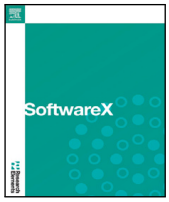
SoftwareX 24 (2023) 101561

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx



Original Software Publication

UQpy v4.1: Uncertainty quantification with Python

Dimitrios Tsapetis [a], Michael D. Shields [a,*], Dimitris G. Giovanis [a], Audrey Olivier [b], Lukas Novak [c], Promit Chakroborty [a], Himanshu Sharma [a], Mohit Chauhan [a], Katiana Kontolati [a], Lohit Vandanapu [a], Dimitrios Loukrezis [d], Michael Gardner [e]

[a] Department of Civil and Systems Engineering, Johns Hopkins University, Baltimore, MD, 21218, USA
[b] Sonny Astani Department of Civil and Environmental Engineering, University of Southern California, Los Angeles, CA, 90089, USA
[c] Faculty of Civil Engineering, Brno University of Technology, Veveri 331/95, Brno 60200, Czech Republic
[d] Department of Electrical Engineering and Information Technology, Technical University of Darmstadt, Darmstadt, Germany
[e] Department of Civil and Environmental Engineering, University of California, Davis, CA, 95616, USA

ARTICLE INFO

Keywords:
Uncertainty quantification
Continuous integration

ABSTRACT

This paper presents the latest improvements introduced in Version 4 of the `UQpy`, Uncertainty Quantification with `Python`, library. In the latest version, the code was restructured to conform with the latest `Python` coding conventions, refactored to simplify previous tightly coupled features, and improve its extensibility and modularity. To improve the robustness of `UQpy`, software engineering best practices were adopted. A new software development workflow significantly improved collaboration between team members, and continuous integration and automated testing ensured the robustness and reliability of software performance. Continuous deployment of `UQpy` allowed its automated packaging and distribution in system agnostic format via multiple channels, while a Docker image enables the use of the toolbox regardless of operating system limitations.

Code metadata

Current code version	v4.1.1
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-23-00337
Code Ocean compute capsule	https://codeocean.com/capsule/9924279/tree
Legal Code License	MIT Licence
Code versioning system used	git
Software code languages, tools, and services used	`Python`, MPI
Compilation requirements, operating environments & dependencies	see `setup.py`
If available Link to developer documentation/manual	https://uqpyproject.readthedocs.io
Support forum for questions	https://github.com/SURGroup/UQpy/discussions

Software metadata

Current software version	4.1.1
Permanent link to executables of this version	pypi.org/project/UQpy github.com/SURGroup/UQpy/releases anaconda.org/conda-forge/uqpy
Legal Software License	MIT License
Computing platforms/Operating Systems	Linux, OS X, Microsoft Windows
Installation requirements & dependencies	see `setup.py`
If available, link to user manual - if formally published include a reference to the publication in the reference list	https://uqpyproject.readthedocs.io

* Corresponding author.
E-mail address: michael.shields@jhu.edu (Michael D. Shields).

Available online 27 October 2023
https://doi.org/10.1016/j.softx.2023.101561

Received 22 May 2023; Received in revised form 20 August 2023; Accepted 12 October 2023

1. Motivation and significance

Uncertainty Quantification (UQ) is the science of characterizing, quantifying, managing, and reducing uncertainties in mathematical, computational and physical systems. Depending on the sources of uncertainty, UQ provides a multitude of methodologies to quantify their effects. For instance, given the probability distribution for the inputs to a computational model, forward uncertainty propagation methods aim to estimate the distributions or statistics of resulting quantities of interest. Inverse UQ, on the other hand, aims to infer uncertainties in input quantities given limitations and uncertainties in the observed system response, e.g. for model calibration from experimental data. Numerous related tasks fall under the broad classification of UQ including sensitivity analysis, which aims to quantify the influence of multiple inputs to a system, and reliability analysis which aims to estimate (and sometimes minimize) the probability of failure of the system.

A major challenge in UQ is to reduce the high computational expense associated with many repeated model evaluations. This can be achieved through advances in sampling, development of computationally inexpensive surrogate models (or metamodels), and by leveraging high performance computing. To address these challenges, multiple software packages and libraries have been developed. Some of the most comprehensive libraries for UQ include `OpenTurns` [1], `Korali` [2], `MUQ` [3], `UQTk` [4], `Dakota` [5], `OpenCossan` [6] and `UQLab` [7]. These software are developed in either `C`, `C++` programming languages or `Matlab` and although many provide bindings to `Python` (to differing extents), they are not generally suitable for direct extension in `Python`, which is one of the most widely used languages in the scientific community.

Apart from these general purpose UQ libraries, several packages that target specific applications or with more limited scope are available. In `R`, the `DiceDesign` [8] package aids experimental design, while `DiceKriging` and `DiceOptim` [9] use Kriging for metamodeling and surrogate-based optimization, respectively. The `Matlab` code `FERUM` [10], developed at UC Berkeley and IFMA, Clermont, serves as a general purpose finite element structural reliability code, while `SUMOToolbox` [11] is a framework for global surrogate modeling and adaptive sampling. The Engineering Risk Analysis group at TU Munich also provides a collection of `Matlab` and `Python` routines [12] related to the group's research. Specifically in `Python`, several focused libraries have been developed. `UncertaintyPy` [13] was developed for UQ in computational neuroscience. `PyROM` framework [14] provides a user-friendly way to implement model reduction techniques. The `ChaosPy` package provides UQ functionality centered around polynomial chaos expansions. Bayesian calibration algorithms are implemented in `SPUX` [15] and `ABCpy` [16] and sensitivity analyses by `SALib` [17]. `PyMC` [18] provides a simple `Python` interface that allows its user to create Bayesian models and fit them using Markov Chain Monte Carlo methods. `PyGPC` [19] library is based on generalized polynomial chaos theory and provides capabilities for uncertainty and sensitivity analysis of computational models. Three of the latest additions are `PyApprox` [20], which provides wide-ranging functionality, `NeuralUQ` [21] focused on UQ in neural network models, and `Fortuna` that provides uncertainty estimates, classification and prediction for production systems.

`UQpy` aims to provide a comprehensive UQ library with wide-ranging capabilities spanning the areas discussed above, as well as a development environment for creating new UQ methodologies. The `UQpy` package was originally introduced in [22], where the overall structure of v3 was described. Since then, the authors have reworked the `UQpy` architecture with the goal to simplify its structure, enhance its extensibility, and make it more robust. The updated architecture of the library rendered it not backwards compatible, as the strategy for construction of classes has changed. Yet porting older solutions to the new structure can be performed in a straightforward manner. This restructuring resulted in the current version we present here, v4.1.

The first task carried out towards v4.1 was to restructure the file system. The previous structure, which maintained a single `Python` file per module, had reached size limitations and made it cumbersome for the team members to add new functionalities or update existing ones. In the reorganization, a directory was created for each module, which contains, in a hierarchical structure, subdirectories for specific functionalities, with one file dedicated to each class. Slight modifications were also made to the existing code to ensure compliance with `PEP8` by renaming modules, classes, and function signatures. Instead of monolithic classes per functionality, each component was split into a separate class with a dedicated `abstract baseclass`, where applicable. This choice reduced code complexity, provided a standardized way of extending components, and enabled the construction of the final functionality, using object composition and inheritance.

The second step to improve internal and external collaboration was to deprecate the "branch-per-developer" strategy and move to a feature-based branch structure using the Github Flow. This removed unnecessary redundancies and complications when multiple people are working on related functionalities. At the same time, the workflow is now directly combined with testing automation and Continuous Integration/Continuous Delivery (CI/CD) workflows. Unit tests were implemented throughout the software, achieving code coverage greater than 80%. The CI pipeline includes linting, code quality checks, and automated semantic versioning, while the CD pipeline packages and distributes the code via multiple channels, such as `PyPI`, `conda-forge`, and `Docker` images. This CI/CD pipelines are explained in more detail in Section 3.

The documentation was revamped to reflect the new hierarchical structure of the code, with embedded examples serving as tutorials to quickly familiarize users with the code functionality. Specifically, for each class, a gallery of examples is created using the `sphinx-gallery` extension [23]. The users can now download the examples in both Jupyter notebook and `Python` format or directly interact with the example in a dedicated Binder environment. Finally, several new functionalities were introduced either by the development team or external collaborations, thus boosting `UQpy`'s capabilities.

2. Software description

2.1. Software architecture

`UQpy` is a `Python`-based toolbox that provides a series of computational methodologies and algorithms for wide-ranging UQ problems. The core of `UQpy` is based on state-of-the-art `Python` libraries, specifically `NumPy` [24], which is the most fundamental package supporting array and linear algebra operations, `SciPy` [25], that provides algorithms for optimization, integration and basic statistics, and `scikit-learn` [26], which includes various tools for supervised and unsupervised learning. `UQpy` is split into eleven modules, nine of which address specific tasks in UQ and which will be discussed in detail in the following section. A module that enables necessary simulations in all other modules, called `run_model`, aids in the batch execution of both `Python` and third-party computational models and includes functionality for parallelization via MPI for high performance computing. Finally, a `utilities` module contains various functions that are common to multiple modules.

2.2. Software modules

In this section, all existing modules of `UQpy` will be briefly introduced, with emphasis on software updates compared to v3. The basic module structure and capabilities are illustrated in Fig. 1. The detailed `UML` diagrams for all modules are included in the `UQpy` documentation allowing architecture visualization.

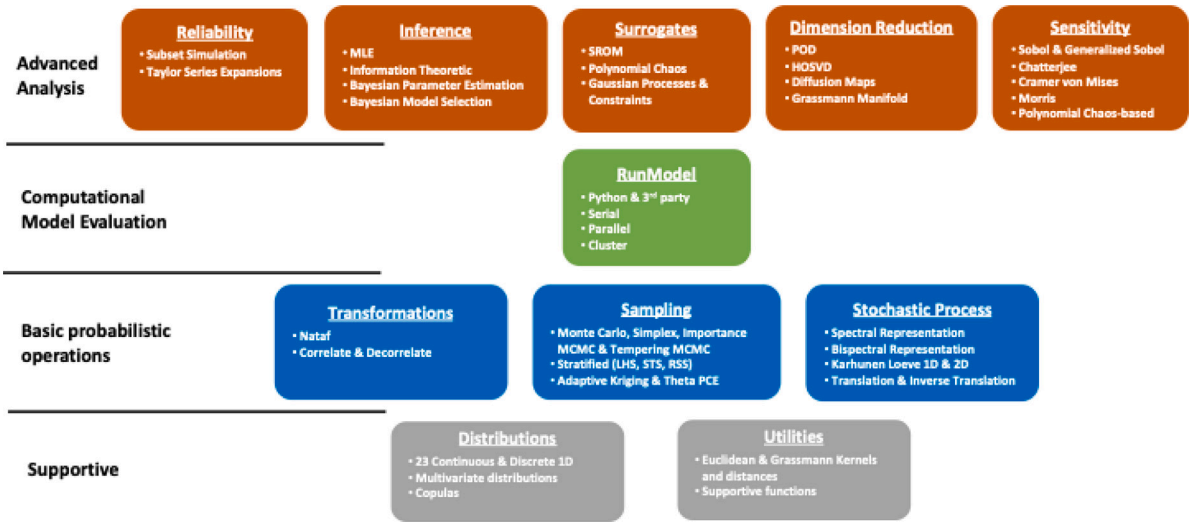


Fig. 1. Structure and capabilities of the UQpy modules.

2.2.1. *distributions module*

The `distributions` module serves as the basis for most probabilistic operations in `UQpy`. It is fully compatible with `scipy` distributions and enables users to create probability distribution objects. Compared to the previous version, the baseclass hierarchy was simplified. An abstract baseclass `Distribution` serves as the interface for creating all subsequent distributions. Depending on the dimensionality of the distribution this baseclass is further refined into `Distribution1D` and `DistributionND` for univariate and multivariate distributions respectively, while the `Distribution1D` is further subclassed into `DistributionsContinuous1D` and `DistributionsDiscrete1D` for continuous and discrete random variables. Within this structure, 23 distinct distributions are implemented. A `Copula` baseclass with two implementations enables users to add dependence between 1D distribution objects. All baseclasses can be easily extended by users to implement any distribution of their choice by simply creating a new child class for the distribution and implementing the requisite methods.

2.2.2. *sampling module*

This module provides a wide range of methods to draw samples of random variables. The following classes enable Monte Carlo simulation and variance reduction methods: `MonteCarloSampling`, `SimplexSampling`, `ImportanceSampling`, and `StratifiedSampling`. The `StratifiedSampling` class has been refactored as a parent class for all stratified sampling approaches with `LatinHypercubeSampling`, `TrueStratifiedSampling`, and `RefinedStratifiedSampling` as child classes, all of which utilize a common `Strata` class for geometric decomposition of the domain. Markov Chain Monte Carlo (MCMC) methods are included, with the `MCMC` abstract baseclass serving as the common interface and 7 different methodologies implemented as subclasses. The latest version includes two new implementations of parallel and sequential tempering MCMC algorithms. Additional MCMC methods can be implemented by the user by simply creating a new subclass with the requisite methods. The module also includes the `AdaptiveKriging` for adaptive sample generation for Gaussian process surrogate modeling (see Section 2.2.9) using specified (and custom) learning functions. Compared to v3, all learning functions have been extracted as separate classes, with a common `LearningFunction` baseclass, allowing users to easily create custom implementations.

2.2.3. *transformations module*

This module contains isoprobabilistic transformations of random variables. Except for updates in naming conventions, this module retained the previous functionality with the `Nataf`, `Correlate`, and `Decorrelate` transformations being available.

2.2.4. *stochastic_process module*

This module supports the simulation of univariate, multivariate, and multidimensional Gaussian and non-Gaussian stochastic processes, with the latest addition since v3 being the two-dimensional Karhunen–Loève Expansion in the `KarhunenLoeveExpansion2D` class. All pre-existing classes of `SpectralRepresentation`, `BispectralRepresentation` and `KarhunenLoeveExpansion` have been updated to conform with PEP8 `Python` coding standards.

2.2.5. *run_model module*

This module is not directly related to any specific UQ operations, yet it is an integral part of the `UQpy` software. It lies at its core and supports the execution of either `Python` or third-party computational models at specified sampling points.

`UQpy` interfaces `Python` models directly, by importing and executing the code. On the other hand, `UQpy` interfaces with third-party software models through ASCII text files to introduce uncertainties in their inputs and uses a standardized scripting format for model execution. In both cases, `UQpy` supports serial and parallel execution. Parallel execution allows the execution of different samples simultaneously, with options for local and cluster execution. Local parallel execution uses MPI and the `mpi4py` library to distribute the random samples among tasks that are processed independently. In this case, the model evaluation cannot invoke MPI internally. In cluster enabled parallelization, with the aid of a bash script, a tiling of the jobs can be performed to include both shared and distributed memory parallelism, while enabling the user to work with different HPC schedulers.

2.2.6. *dimension_reduction module*

In the update from v3 to v4.1, the `dimension_reduction` module was rewritten from scratch. The existing `DirectPOD` and `SnapshotPOD` methods were reworked to comply with the latest `Python` coding conventions and the `HigherOrderSVD` class was added. To support Grassmann manifold projections and operations, a series of classes were added. The `GrassmannProjection` class serves as the parent for classes that project data arrays onto the manifold, with the `SVDprojection` subclass currently available. After the data have been projected, operations such as computing the Karcher mean or

Frechet variance are available with the aid of the GrassmannOperations class. Interpolation can be performed on the manifold with the GrassmannInterpolation class. Special attention was given to the DiffusionMaps class, where the kernel computation was extracted and delegated to a hierarchy of kernel classes in the utilities modules for broader use in future development of other kernel-based methods. More detail can be found in Section 2.2.11.

2.2.7. *inference* module

The functionality of the inference module was retained from v3 to v4.1 but restructured. The previous InferenceModel class, which defines the model on which inference is performed, has now been split into three separate classes depending on the specific model type, namely DistributionModel, LogLikelihoodModel and ComputationalModel, all under the revised InferenceModel baseclass. For information theoretic-based model selection using the InformationModelSelection class, the information criteria have been extracted as separate classes, AIC, BIC, AICc, under a new common InformationCriterion baseclass. The remaining functionality of MLE, BayesParameterEstimation, and BayesModelSelection was updated according to the newly adopted coding conventions and, for Bayesian evidence computation, the EvidenceMethod baseclass has been established with the HarmonicMean subclass defined and allowing straightforward implementation of new Bayesian evidence methods as distinct subclasses.

2.2.8. *reliability* module

Modifications to the reliability module were made to ensure compliance with the latest Python coding guidelines. The first and second-order reliability methods, FORM and SORM, were restructured as subclasses under a common TaylorSeries baseclass to remove code redundancies. The existing SubsetSimulation class was retained and revised to match best practices.

2.2.9. *surrogates* module

One of the most heavily refactored modules in the latest version is surrogates. Generally, surrogate models are now developed under the abstract Surrogate baseclass. The previously existing Kriging class was removed entirely and is now replaced with the more general GaussianProcessRegression, which includes the functionality to perform regression or interpolation (Kriging). Kernels are extracted as separate classes, with the abstract baseclass Kernel (from the utilities module), serving as an interface. The RBF and Matern kernels have been implemented. For use with GaussianProcessRegression, multiple regression methods are implemented as subclasses under the Regression baseclass. The newest addition to GaussianProcessRegression is the ability to add constraints using the virtual point method. These constraints are implemented under the Constraints baseclass, which makes adding new constraints straightforward by implementing a new subclass with the requisite methods.

The PolynomialChaosExpansion class was rewritten from scratch (now as a subclass of Surrogate) to resolve performance issues. Two new baseclasses were introduced. The Polynomials baseclass defines sets of orthogonal polynomials as subclasses, including the Hermite and Legendre polynomial classes. The PolynomialBasis baseclass establishes a set of subclasses to define the polynomial basis, e.g. using a classical tensor product basis TensorProductBasis, or introducing new ways to reduce the basis computation such as the TotalDegreeBasis and HyperbolicBasis classes. This makes the code easily extensible to include new means of basis construction. All regression methods were united as subclasses under the Regression baseclass, again making it more easily extended for new methods, and the computationally efficient LeastAngleRegression was added.

Lastly, the SROM method was retained and updated to conform with the latest Python software development practices.

2.2.10. *sensitivity* module

In v3, the sensitivity module only contained the MorrisSensitivity method. This module significantly benefited from the extensibility introduced in UQpy with v4.1. The Sensitivity abstract baseclass now contains the first major contribution from external collaborators introduced in a set of subclasses that include SobolSensitivity, GeneralizedSobolSensitivity, ChatterjeeSensitivity, and CramerVonMisesSensitivity. Additionally, the updated polynomial chaos expansion code in the surrogates module (see Section 2.2.9), allows the computation of first and total order sensitivity indices with reduced computational cost through the PceSensitivity class, which takes advantage of a fitted PolynomialChaosExpansion object.

2.2.11. *utilities* module

The new utilities module contains code that may be used in multiple modules. This currently contains two abstract baseclasses, the Kernel baseclass and the Distance baseclass for computing kernels and measures of distance, respectively. Within each baseclass, there are two additional baseclasses for Euclidean and Grassmannian kernels/distances. Several kernels and distances have been added as subclasses and new ones can be easily developed by writing a new subclass with the requisite methods.

3. Continuous integration

UQpy v3 [22] was developed using the flexible standards of an academic software, which challenged the ability of the team to collaborate and develop new features using a streamlined workflow. To this end, the latest version was fully restructured to enhance its extensibility, while modern software development practices were introduced to support collaboration and ensure code robustness and quality. The standard of Github Flow was adopted as development strategy. The master branch of the Github repository always contains the latest stable version. A Development branch is now used for merging all newly developed functionality and bug fixes. New versions of the software are released when a pull request is merged from the Development branch to master. For developing new features, a feature-{functionality} branch is created from the latest Development state and merged back once complete. The case is similar for bug fixes, with branches following the bugfix-{bug} naming convention. The aforementioned workflow enables a consistent way of treating new functionality or addressing errors arising during development.

To ensure the code quality of all previously implemented features, the development team enforced unit testing practices. Since the functionality implemented in UQpy is inherently stochastic, and its randomness stems from random number generators, a process of setting the seed to ensure test reproducibility is adopted. All previous functionalities are tested against benchmark problems to achieve a minimum of 80% line coverage. To ensure that the code coverage directive is enforced, Azure Pipelines were used to automatically run all tests and compute coverage when a commit is pushed to the Github repository. The static code analyzer Pylint is also used to enforce coding standards and ensure that no syntax errors are allowed. In addition to these checks, a code quality tool named SonarCloud is used to eliminate code vulnerabilities. This tool is triggered when creating a pull request and automatically detects any code smells, bugs or code duplications introduced, and fails when exceeding a predefined threshold. For a pull request to be acceptable, all test, linting, and code quality must satisfy minimum acceptance criteria and must pass a detailed code review from the code owners. Only then will the additions be merged to Development and subsequently to master branch.

Apart from the Continuous Integration process mentioned above, that ensure the robustness of UQpy, a set of Continuous Deployment

(CD) actions are triggered. The first action is to evoke the `GitVersion` tool, which traverses the Git history of the code and determines the version of the code automatically, as a sequence of numbers `v{Major}.{Minor}.{Patch}`. Using the computed version, the code is packaged and automatically distributed to `Python` Packaging Index (`PyPI`), Github release inside the repository, as well as a `Docker` image that contains the latest `UQpy` version.

Finally, a structured logging framework was established – in lieu of `print` commands triggered by `if` statements that were previously used to indicate errors or faults – that allows users to select the required level of severity tracked during code execution. Six different levels of severity are available in `Python`, namely `NOTSET`, `DEBUG`, `INFO`, `WARN`, `ERROR`, and `CRITICAL`, with the `ERROR` being the default case in `UQpy`. The users can choose a more verbose setting by opting for the `INFO` severity level. Logging output is then directed to their sinks of choice e.g. Terminal, Logfile, Http Streams, etc.

4. Impact

The latest version of `UQpy` modernizes the software to meet best practices in scientific software development, while also updating and improving functionality. This makes the package easier to use and more robust, broadens the classes of problems that it can solve, and greatly enhances the development experience. These points are critical to the widespread adoption of UQ in scientific applications. This robust yet friendly `Python` library is both user- and developer-friendly and provides core functionality to casual users, state-of-the-art methods for advanced users, and a carefully designed environment for developers of UQ methods. With the advent of version 4, we have seen the user-base increase as the library has been adopted by external UQ teams, and have now successfully integrated updates from third-party developers — both of which serve to advance the field of UQ.

To summarize, the entire package has been restructured from a single-file per area to a module hierarchy. Wherever possible, suboptions inside algorithms were extracted using the Strategy design pattern to enhance encapsulation and allow users to select their functionality in a more clear and straightforward manner. Baseclasses are now used throughout the code, which provides interfaces for the implementation of new algorithmic alternatives. To enhance the team collaboration efforts, the already existing version control and Github repository were supported with a CI/CD pipeline that automates software testing and code quality checks to ensure the best scientific output, while each new merge to the master is followed by package releases to `PyPI`, `conda-forge`, and Dockerhub image repository.

Compared to the other existing UQ packages, many of which have been listed above, the aim of `UQpy` is twofold. First of all, we aim to provide an extensive *fully Python-based* UQ library that addresses the wide-ranging needs of the scientific community. At the same time, we want to provide a toolbox that allows its straightforward extension with new functionalities and its use in real-world UQ applications. The developments outlined here represent significant advancements toward these two objectives.

5. Conclusions

In this work, the open-source library for uncertainty quantification `UQpy` and specifically the latest v4.1 was introduced. All changes and updates to the modules of the library were explained in detail, with one of the most significant being the new software development and continuous integration workflow. The latest version enables users and external collaborators to expedite the development of new features using `UQpy` as a platform. This is proven by the new functionalities introduced from both the development team, as well as external collaborators.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Michael Shields reports financial support was provided by US Department of Energy. Michael Shields reports financial support was provided by Defense Threat Reduction agency. Michael Shields reports financial support was provided by National Science Foundation.

Data availability

No data was used for the research described in the article.

Acknowledgments

The authors gratefully acknowledge the following individuals who contributed to code development in `UQpy`: Ulrich Römer, TU Braunschweig; Julius Schultz, TU Braunschweig; Prateek Bhustali, TU Braunschweig. The authors of this work have been supported by the following sources: U.S. Dept. of Energy Award Number DE-SC0020428 (DT, MDS, DG, KK) and Defense Threat Reduction Agency, United States Award Number DTRA1-20-2-0001 (DT, MDS, DG, HS); INL Laboratory Directed Research & Development (LDRD) Program under DOE Idaho Operations Office Contract DE-AC07-05ID14517 (PC); National Science Foundation, United States under award numbers 1652044 (LV) and 1930389 (MC). The content of the information does not necessarily reflect the position or the policy of the federal government, and no official endorsement should be inferred.

Appendix. UQpy future extensions

`UQpy` is continuously updated with new capabilities developed as a product of the research conducted in the group and through external contributions. When possible, existing capabilities are also made more robust and computationally efficient to address the needs of computationally intensive research objectives in UQ. The following describes tentative plans for future development in `UQpy`.

Existing modules are planned to be expanded with the following capabilities:

- `Dimension Reduction`: We are looking to enrich the module with advanced manifold projection methods including kernel PCA [27], principal geodesic analysis (PGA) [28], and Isomaps [29] among others, while also adding new Euclidean/Grassmannian kernels [30–32].
- `Sampling`: We aim to expand the StratifiedSampling class with existing generalized of stratified sampling methods developed in the group such as: Partially Stratified Sampling (PSS) [33], Latinized Stratified Sampling (LSS) [33], and Hierarchical Latin Hypercube Sampling (HLHS) [34].
- `Surrogates`: We plan to enrich the Gaussian Processes with additional constraints such as monotonicity, convexity and differential constraints [35].
- `Surrogates`: We plan to incorporate the novel physics constrained Polynomial Chaos Expansions developed in the group [36,37].
- `Surrogates`: We plan to add Geometric Harmonics [38] for out-of-sample extensions on a manifold.
- `Sensitivity`: We are currently building the capability for Gaussian Process-based Sobol indices estimates [39].

In addition to these extension we envision the development of new modules that will further enhance the capabilities of `UQpy` as a general purpose uncertainty quantification library.

- `Multi-fidelity Modeling`: This module will include multi-fidelity modeling algorithms that include those based on control variates [40] as well as those using Gaussian process corrections and model selection – including methods developed in the group [41–43].
- `Neural Networks`: Algorithms for Bayesian Neural Networks [44], physics-informed Neural Networks [45] and Deep Operator Networks [46] are under development within the group and are planned to be incorporated in the future versions of `UQpy` – likely by coupling with other open-source `Python` libraries that contain computationally efficient Neural Network implementations [26,47,48].
- `Multi-scale Modeling`: A module for stochastic hierarchical multi-scale modeling is currently under development that advances deterministic capabilities developed in this field [49].

These future enhancements will further solidify `UQpy` as a leading software in the field and will be made possible by the streamlined architecture, development, and integration processes described herein. Please note that the list above is subject to change in future releases.

References

[1] Baudin M, Dutfoy A, Iooss B, Popelin A-L. OpenTURNS: An industrial software for uncertainty quantification in simulation. In: Ghanem R, Higdon D, Owhadi H, editors. Handbook of uncertainty quantification. Cham: Springer International Publishing; 2017, p. 2001–38. http://dx.doi.org/10.1007/978-3-319-12385-1_64.

[2] Martin SM, Wälchli D, Arampatzis G, Economides AE, Karnakov P, Koumoutsakos P. Korali: Efficient and scalable software framework for Bayesian uncertainty quantification and stochastic optimization. Comput Methods Appl Mech Engrg 2022;389:114264. http://dx.doi.org/10.1016/j.cma.2021.114264.

[3] Parno M, Davis A, Seelinger L. MUQ: The MIT uncertainty quantification library. J Open Source Softw 2021;6(68):3076. http://dx.doi.org/10.21105/joss.03076.

[4] Debusschere BJ, Najm HN, Pébay PP, Knio OM, Ghanem RG, Le Maıtre OP. Numerical challenges in the use of polynomial chaos representations for stochastic processes. SIAM J Sci Comput 2004;26(2):698–719. http://dx.doi.org/10.1137/S1064827503427741.

[5] Dalbey K, Eldred MS, Geraci G, Jakeman JD, Maupin KA, Monschke JA, Seidl DT, Swiler LP, Tran A, Menhorn F, Zeng X. Dakota a multilevel parallel object-oriented framework for design optimization parameter estimation uncertainty quantification and sensitivity analysis: Version 6.12 theory manual. 2020, http://dx.doi.org/10.2172/1630693.

[6] Patelli E, Broggi M, de Angelis M, Beer M. OpenCossan: An efficient open tool for dealing with epistemic and aleatory uncertainties. 2014, http://dx.doi.org/10.1061/9780784413609.258.

[7] Marelli S, Sudret B. UQLab: A Framework for Uncertainty Quantification in Matlab, pp. 2554–2563 arXiv:https://ascelibrary.org/doi/pdf/10.1061/9780784413609.257 http://dx.doi.org/10.1061/9780784413609.257.

[8] Dupuy D, Helbert C, Franco J. DiceDesign and DiceEval: Two r packages for design and analysis of computer experiments. J Stat Softw 2015;65(11):1–38. http://dx.doi.org/10.18637/jss.v065.i11.

[9] Roustant O, Ginsbourger D, Deville Y. DiceKriging, DiceOptim: Two r packages for the analysis of computer experiments by kriging-based metamodeling and optimization. J Stat Softw 2012;51(1):1–55. http://dx.doi.org/10.18637/jss.v051.i01.

[10] Bourinet J-M, Mattrand C, Dubourg V. A review of recent features and improvements added to FERUM software. In: Furuta H, Frangopol DM, Shinozuka M, editors. Proc. 10th International Conference on Structural Safety and Reliability (ICOSSAR 2009), Osaka, Japan, September 13–17, 2009. CRC Press; 2009.

[11] Gorissen D, Couckuyt I, Demeester P, Dhaene T, Crombecq K. A surrogate modeling and adaptive sampling toolbox for computer based design. J Mach Learn Res 2010;11(68):2051–5, URL http://jmlr.org/papers/v11/gorissen10a.html.

[12] Technical university of munich, TUM school of engineering and design, engineering risk analysis group. 2023, [link]. URL https://www.cee.ed.tum.de/era/software/.

[13] Tennøe S, Halnes G, Einevoll GT. Uncertainpy: A python toolbox for uncertainty quantification and sensitivity analysis in computational neuroscience. Front Neuroinform 2018;12. http://dx.doi.org/10.3389/fninf.2018.00049.

[14] Puzyrev V, Ghommem M, Meka S. PyROM: A computational framework for reduced order modeling. J Comput Sci 2019;30:157–73. http://dx.doi.org/10.1016/j.jocs.2018.12.004.

[15] Šukys J, Kattwinkel M. SPUX: Scalable particle Markov chain Monte Carlo for uncertainty quantification in stochastic ecological models. 2017, arXiv:1711.01410.

[16] Dutta R, Schoengens M, Pacchiardi L, Ummadisingu A, Widmer N, Künzli P, Onnela J-P, Mira A. ABCpy: A high-performance computing perspective to approximate Bayesian computation. J Stat Softw 2021;100(7):1–38. http://dx.doi.org/10.18637/jss.v100.i07.

[17] Herman J, Usher W. SALib: An open-source Python library for sensitivity analysis. J Open Source Softw 2017;2(9):97. http://dx.doi.org/10.21105/joss.00097.

[18] Salvatier J, Wiecki TV, Fonnesbeck C. Probabilistic programming in Python using PyMC3. PeerJ Comput Sci 2016;2:e55. http://dx.doi.org/10.7717/peerj-cs.55.

[19] Weise K, Poßner L, Müller E, Gast R, Knösche TR. Pygpc: A sensitivity and uncertainty analysis toolbox for Python. SoftwareX 2020;11:100450. http://dx.doi.org/10.1016/j.softx.2020.100450.

[20] Jakeman JD. PyApprox: Enabling efficient model analysis. 2022, http://dx.doi.org/10.2172/1879614, URL https://www.osti.gov/biblio/1879614.

[21] Zou Z, Meng X, Psaros AF, Karniadakis GE. NeuralUQ: A comprehensive library for uncertainty quantification in neural differential equations and operators. 2022, arXiv:2208.11866.

[22] Olivier A, Giovanis DG, Aakash B, Chauhan M, Vandanapu L, Shields MD. UQpy: A general purpose python package and development environment for uncertainty quantification. J Comput Sci 2020;47:101204. http://dx.doi.org/10.1016/j.jocs.2020.101204.

[23] Nájera Ó, Larson E, Estève L, Varoquaux G, Liu L, Grobler J, de Andrade ES, Holdgraf C, Gramfort A, Jas M, Nothman J, Grisel O, Varoquaux N, Gouillart E, Luessi M, Lee A, Vanderplas J, Hoffmann T, Caswell TA, Sullivan B, Batula A, jaeilepp, Robitaille T, Appelhoff S, Kunzmann P, Geier M, Lars, Sunden K, Stańczak D, Shih AY. Sphinx-gallery/sphinx-gallery: Release v0.7.0. 2020, http://dx.doi.org/10.5281/zenodo.3838216.

[24] Harris CR, Millman KJ, van der Walt SJ, Gommers R, Virtanen P, Cournapeau D, Wieser E, Taylor J, Berg S, Smith NJ, Kern R, Picus M, Hoyer S, van Kerkwijk MH, Brett M, Haldane A, del Río JF, Wiebe M, Peterson P, Gérard-Marchant P, Sheppard K, Reddy T, Weckesser W, Abbasi H, Gohlke C, Oliphant TE. Array programming with numPy. Nature 2020;585(7825):357–62. http://dx.doi.org/10.1038/s41586-020-2649-2.

[25] Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, Cournapeau D, Burovski E, Peterson P, Weckesser W, Bright J, van der Walt SJ, Brett M, Wilson J, Millman KJ, Mayorov N, Nelson ARJ, Jones E, Kern R, Larson E, Carey CJ, Polat İ, Feng Y, Moore EW, VanderPlas J, Laxalde D, Perktold J, Cimrman R, Henriksen I, Quintero EA, Harris CR, Archibald AM, Ribeiro AH, Pedregosa F, van Mulbregt P, Vijaykumar A, Bardelli AP, Rothberg A, Hilboll A, Kloeckner A, Scopatz A, Lee A, Rokem A, Woods CN, Fulton C, Masson C, Häggström C, Fitzgerald C, Nicholson DA, Hagen DR, Pasechnik DV, Olivetti E, Martin E, Wieser E, Silva F, Lenders F, Wilhelm F, Young G, Price GA, Ingold G-L, Allen GE, Lee GR, Audren H, Probst I, Dietrich JP, Silterra J, Webber JT, Slavič J, Nothman J, Buchner J, Kulick J, Schönberger JL, de Miranda Cardoso JV, Reimer J, Harrington J, Rodríguez JLC, Nunez-Iglesias J, Kuczynski J, Tritz K, Thoma M, Newville M, Kümmerer M, Bolingbroke M, Tartre M, Pak M, Smith NJ, Nowaczyk N, Shebanov N, Pavlyk O, Brodtkorb PA, Lee P, McGibbon RT, Feldbauer R, Lewis S, Tygier S, Sievert S, Vigna S, Peterson S, More S, Pudlik T, Oshima T, Pingel TJ, Robitaille TP, Spura T, Jones TR, Cera T, Leslie T, Zito T, Krauss T, Upadhyay U, Halchenko YO, Vázquez-Baeza Y, 1.0 Contributors S. SciPy 1.0: fundamental algorithms for scientific computing in Python. Nature Methods 2020;17(3):261–72. http://dx.doi.org/10.1038/s41592-019-0686-2.

[26] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay É. Scikit-learn: Machine learning in Python. J Mach Learn Res 2011;12(85):2825–30, URL http://jmlr.org/papers/v12/pedregosa11a.html.

[27] Schölkopf B, Smola A, Müller K-R. Nonlinear component analysis as a kernel eigenvalue problem. Neural Comput 1998;10(5):1299–319. http://dx.doi.org/10.1162/089976698300017467, arXiv:https://direct.mit.edu/neco/article-pdf/10/5/1299/813905/089976698300017467.pdf.

[28] Fletcher P, Lu C, Pizer S, Joshi S. Principal geodesic analysis for the study of nonlinear statistics of shape. IEEE Trans Med Imaging 2004;23(8):995–1005. http://dx.doi.org/10.1109/TMI.2004.831793.

[29] Tenenbaum JB, Silva Vd, Langford JC. A global geometric framework for nonlinear dimensionality reduction. Science 2000;290(5500):2319–23. http://dx.doi.org/10.1126/science.290.5500.2319.

[30] Harandi MT, Salzmann M, Jayasumana S, Hartley R, Li H. Expanding the family of grassmannian kernels: An embedding perspective. In: Fleet D, Pajdla T, Schiele B, Tuytelaars T, editors. Computer vision – ECCV 2014. Cham: Springer International Publishing; 2014, p. 408–23.

[31] Hamm J, Lee D. Extended Grassmann Kernels for subspace-based learning. In: Koller D, Schuurmans D, Bengio Y, Bottou L, editors. Advances in neural information processing systems, Vol. 21. Curran Associates, Inc.; 2008, URL https://proceedings.neurips.cc/paper_files/paper/2008/file/e7f8a7fb0b77bcb3b283af5be021448f-Paper.pdf.

[32] Hong J, Chen H, Lin F. Disturbance Grassmann Kernels for subspace-based learning. In: Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining. KDD '18, New York, NY, USA: Association for Computing Machinery; 2018, p. 1521–30. http://dx.doi.org/10.1145/3219819.3219959.

[33] Shields MD, Zhang J. The generalization of Latin hypercube sampling. Reliab Eng Syst Saf 2016;148:96–108. http://dx.doi.org/10.1016/j.ress.2015.12.002.
[34] Vořechovskỳ M. Hierarchical refinement of latin hypercube samples. Comput-Aided Civ Infrastruct Eng 2015;30(5):394–411.
[35] Swiler LP, Gulian M, Frankel AL, Safta C, Jakeman JD. A survey of constrained Gaussian process regression: approaches and implementation challenges. J Mach Learn Model Comput 2020;1(2):119–56.
[36] Novak L, Sharma H, Shields MD. On physically-constrained non-intrusive polynomial chaos expansion. In: 5th ECCOMAS thematic conference on uncertainty quantification in computational sciences and engineering. 2023.
[37] Sharma H, Shields MD, Novak L. Constrained non-intrusive polynomial chaos expansion for physics-informed machine learning regression. In: 14th international conference on applications of statistics and probability in civil engineering, ICASP14. 2023.
[38] Coifman RR, Lafon S. Geometric harmonics: A novel tool for multiscale out-of-sample extension of empirical functions. Appl Comput Harmon Anal 2006;21(1):31–52. http://dx.doi.org/10.1016/j.acha.2005.07.005, Special Issue: Diffusion Maps and Wavelets.
[39] Marrel A, Iooss B, Laurent B, Roustant O. Calculations of Sobol indices for the Gaussian process metamodel. Reliab Eng Syst Saf 2009;94(3):742–51. http://dx.doi.org/10.1016/j.ress.2008.07.008.
[40] Gorodetsky AA, Geraci G, Eldred MS, Jakeman JD. A generalized approximate control variate framework for multifidelity uncertainty quantification. J Comput Phys 2020;408:109257. http://dx.doi.org/10.1016/j.jcp.2020.109257.
[41] Chakroborty P, Dhulipala SLN, Che Y, Jiang W, Spencer BW, Hales JD, Shields MD. General multi-fidelity surrogate models: Framework and active learning strategies for efficient rare event simulation. 2022, arXiv:2212.03375.
[42] Dhulipala SL, Shields MD, Chakroborty P, Jiang W, Spencer BW, Hales JD, Labouré VM, Prince ZM, Bolisetti C, Che Y. Reliability estimation of an advanced nuclear fuel using coupled active learning, multifidelity modeling, and subset simulation. Reliab Eng Syst Saf 2022;226:108693. http://dx.doi.org/10.1016/j.ress.2022.108693.
[43] Dhulipala SL, Shields MD, Spencer BW, Bolisetti C, Slaughter AE, Labouré VM, Chakroborty P. Active learning with multifidelity modeling for efficient rare event simulation. J Comput Phys 2022;468:111506. http://dx.doi.org/10.1016/j.jcp.2022.111506.
[44] Olivier A, Shields MD, Graham-Brady L. Bayesian neural networks for uncertainty quantification in data-driven materials modeling. Comput Methods Appl Mech Engrg 2021;386:114079. http://dx.doi.org/10.1016/j.cma.2021.114079.
[45] Raissi M, Perdikaris P, Karniadakis GE. Physics informed deep learning (Part I): Data-driven solutions of nonlinear partial differential equations. 2017, arXiv:1711.10561.
[46] Lu L, Jin P, Pang G, Zhang Z, Karniadakis GE. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. Nat Mach Intell 2021;3(3):218–29. http://dx.doi.org/10.1038/s42256-021-00302-5.
[47] Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow I, Harp A, Irving G, Isard M, Jia Y, Jozefowicz R, Kaiser L, Kudlur M, Levenberg J, Mane D, Monga R, Moore S, Murray D, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker P, Vanhoucke V, Vasudevan V, Viegas F, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu Y, Zheng X. TensorFlow: Large-scale machine learning on heterogeneous distributed systems. 2016, arXiv:1603.04467.
[48] Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, Desmaison A, Kopf A, Yang E, DeVito Z, Raison M, Tejani A, Chilamkurthy S, Steiner B, Fang L, Bai J, Chintala S. PyTorch: An imperative style, high-performance deep learning library. In: Wallach H, Larochelle H, Beygelzimer A, d' Alché-Buc F, Fox E, Garnett R, editors. Advances in neural information processing systems, Vol. 32. Curran Associates, Inc.; 2019.
[49] Knap J, Spear C, Leiter K, Becker R, Powell D. A computational framework for scale-bridging in multi-scale simulations. Internat J Numer Methods Engrg 2016;108(13):1649–66. http://dx.doi.org/10.1002/nme.5270, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.5270.