plotting software. The standard deviation indicates how much uncertainty of input parameters (currently only four are taken into account) is reflected in the air quality predictions. In the forthcoming work, a sensitivity analysis will be conducted to select the most important uncertainty input parameters.

10. Discussion and Conclusions

In this work, we have applied EasyVVUQ to five diverse application areas, in order to extract information on sensitivity or

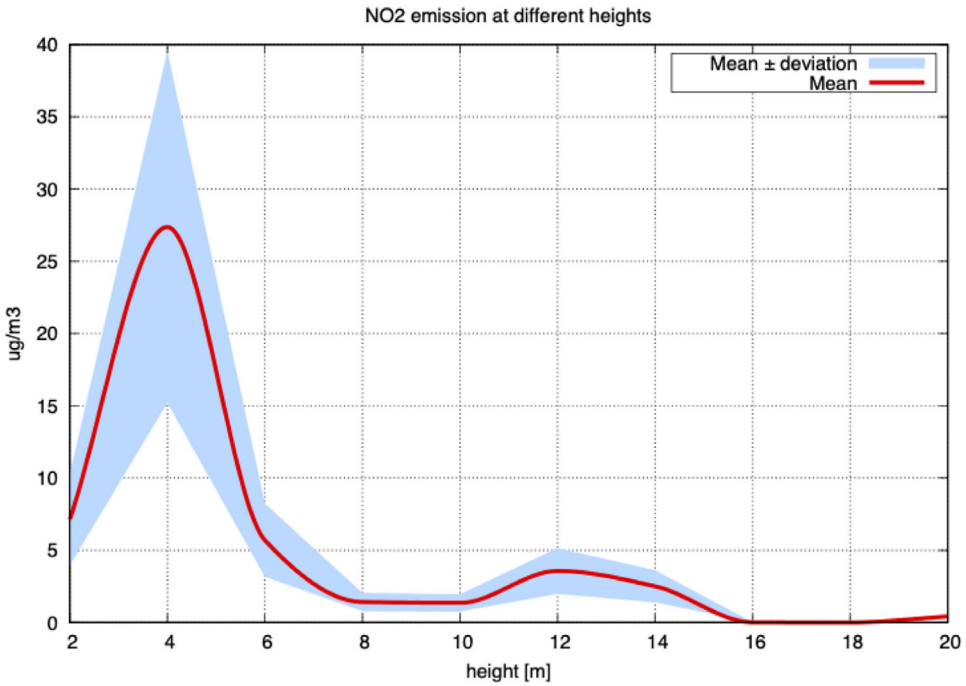


Figure 13. Emissions of NO_2 at different heights above street-level from road transportation, with the mean (red line) and standard deviation (blue region) calculated using the EasyVVUQ campaign.

uncertainty in these pre-existing models, without the need for intrusive modifications to the code. EasyVVUQ provides the tools necessary for computational scientists to add state of the art VVUQ algorithms to their simulation workflows without modifying the underlying codebase.

The library is intentionally execution-method agnostic, providing the base VVUQ workflow elements to allow for different execution patterns (such as Pilot Jobs) facilitated by any choice of middleware solutions. The agnosticism to choice of middleware (including using no middleware at all), and restartability of the workflow, provide the flexibility necessary for EasyVVUQ to be applied to many workflows in the HPC domain. For example, the Fusion application above uses the PSNC Pilot Job Manager to manage job execution, whereas the Ocean Circulation and Migration applications rely on FabSim3. Execution of the materials application, meanwhile, is handled manually by the user. Other middleware solutions may be used, such as RADICAL Cybertools,[9] Dask JobQueue,[46] or cloud submission tools.

The encoding and decoding steps of a standard EasyVVUQ script ensure that application-specific information is abstracted from the rest of the VVUQ workflow. This keeps the UQ algorithms in the sampling elements entirely generic. As such, multiple sampling elements may be chained or combined into more complex sampling elements (such as via use of the `MultiSampler` element). Complex encoding may also be achieved through combining multiple encoders into a single `MultiEncoder` element.

This generic approach is intended to accommodate switching to different UQ methods at no development cost to the user, allowing users to easily try out a variety of UQ approaches. It is intended that many more UQ algorithms will be integrated into this framework over time.

Acknowledgements

D.W.W. and R.A.R. authors contributed equally to this work. The authors are grateful to the VECMA consortium, its Scientific Advisory Board, and the VECMAtk alpha users for their constructive discussions and input around this work. The authors acknowledge funding support from the European Union's Horizon 2020 Research and Innovation Programme under grant agreement 800925 (VECMA project, www.vecma.eu), and the UK Consortium on Mesoscale Engineering Sciences (UKCOMES, http://www.ukcomes.org), EPSRC reference EP/L00030X/1. This work was supported by the Netherlands eScience Center. The calculations were performed in part at the Poznan Supercomputing and Networking Center.

Note: In the originally published version, the URL for the VECMA open source toolkit was repeated in paragraph 3 of Section 1. This was corrected on August 10, 2020, removing the duplicate text.

Conflict of Interest

The authors declare no conflict of interest.

Keywords

high-performance computing, multiscale simulations, uncertainty quantification

Received: December 13, 2019
Revised: April 24, 2020
Published online: June 15, 2020

[1] W. L. Oberkampf, C. J. Roy, *Verification and Validation in Scientific Computing*, Cambridge University Press, Cambridge **2010**.
[2] W. L. Oberkampf, S. M. DeLand, B. M. Rutherford, K. V. Diegert, K. F. Alvin, *Reliab. Eng. Syst. Safe.* **2002**, *75*, 333.

[3] R. A. Richardson, D. W. Wright, W. Edeling, V. Jancauskas, J. Lakhlili, P. V. Coveney, *J. Open Res. Softw.* **2020**, *8*, 11.
[4] D. Groen, R. A. Richardson, D. W. Wright, V. Jancauskas, R. Sinclair, P. Karlshoefer, M. Vassaux, H. Arabnejad, T. Piontek, P. Kopta, B. Bosak, J. Lakhlili, O. Hoenen, D. Suleimenova, W. Edeling, D. Crommelin, A. Nikishova, P. V. Coveney, in *Computational Science – ICCS 2019*, Springer International Publishing, New York **2019**, pp. 479–492.
[5] D. Groen, A. P. Bhati, J. Suter, J. Hetherington, S. J. Zasada, P. V. Coveney, *Comput. Phys. Commun.* **2016**, *207*, 375.
[6] T. Piontek, B. Bosak, M. Ciżnicki, P. Grabowski, P. Kopta, M. Kulczewski, D. Szejnfeld, K. Kurowski, *J. Grid Comput.* **2016**, *14*, 559.
[7] B. Bosak, J. Lakhlili, EasyVVUQ-QCGPJ, https://github.com/vecma-project/easyvvuq-qcgpj (accessed: May 2020).
[8] P. Kopta, B. Bosak, QCG-PilotJob, https://github.com/vecma-project/QCG-PilotJob (accessed: May 2020).
[9] V. Balasubramanian, S. Jha, A. Merzky, M. Turilli, *arXiv preprint arXiv:1904.03085*, **2019**.
[10] Cerberus: Lightweight, extensible data validation library for python, https://github.com/pyeve/cerberus (accessed: April 2020).
[11] Supplementary information/code repository for this paper, https://github.com/vecma-project/EasyVVUQApplicationsSupplementary (accessed: December 2019).
[12] M. Eldred, J. Burkardt, in *47th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, American Institute of Aeronautics and Astronautics, Reston, VA **2009**, p. 976.
[13] D. Xiu, *Numerical Methods for Stochastic Computations: A Spectral Method Approach*, Princeton University Press, Princeton, NJ **2010**.
[14] R. Preuss, U. von Toussaint, in *AIP Conf. Proc.*, Vol. 1756, AIP Publishing, New York **2016**, p. 060001.
[15] I. Sobol, *Math. Comput. Simulat.* **2001**, *55*, 271.
[16] G. Tang, G. Iaccarino, M. Eldred, in *51st AIAA/ASME/ASCE/AHS/ASC Structures*, Structural Dynamics, and Materials Conf., Orlando, FL **2010**, pp. 1–13.
[17] B. Sudret, *Reliab. Eng. Syst. Safe.* **2008**, *93*, 964.
[18] M. Vassaux, R. C. Sinclair, R. A. Richardson, J. L. Suter, P. V. Coveney, *Adv. Theory Simul.* **2019**, 1900122.
[19] M. Vassaux, R. Richardson, P. Coveney, *Philos. Trans. R. Soc., A* **2019**, *377*, 20180150.
[20] R. C. Sinclair, Epoxy polymerisation code, https://github.com/velocirobbie/epoxy_polymerisation (accessed: August 2019).
[21] M. Vassaux, R. C. Sinclair, R. A. Richardson, J. L. Suter, P. V. Coveney, *Adv. Theory Simul.* **2019**, *2*, 1800168.
[22] S. Plimpton, *J. Comput. Phys.* **1995**, *117*, 1.
[23] B. M. Boghosian, P. V. Coveney, H. Wang, *Adv. Theory Simul.* **2019**, 1900125.
[24] W. Verkley, P. Kalverla, C. Severijns, *Q. J. R. Meteorol. Soc.* **2016**, *142*, 2273.
[25] R. Peyret, *Spectral Methods for Incompressible Viscous Flow*, Springer Science & Business Media, New York **2013**.
[26] J. Feinberg, H. P. Langtangen, *J. Comput. Sci.* **2015**, *11*, 46.
[27] W. Edeling, D. Groen, FabUQCampaign, https://github.com/wedeling/FabUQCampaign (accessed: May 2020).
[28] G. L. Falchetto, D. Coster, R. Coelho, B. D. Scott, L. Figini, D. Kalupin, E. Nardon, S. Nowak, L. L. Alves, J. F. Artaud, V. Basiuk1, J. P. S. Bizarro, C. Boulbe, A. Dinklage, D. Farina, B. Faugeras, J. Ferreira, A. Figueiredo, Ph. Huynh, F. Imbeaux, I. Ivanova-Stanik, T. Jonsson, H.-J. Klingshirn, C. Konz, A. Kus, N. B. Marushchenko, G. Pereverzev, M. Owsiak, E. Poli, Y. Peysson, et al., *Nucl. Fusion* **2014**, *54*, 043018.
[29] F. Imbeaux, J. Lister, G. Huysmans, W. Zwingmann, M. Airaj, L. Appel, V. Basiuk, D. Coster, L. G. Eriksson, B. Guillerminet, D. Kalupin, C. Konz, G. Manduchi, M. Ottaviani, G. Pereverzev, Y. Peysson, O. Sauter, J. Signoret, P. Strand, *Comput. Phys. Commun.* **2010**, *181*, 987.
[30] O. O. Luk, O. Hoenen, A. Bottino, B. D. Scott, D. P. Coster, *Comput. Phys. Commun.* **2019**, *239*, 126.
[31] J. Lakhlili, O. Hoenen, O. Luk, D. Coster, Multiscale Fusion Workflow, https://github.com/vecma-ipp/MFW (accessed: May 2020).
[32] A. Nikishova, A. G. Hoekstra, *J. Comput. Sci.* **2019**, *35*, 80.
[33] G. M. Venturini, *PhD Thesis*, Universidad Carlos III de Madrid (Spain) **2015**.
[34] J. Lin, *IEEE Trans. Inf. Theory* **1991**, *37*, 145.
[35] UNHCR, Figures at a glance, https://www.unhcr.org/figures-at-a-glance.html (accessed: May 2020).
[36] D. Groen, *Procedia Comput. Sci.* **2016**, *80*, 2251.
[37] D. Suleimenova, D. Bell, D. Groen, *Sci. Rep.* **2017**, *7*, 13377.
[38] F. Chen, H. Kusaka, R. Bornstein, J. Ching, C. Grimmond, S. Grossman-Clarke, T. Loridan, K. Manning, A. Martilli, S. Miao, D. Sailor, F. Salamanca, M. Taha, H. abd Tewari, X. Wang, A. Wyszogrodzki, C. Zhang, *Int. J. Climatol.* **2011**, *31*, 273.
[39] J. M. Prusa, P. K. Smolarkiewicz, A. Wyszogrodzki, *Comput. Fluids* **2008**, *37*, 1193.
[40] P. K. Smolarkiewicz, R. Sharman, in *8th GMU Conf. on Transport and Dispersion Modeling*, George Mason University, Fairfax, VA **2004**.
[41] P. K. Smolarkiewicz, R. Sharman, J. Weil, S. G. Perry, D. Heist, G. Bowker, *J. Comput. Phys.* **2007**, *227*, 633.
[42] A. Wyszogrodzki, S. Miao, F. Chen, *Atmos. Res.* **2012**, *118*, 324.
[43] A. Kumar, R. Jimenez, L. Belalcazar, N. Rojas, *Aerosol Air Qual. Res.* **2015**, *16*, 12.
[44] J. Karlický, P. Huszár, T. Halenka, *Adv. Sci. Res.* **2017**, *227*, 181.
[45] Z. P. Piotrowski, A. Wyszogrodzki, P. K. Smolarkiewicz, *Acta Geophys.* **2011**, *59*, 1294.
[46] M. Rocklin, in Proc. of the 14th Python in Science Conf., Citeseer, **2015**, pp. 130–136.