

SOFTWARE METAPAPER

EasyVVUQ: A Library for Verification, Validation and Uncertainty Quantification in High Performance Computing

Robin A. Richardson¹, David W. Wright¹, Wouter Edeling², Vytautas Jancauskas³, Jalal Lakhili⁴ and Peter V. Coveney¹

¹ Centre for Computational Science, Department of Chemistry, University College London, UK

² Centrum Wiskunde and Informatica, Amsterdam, NL

³ Leibniz Supercomputing Centre, Garching, DE

⁴ Max-Planck Institute for Plasma Physics – Garching, Munich, DE

Corresponding authors: Robin A. Richardson (robin.richardson@ucl.ac.uk); David W. Wright (dave.wright@ucl.ac.uk)

EasyVVUQ is an open source Python library (<https://github.com/UCL-CCS/EasyVVUQ>) designed to facilitate verification, validation and uncertainty quantification (VVUQ) for a wide variety of simulations. The goal of EasyVVUQ is to make it as easy as possible to implement advanced VVUQ techniques for existing application codes or workflows. Our aim is to expose these features in an accessible way for users of scientific software, in particular for simulation codes running on high performance computers.

Keywords: Validation; Verification; Uncertainty Quantification; High Performance Computing; Multiscale
Funding statement: We 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 (UK-COMES, <http://www.ukcomes.org>), EPSRC reference EP/L00030X/1.

(1) Overview

Introduction

An overarching goal of computational modelling is to provide insight into questions that otherwise could only be addressed by costly experimentation, if at all. In order for the results of computational science to impact decision making, for example in industrial or clinical settings, it is vital that they are accompanied by a robust understanding of their degree of validity. In practice, this can be decomposed into checks of whether the codes employed are solving the governing equations correctly (*verification*), solving the correct equations to begin with (*validation*), and providing estimates that comprehensively capture uncertainty (*uncertainty quantification*) [11, 12]. These processes, collectively known as VVUQ, provide the basis for determining our level of trust in any given model and the results obtained using it [15].

Recent advances in the scale of computational resources available, and the algorithms designed to exploit them, mean that it is increasingly possible to conduct the additional sampling required by VVUQ even for highly complex calculations and workflows. EasyVVUQ is being developed as part of the the VECMA project (www.vecma.eu), whose goal is to provide an open source toolkit (www.vecma-toolkit.eu) containing a wide range of tools to facilitate the use of VVUQ techniques in multiscale, multiphysics

applications [6]. Our aim is to define stable interfaces and data formats that facilitate VVUQ in the widest range of applications. This would then provide the platform to support complex multi-solver workflows. Several software packages or libraries are already available for performing VVUQ (such as OpenTurns [2], UQLab [8], Uncertainpy [19], Chaospy [4], SALib [7], URANIE [5], UQTK [3], etc.), but in many cases these rely on closed source components and none of them provide the separation of concerns needed to allow the analysis of both small local computations and highly compute intensive kernels (potentially using many thousands of cores and GPUs on HPC or cloud resources). Consequently, the design of EasyVVUQ is focused on making a wide range of VVUQ techniques available for scientists employing unmodified versions of existing applications. In particular, key considerations for us are the ability to support HPC codes, large job counts, and the robustness and restartability of workflows. Nonetheless, we have no intention of reinventing the wheel and reuse existing tools where appropriate to provide robust and optimized code for sampling and analysis.

Implementation and architecture

EasyVVUQ aims to decouple the implementation of VVUQ algorithms from the simulation codes to which they will be applied. In this section we describe the concepts used

to achieve this and how they are translated into code. We make use of the idea of computational *patterns*, which in this context are defined as “abstractions that describe, in a non-application and non-domain specific manner, a workflow or algorithm for conducting validation, verification, uncertainty quantification or sensitivity analysis”. Making use of such patterns in practice requires that they are decomposed into components which can be flexibly combined to implement a range of algorithms. We call these components *Elements* and distinguish two classes: those which implement generic VVUQ functionality and those which translate between the requirements of the VVUQ algorithm and the input and output formats of any given application. **Figure 1** illustrates the decomposition of a generalised VVUQ workflow into different steps which are encapsulated by EasyVVUQ elements. EasyVVUQ is designed around a breakdown of such workflows into four distinct stages; **Sampling**, **Model Evaluation**, **Collation**, and **Analysis**. In an HPC context the model evaluation step is generally equivalent to the execution of a computationally expensive simulation. The actual simulation execution is beyond the remit of the package but EasyVVUQ is designed to wrap around simulation execution, providing functions to generate input (an **Encoder**) and to transform simulation output into common formats for analysis (a **Decoder**). In this section we describe in greater detail how each of these Elements is conceived and implemented.

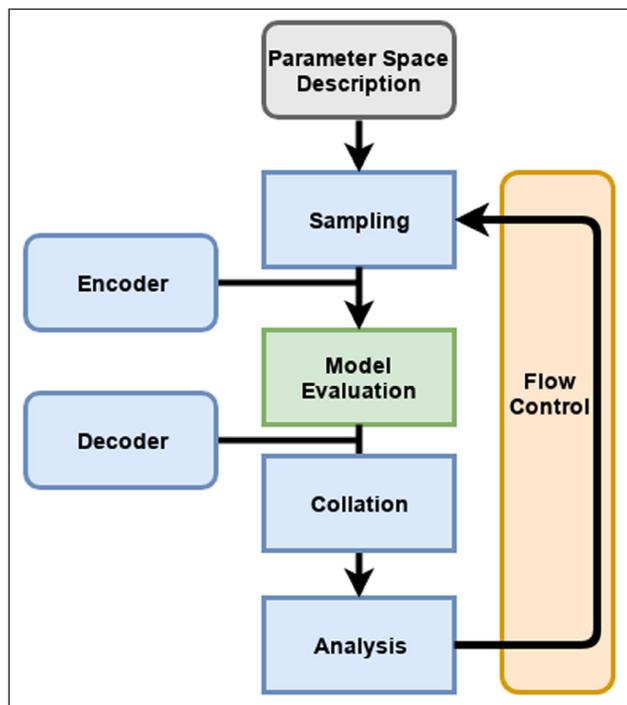


Figure 1: Decomposition of a generalised VVUQ workflow into different steps. These steps are implemented as VVUQ *Elements* in EasyVVUQ. ~ Boxes with straight corners represent generic ~ *Elements* which are defined by the VVUQ workflow. Boxes with rounded corners are specified by users to tailor general workflows to a given use case.

Parameter Description

The first step in our generalised workflow is a description of the model parameters and how they might vary in the sampling phase of the VVUQ pattern. Typically, the user will specify all numerical parameters, the distributions from which they should be drawn and physically acceptable limits on their values.

Campaign

EasyVVUQ workflows are coordinated by an object called a “Campaign”. This contains a common database, the “CampaignDB”, which contains information on the application(s) being analysed alongside the runs mandated by the sampling algorithm(s) employed. The “Campaign” handles all validation and transfers information between each stage of the workflow.

The run information stored in the “CampaignDB” includes a status flag which indicates where in the VVUQ workflow the run is. The recorded steps are when: (1) a set of parameters for a run are added to the database (NEW), (2) simulation inputs are generated (ENCODED) and (3) simulation output is successfully read and prepared for analysis (COLLATED).

Samplers

A “Sampler” populates the “CampaignDB” with a set of run specifications based on the parameter description provided by the user. Each “Sampler” is designed to employ one of a range of algorithms, such as the Monte Carlo or Quasi Monte Carlo approaches [16, 18]. They deal with generic information in the sense that all parameters use the nomenclature and units provided by the user rather than anything specific to any application or workflow.

Encoders

The role of an “Encoder” is to convert generic parameter descriptions into inputs (for example configuration files) which can be used in a specific application. Included in the base application is a simple templating system in which values are substituted into a text input file. For many applications it is envisioned that specific encoders will be needed and the framework of EasyVVUQ means that any class derived from a generic Encoder base class is picked up and may be used. This enables EasyVVUQ to be easily extended for new applications by experienced users.

Decoders

The role of a “Decoder” is twofold, to record simulation completion in the “CampaignDB” and to extract the output information from the simulation runs. Similarly to an “Encoder”, a “Decoder” is designed to be user extendable to facilitate analysis of a wide range of applications.

Collation

“Collation” elements gather “Decoder” output across multiple runs to provide a combined and generic expression of the simulation results for further analysis (for example, the default is to bring together output from all simulation runs in a `pandas` dataframe).

Analysis

The final goal of any VVUQ workflow is an analysis which provides information on the simulation output across a range of runs. Different types of analysis (for example bootstrapping of multiple runs from varied initial conditions) are, or will be, provided by EasyVVUQ.

Dataflow

Using these concepts, we may construct the generalized VVUQ workflow depicted in **Figure 1** in terms of VVUQ elements, as shown in **Figure 2**. A database is populated with runs generated by the chosen sampling element. These generic run descriptions are *encoded* to their application specific input formats and executed. A collation element aggregates the output from finished runs, using the appropriate decoder to extract the desired information. Finally one or more analyses can be carried out on this aggregated data. More runs can subsequently be drawn from the sampler, executed, and collated into the same output for analysis.

The workflow is orchestrated via a Campaign object that stores information such as run descriptions, current status of a run in the workflow (NEW, ENCODED or COLLATED), and the collated output in its database, which we refer to as the CampaignDB. A diagram of the database structure is shown in **Figure 3**. Each run stored in the run table retains the id of the campaign to which it belongs, the application it is generated for, and the sampler which generated it. Runs may also be filtered by their currently recorded status. The collated data is also stored in the database, although this is not shown in **Figure 3** as the table fields depend on the output being collated.

The ultimate goal for EasyVVUQ to support complex multiscale workflows has shaped the design of the Campaign database, which must facilitate storing information from multiple applications. In EasyVVUQ an “app” is defined as a set of parameters (and fixtures which are our term for data sources where the paths may need to be manipulated by Encoders), an Encoder and a Decoder.

Installation

EasyVVUQ is available on the Python Package Index (PyPI), and can be installed using

```
pip install easyvvuq
```

for python versions 3.6 and above, with documentation and tutorials provided at <https://easyvvuq.readthedocs.io>. The latest development version can be obtained directly from the git repository at github.com/UCL-CCS/EasyVVUQ.

Features

The EasyVVUQ library is designed to be easily extended but already implements a variety of UQ algorithms (frequently building upon those found in the `chaospy` library [4]). At present, Stochastic Collocation, Polynomial Chaos Expansion, Quasi Monte Carlo, and parameter sweeps are implemented as sampling elements, some with corresponding analysis elements. It is envisioned that novel algorithms will be designed and implemented within the VECMA project and these integrated with the library, for example semi-intrusive methods for multiscale applications [10].

Data processing and storage within EasyVVUQ is handled using well established libraries. `pandas` dataframes are used as a ‘standard’ container for collated data, allowing simpler interfaces to be designed for the Analysis *elements*. The package includes an integrated wrapper around

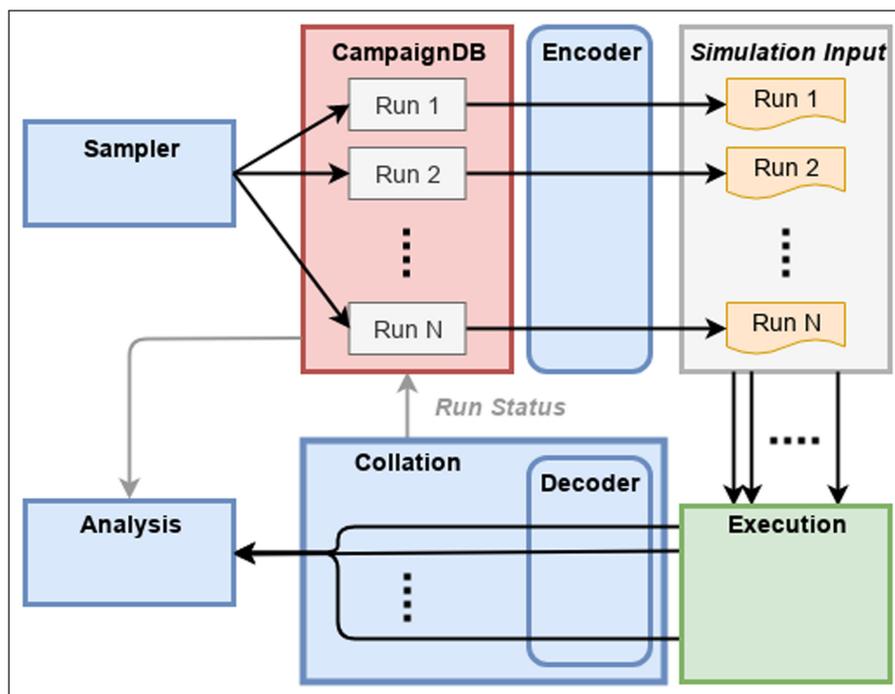


Figure 2: A simple EasyVVUQ workflow for one sampler and one application (simulation code), in terms of VVUQ elements (shown in blue).

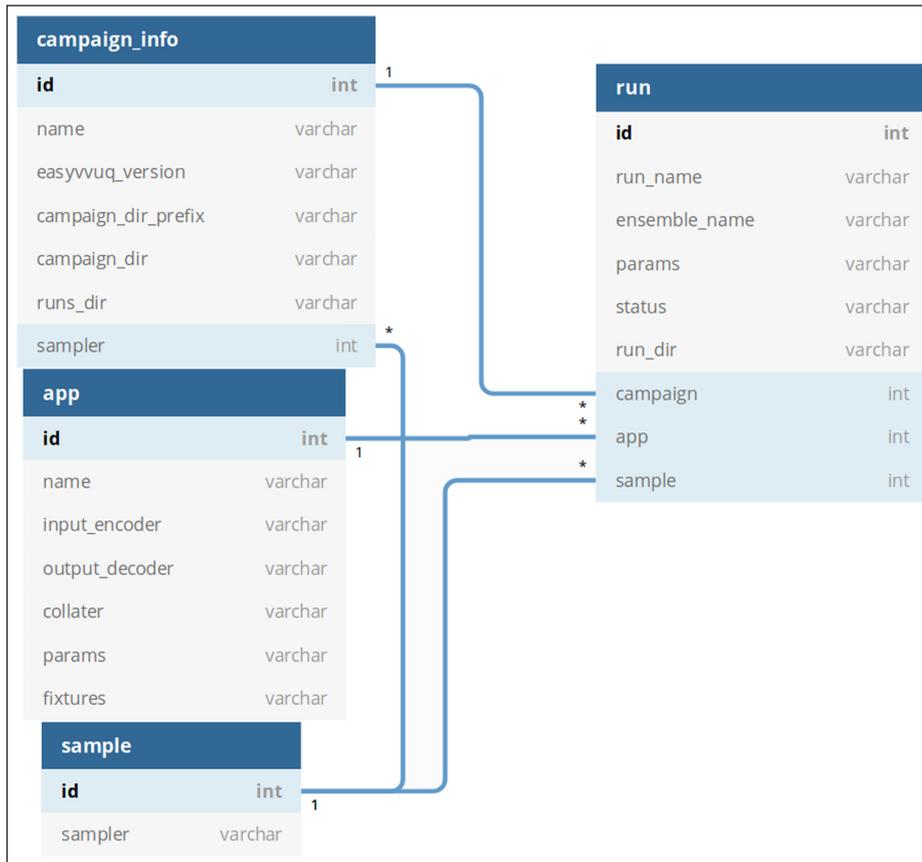


Figure 3: The structure of the Campaign database.

pandas summary statistics, alongside bootstrap statistics and sampler coupled analysis functions for Stochastic Collocation and Polynomial Chaos Expansion workflows. Interaction with databases is via `sqlalchemy` [9] (www.sqlalchemy.org), which provides a choice of several database backends. The use of flexible technology means that, depending on the application and resource on which it is to be run, different database choices will provide the needed performance, scalability and availability.

One of the major motivations behind employing a database is to allow restarting of the workflow—an important consideration for HPC workflows involving a large number of runs, for which the cumulative computational cost may become very large. Entire Campaigns can be restarted from the contents of the database, in which VVUQ *elements* such as samplers serialize their state. At present this serialized state is a JSON format string. For more complex objects this is achieved using the `jsonpickle` (github.com/jsonpickle/jsonpickle) library.

Physical range and type checking is performed on all parameters using the `cerberus` python library (github.com/pyeve/cerberus).

Quality control

Testing is carried out with `pytest` (github.com/pytest-dev/pytest), using Travis (travis-ci.org/) for Continuous Integration. The current test suite consists mostly of high level “integration” style tests where entire workflows are tested. These are available for most of the software components, such as samplers, encoders, decoders, etc.

Unit testing is currently being implemented too, with the more complex classes and methods. For example, the database and more elaborate sampling techniques. There will be alpha testing from the project community for the duration of the VECMA project.

Example application: Cooling coffee cup

In this section, we illustrate the intended EasyVVUQ v0.5 [14] workflow using the following basic example script, a python implementation of the cooling coffee cup model used in the *uncertainpy* documentation (code for which is in the `tests/cooling/subdirectory` of the EasyVVUQ distribution directory). The code takes a small key/value pair input and outputs a comma separated value (CSV) file. The model uses Newton’s law of cooling to evolve the temperature, T , over time (t) in an environment at T_{env} :

$$\frac{dT(t)}{dt} = -k(T(t) - T_{env}) \quad (1)$$

The constant κ characterizes the rate at which the coffee cup transfers heat to the environment. In this example we will analyze this model using the polynomial chaos expansion (PCE) UQ algorithm. We will use a constant initial temperature $T_0 = 95^\circ\text{C}$, and vary κ and T_{env} , sampling them from a uniform distribution in the ranges 0.025–0.075 and 15–25 respectively.

Below we provide a commented script that shows how the Campaign is built up and then employed. We also provide an outline of how each element is set up:

```

import easyvvuq as uq
import chaospy as cp

# Set up a fresh campaign called "coffee_pce"
my_campaign = uq.Campaign(name='coffee_pce')

# Define parameter space
params = {
    "temp_init": {"type": "float", "min": 0.0, "max": 100.0, "default": 95.0},
    "kappa": {"type": "float", "min": 0.0, "max": 0.1, "default": 0.025},
    "t_env": {"type": "float", "min": 0.0, "max": 40.0, "default": 15.0},
    "out_file": {"type": "string", "default": "output.csv"}
}

# Create an encoder, decoder and collater for PCE test app
encoder = uq.encoders.GenericEncoder(
    template_fname='cooling.template',
    delimiter='$',
    target_filename='cooling_in.json')

decoder = uq.decoders.SimpleCSV(target_filename="output.csv",
                                output_columns=["te"],
                                header=0)

collater = uq.collate.AggregateSamples(average=False)

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

# Create the sampler
vary = {
    "kappa": cp.Uniform(0.025, 0.075),
    "t_env": cp.Uniform(15, 25)
}

my_sampler = uq.sampling.PCESampler(vary=vary, polynomial_order=3)

# Associate the sampler with the campaign
my_campaign.set_sampler(my_sampler)

# Will draw all (of the finite set of samples)
my_campaign.draw_samples()

# Encode and execute all runs
my_campaign.populate_runs_dir()
my_campaign.apply_for_each_run_dir(uq.actions.ExecuteLocal(
    "cooling_model.py cooling_in.json"))

# Aggregate decoded output for all runs
my_campaign.collate()

# Post-processing analysis
my_analysis = uq.analysis.PCEAnalysis(sampler=my_sampler, qoi_cols=["te"])
my_campaign.apply_analysis(my_analysis)

# Get Descriptive Statistics
results = my_campaign.get_last_analysis()
stats = results['statistical_moments']['te']
per = results['percentiles']['te']
sobols = results['sobols_first']['te']

```

In the above, the output of the `cooling_model` is the temperature, 'te' (T at the end of the simulation).

Parameter space definition

The parameter space is defined using a dictionary. Each entry in the dictionary follows the format:

```

"parameter_name": {
    "type":      "<datatype>",
    "min":      <value>,
    "max":      <value>,
    "default":  <value>
}

```

with a defined type, minimum and maximum value and default. If the parameter is not selected to vary in the Sampler (see below) then the default value is used for every run.

App creation

In this example the `GenericEncoder` and `SimpleCSV`, both included in the core EasyVVUQ library, were used as the encoder/decoder pair for this application. `GenericEncoder` performs simple text substitution into a supplied template, using a specified delimiter to identify where parameters should be placed. The template is shown below (is used as the delimiter). The template substitution approach is likely to suit most simple applications but in practice many large applications have more complex requirements, for example the multiple input files or the creation of a directory hierarchy. In such cases, users may write their own encoders by extending the `BaseEncoder` class.

```
{
  "T0": "temp_init",
  "kappa": "$kappa",
  "t_env": "$t_env",
  "out_file": "$out_file"
}
```

As can be inferred from its name, SimpleCSV reads CSV files produced by the cooling model code. Again, many applications output results in different formats, potentially requiring bespoke Decoders.

In this workflow all application runs will be analyzed as individual data points, so we set the collator to `AggregateSamples` without averaging. This element simply extracts information using the assigned decoder and adds it to a summary dataframe.

The sampler

The user specifies which parameters will vary and their corresponding distributions. In this case the `kappa` and `t_env` parameters are varied, both according to a uniform distribution:

```
vary = {
  "kappa": cp.Uniform(0.025, 0.075),
  "t_env": cp.Uniform(15, 25)
}
```

Once created, the sampler is associated to the campaign object using the `set_sampler()` method. Calling the campaign's `draw_samples()` method will cause the specified number of samples to be added as runs to the campaign database, awaiting encoding and execution. If no arguments are passed to `draw_samples()` then all samples will be drawn, unless the sampler is not finite.

Execute runs

Due to the diverse range of execution patterns required by large HPC workflows, EasyVVUQ does not intend to handle matters of execution, leaving this instead to the user's chosen middleware approach. However, some very basic methods are provided to aid in local execution for testing purposes. The `populate_runs_dir()` method of `my_campaign` will create a directory hierarchy containing the

encoded input files for every run that has not yet been completed. Finally, in this example, a shell command is executed in each directory to execute the simple test code. In practice this stage would be best handled using, for example, a pilot job manager.

Collation and analysis

Calling `my_campaign.collate()` at any stage causes the campaign to aggregate decoded simulation output for all runs which have not yet been collated. This collated data is stored in the campaign database. An analysis element can then be applied to the campaign's collation result. The output of this analysis is dependent on the type of analysis element employed.

In the example application described above, we apply the `PCEAnalysis` element – the logical counterpart to the `PCESampler` used earlier to generate the samples. The output of this analysis ('results' in the above example script) is a python dict containing the statistical moments (mean, standard deviation etc.) of the Quantity of Interest – in this case, the Temperature, T ('te' in the script) – and the associated percentiles of the distribution. The results dict also contains the Sobol indices [17], which measure the sensitivity of the model output (T) to each of the input parameters being varied. Note that, in the script above, this final temperature is referred to as 'te'. In this case the PCE is being carried out over two inputs, κ and T_{env} , so the 'sobols_first' dict entry contains two values.

Here, the only model output was T , so the analysis element is instructed to operate only on this one output. In general, however, a model can have multiple outputs. In such a case the `PCEAnalysis` output dict would contain the relevant statistics for each of the outputs, obtainable through using a variable name other than ["te"] as index.

For illustration purposes, the statistical moments (mean, standard deviation and 90% prediction interval) and the first-order Sobol indices for the sensitivity analysis (SA) of the Cooling Coffee Cup model have been plotted in **Figure 4**. As expected, the mean temperature

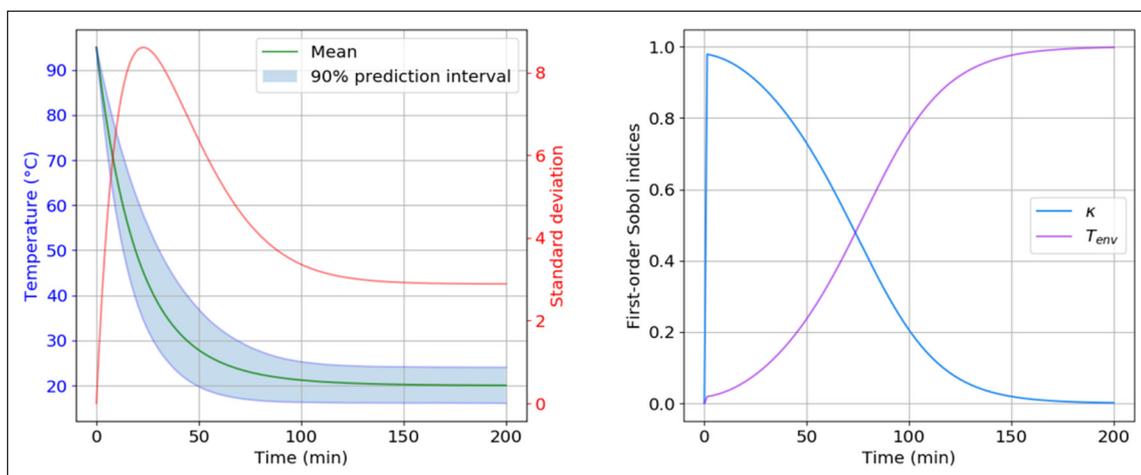


Figure 4: Descriptive statistics and sensitivity analysis of the cooling coffee cup: on the left we have the mean, variance, and 90% prediction interval of the cup temperature T , and on the right we have the first order Sobol indices for each of the uncertain parameters, κ and T_{env} .

T is decreasing exponentially towards the environment temperature T_{env} . For the SA as can be observed (when the first Sobol index is close to 1), T is more affected at the beginning of the simulation by the uncertainty in the rate κ , while it is exclusively affected at the end by the uncertainty in T_{env} .

(2) Availability

Operating system

Modern Linux (or OSX) with appropriate python version.

Programming language

Python 3.6

Additional system requirements

Memory and disk space dependent on usage case.

Dependencies

Tested with the following:

- numpy 1.16.2
- pandas 0.25
- scipy 1.3.1
- chaospy 3.0.17
- SALib 1.3.8
- pytest 4.3.1
- pytest-pep8 1.0.6
- SQLAlchemy 1.3.8
- sqlalchemy-utils 0.34.2
- jsonpickle 1.2
- cerberus 1.3.1

List of contributors

In addition to the paper authors, we wish in particular to acknowledge contributions from the following people:

Bartosz Bosak (Poznań Supercomputing and Networking Center, Poznań, Poland) for discussions on middleware and workflow integration.

Derek Groen (Department of Computer Science, Brunel University London, London, UK), as VECMA project Technical Manager.

Consult the CONTRIBUTIONS.md file in the code repository for a more complete listing of contributions.

Software location

Name: EasyVVUQ v0.5

Persistent identifier: <https://doi.org/10.5281/zenodo.3722092>

Licence: LGPLv3

Version published: v0.5

Date published: 13/12/2019

Code repository

Name: EasyVVUQ

Persistent identifier: <https://github.com/UCL-CCS/EasyVVUQ>

Licence: LGPLv3

Date published: 14/12/2018 (date of v0.1 release)

Language

English

(3) Reuse potential

EasyVVUQ provides the tools to enable 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. Within the VECMA project, workflows have been created which employ PSNC PilotJob Manager, FabSim, RADICAL Cybertools (see www.vecma-toolkit.eu) and Dask, but users are free to use other solutions (for example Taverna or cloud submission tools).

The EasyVVUQ library is intended to provide a platform to design and evaluate novel VVUQ algorithms and to be user extensible. As such, more sampling and analysis options will gradually be made available within the library, alongside a wider range of application specific wrappers. The library also provides the foundation for tools that will help understand the propagation of uncertainty through complex workflows (such as those underpinning multiscale simulations).

Through facilitating the use of rigorous VVUQ procedures in computational science we hope EasyVVUQ will aid the statistical response to the “reproducibility crisis” in science [1, 13].

Acknowledgements

We are grateful to the VECMA consortium, Scientific Advisory Board and the VECMAtk alpha users for their constructive discussions and input around this work. This work was additionally supported by the Netherlands eScience Center.

Competing Interests

The authors have no competing interests to declare.

References

1. **Baker, M** 2016 1,500 scientists lift the lid on reproducibility. *Nature*, 533: 452–4. DOI: <https://doi.org/10.1038/533452a>
2. **Baudin, M, Dutfoy, A, Iooss, B and Popelin, A-L** 2017 OpenTURNS: An Industrial Software for Uncertainty Quantification in Simulation. In: *Handbook of Uncertainty Quantification*, 2001–2038. Springer International Publishing. DOI: https://doi.org/10.1007/978-3-319-12385-1_64
3. **Debusschere, B, Sargsyan, K, Safta, C and Chowdhary, K** 2017 Uncertainty Quantification Toolkit (UQTK). In: *Handbook of Uncertainty Quantification*, 1807–1827. Springer International Publishing. DOI: https://doi.org/10.1007/978-3-319-12385-1_56
4. **Feinberg, J and Langtangen, H P** 2015 ‘Chaospy: An open source tool for designing methods of uncertainty quantification’. *Journal of Computational Science*, 11: 46–57. DOI: <https://doi.org/10.1016/j.jocs.2015.08.008>
5. **Gaudier, F** 2010 ‘URANIE: The CEA/DEN Uncertainty and Sensitivity platform’. *Procedia – Social and*

- Behavioral Sciences*, 2(6): 7660–7661. DOI: <https://doi.org/10.1016/j.sbspro.2010.05.166>
6. **Groen, D, Richardson, R A, Wright, D W, Jancauskas, V, Sinclair, R, Karlshoefler, P, Vassaux, M, Arabnejad, H, Piontek, T, Kopta, P, Bosak, B, Lakhilili, J, Hoenen, O, Suleimenova, D, Edeling, W, Crommelin, D, Nikishova, A and Coveney, P V** 2019 Introducing VECMAtk – Verification Validation and Uncertainty Quantification for Multiscale and HPC Simulations. In: *'Lecture Notes in Computer Science'*, 479–492. Springer International Publishing. DOI: https://doi.org/10.1007/978-3-030-22747-0_36
 7. **Herman, J and Usher, W** 2017 'SALib: An open-source Python library for Sensitivity Analysis'. *The Journal of Open Source Software*, 2(9): 97. DOI: <https://doi.org/10.21105/joss.00097>
 8. **Marelli, S and Sudret, B** 2014 UQLab: A Framework for Uncertainty Quantification in Matlab, in 'Vulnerability Uncertainty, and Risk'. American Society of Civil Engineers. DOI: <https://doi.org/10.1061/9780784413609.257>
 9. **Myers, J and Copeland, R** 2015 *Essential SQLAlchemy: Mapping Python to Databases*. O'Reilly Media, Inc.
 10. **Nikishova, A and Hoekstra, A G** 2019 'Semi-intrusive uncertainty propagation for multiscale models'. *Journal of Computational Science*, 35: 80–90. DOI: <https://doi.org/10.1016/j.jocs.2019.06.007>
 11. **Oberkampf, W L, DeLand, S M, Rutherford, B M, Diegert, K V and Alvin, K F** 2002 'Error and uncertainty in modeling and simulation'. *Reliability Engineering & System Safety*, 75(3): 333–357. DOI: [https://doi.org/10.1016/S0951-8320\(01\)00120-X](https://doi.org/10.1016/S0951-8320(01)00120-X)
 12. **Oberkampf, W L and Roy, C J** 2010 *Verification and Validation in Scientific Computing*. Cambridge University Press. DOI: <https://doi.org/10.1017/CBO9780511760396>
 13. **Peng, R** 2015 'The reproducibility crisis in science: A statistical counterattack'. *Significance*, 12(3): 30–32. DOI: <https://doi.org/10.1111/j.1740-9713.2015.00827.x>
 14. **Richardson, R A, Wright, D W, Jancauskas, V, Lakhilili, J and Edeling, W** 2019 'Easyvvuq v0.5'. Documentation at <https://easyvvuq.readthedocs.io>. DOI: <https://doi.org/10.5281/zenodo.3722092>
 15. **Roy, C J and Oberkampf, W L** 2011 'A comprehensive framework for verification validation, and uncertainty quantification in scientific computing'. *Computer Methods in Applied Mechanics and Engineering*, 200(25–28): 2131–2144. DOI: <https://doi.org/10.1016/j.cma.2011.03.016>
 16. **Rubinstein, R Y and Kroese, D P** 2007 *Simulation and the Monte Carlo Method*. John Wiley & Sons Inc. DOI: <https://doi.org/10.1002/9780470230381>
 17. **Saltelli, A, Ratto, M, Andres, T, Campolongo, F, Cariboni, J, Gatelli, D, Saisana, M and Tarantola, S** 2008 *Global sensitivity analysis: the primer*. John Wiley & Sons. DOI: <https://doi.org/10.1002/9780470725184>
 18. **Sobol, I M** 1998 'On quasi-Monte Carlo integrations'. *Mathematics and Computers in Simulation*, 47(2–5): 103–112. DOI: [https://doi.org/10.1016/S0378-4754\(98\)00096-2](https://doi.org/10.1016/S0378-4754(98)00096-2)
 19. **Tennøe, S, Halnes, G and Einevoll, G T** 2018 Uncertainpy: A Python Toolbox for Uncertainty Quantification and Sensitivity Analysis in Computational Neuroscience. *Frontiers in Neuroinformatics*, 12. DOI: <https://doi.org/10.3389/fninf.2018.00049>

How to cite this article: Richardson, R A, Wright, D W, Edeling, W, Jancauskas, V, Lakhilili, J and Coveney, P V 2020 EasyVVUQ: A Library for Verification, Validation and Uncertainty Quantification in High Performance Computing. *Journal of Open Research Software*, 8: 11. DOI: <https://doi.org/10.5334/jors.303>

Submitted: 27 September 2019

Accepted: 23 March 2020

Published: 29 April 2020

Copyright: © 2020 The Author(s). This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International License (CC-BY 4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited. See <http://creativecommons.org/licenses/by/4.0/>.

 *Journal of Open Research Software* is a peer-reviewed open access journal published by Ubiquity Press.

OPEN ACCESS 