Available online at www.eccomasproceedia.org
Eccomas Proceedia UNCECOMP (2025) 331-356

ECCOMAS
Proceedia

UNCECOMP 2025
6th ECCOMAS Thematic Conference on
Uncertainty Quantification in Computational Science and Engineering
M. Papadrakakis, V. Papadopoulos, G. Stefanou (eds.)
Rhodes Island, Greece, 15-18 June 2025

MULTI-DISCIPLINARY DESIGN OPTIMIZATION UNDER UNCERTAINTY: THE OPEN SOURCE CAPABILITIES OF GEMSEO

Matthias De Lozzo[1], Clément Laboulfie[1], Olivier Sapin[1], Nicolas Roussouly[1], François Gallard[1], Amine Aziz-Alaoui[1], Antoine Dechaume[1], and Anne Gazaix[1]

[1]IRT Saint Exupéry
Toulouse, France, 31400
e-mail: matthias.delozzo@irt-saintexupery.com

Abstract. *The increasing complexity of systems involved in engineering applications requires a breakthrough in the management of coupled models, also called disciplines, to be able to design new products in line with nowadays constraints. This design can be supported by multi-disciplinary optimization (MDO) methods, in order to handle optimization and coupling problems together by means of MDO formulations, a.k.a. MDO architectures. The Python library GEMSEO automates the MDO process generation from the MDO problem specification and a given MDO formulation. GEMSEO can also be used for surrogate modeling, to visualize optimization profiles, to manage data as the process progresses, etc. Over the last years, the GEMSEO team put a major effort to add uncertainty quantification (UQ) techniques into GEMSEO, and at the core of this, the extension of the MDO formulations to uncertainty. A unified way to extend existing deterministic MDO formulations (MDF, IDF, multi-level ones) has been proposed. Besides, the whole MDO suite has been extended to account for uncertain effects: defining and visualizing uncertainties, propagating them through multi-disciplinary systems, and performing statistical analysis of the quantities of interest, such as sensitivity analysis. In the last years, GEMSEO plugins have also been enriched to allow batch acquisition and quantile estimation with active learning methods, and to solve MDO problems under uncertainty, in order to promote the use of UQ techniques, not only in academia but also in industry. In this work, we present how GEMSEO and its plugins can be useful to carry out multi-disciplinary studies with uncertainties. We first give a brief overview of GEMSEO. Then, we demonstrate some capabilities for propagating uncertainties through weakly and strongly coupled disciplines, and solving MDO problems under uncertainty, with a focus on the use of surrogate-based variance reduction methods. In this work, we also show how GEMSEO can be enriched with new algorithms and software libraries, through intensive use of the object-oriented paradigm. Finally, we give some ideas for the development of UQ capabilities in GEMSEO in the coming years.*

Keywords: Multi-Disciplinary Optimization, Uncertainty Quantification Robust Optimization, Sensitivity Analysis, Surrogate Models, Open Source Software.

ISSN:2623-3339
Peer-review under responsibility of the organizing committee of UNCECOMP 2025.
doi: 10.7712/120225.12360.21300

1 INTRODUCTION

Optimization problems are common to design the product or the strategy that minimizes a cost (or maximizes a performance) while respecting constraints. An optimization problem can be written in a standard way as

$$\begin{array}{lll} \min\limits_{x} & f(x,u) & \\ \text{s.t.} & g(x,u) & \leq 0 \\ & h(x,u) & = 0 \\ & x \in \mathcal{X} & \end{array} \tag{1}$$

where x groups the optimization variables, f is the objective function (either the cost or the inverse of the performance), g are inequality constrained functions, h are equality constrained functions, and u groups various parameters, typically, in the following, uncertain variables.

1.1 Multi-disciplinary design optimization

In the context of multi-disciplinary design optimization (MDO), the optimization problems are generated from processes, which are composed of several models called disciplines. This specificit comes from the fact that disciplines are linked by coupling variables y, some weakly (or one-way coupling), others strongly (or two-way coupling, i.e., more rigorously, when the disciplines are involved in cycles of the input/outputdependency graph). The handling of these couplings, combined with optimization, is addressed by the so-called MDO formulations or MDO architectures [14]. An MDO architecture is the combination of an optimization problem and a computational process that computes the objective function and constraints for given values of the optimization variables. There are several MDO formulations, which reformulate the original design problem, called the SAND formulation, into computationally tractable ones. The most widely used is MDF (multi-disciplinary feasible), which formulates an optimization problem that is very similar to Problem (1). The coupling variables do not appear in this problem because the process associated with MDF is the multi-disciplinary analysis (MDA), that computes the coupling variables at each iteration of the optimization. They are implicit functions of the design variables. Another very common formulation, because of its simplicity, is the individual design feasible (IDF) formulation, which executes all disciplines in parallel at each iteration, where the coupling consistency being ensured, at convergence, by specifi equality constraints added to the optimization problem, and coupling variable targets y_t being added to the optimization variables. The optimization problem associated with IDF is also similar to Problem (1), when x is replaced by x, y_t and consistency constraints $c(x, y_t, u) = y(x, y_t, u) - y_t = 0$ are added to the problem, in addition to the user equality constraints $h(x, u) = 0$.

Some MDO formulations involve multiple optimization problems, and are then called distributed, or multi-level when the problems are nested such as CSSO and BLISS, each with its own advantages and disadvantages depending on the use case [29]. More recently, [9] proposed a bi-level formulation that is particularly suited to studies already equipped with disciplinary optimizers, e.g. structure sizing, or to studies where it would be a good idea to separate the optimization problem by type of variables, e.g. continuous vs. discrete variables. These formulations also generate optimization problems that are similar to Problem (1), since they are derived from MDF.

1.2 The open source Python library GEMSEO

In the face of this, the open source Python library GEMSEO allows the user to set up the optimization problem (1) from a collection of disciplines and a catalog of MDO formulations, including bi-level ones [9], without having to connect the disciplines by hand. It also enables the process to be automatically reconfigure when any element of the problem is changed, e.g. the design space, a discipline or the MDO formulation. In addition to the MDO formulations, optimization and coupling algorithms mentioned above, GEMSEO can be used to create surrogate models, to visualize optimization profiles to manage data as the process progresses, etc. Any type of model can be connected to GEMSEO, through specifi wrappers of Python functions, external executables, Excel spreadsheets, Matlab scripts or others. It also offers several features for parallel computation and remote execution via SSH, HTTPS, or HPC job schedulers.

GEMSEO was initially created in 2015 within the MDO competence center of IRT Saint Exupéry. This center is dedicated to the development of process automation technologies encompassing a wide range of disciplines, and their usage in a range of applications. Originally known as GEMS (Generic Engine for MDO Scenarios), this Python package became GEMSEO in 2021 when IRT Saint Exupéry and its partners decided to release it as an open source project with the aim to make it widely collaborative. GEMSEO is the acronym for generic engine for multi-disciplinary scenarios, exploration and optimization. It is the result of successive projects carried by IRT Saint Exupéry, funded by both Investments for the Future Programme (french acronym: PIA) and companies and involving both academic and industrial partners. It has been enriched by contributions from European projects. The GEMSEO team organized the firs annual User Day on November 28, 2024 to bring the community together and share experiences.

GEMSEO is open source under the GNU LGPL v3.0 license for the source code, BSD 0-Clause for the examples and CB BY-SA 4.0 license for the documentation. It gives users the right to to use the software, study and analyze its source code, to modify the software and thus its source code, to distribute the original software or its derivatives. The project is hosted on gitlab (`www.gitlab.com/gemseo/dev/gemseo`), the main documentation can be accessed at `www.gemseo.org` and the packages are available on pypi. To improve the quality of GEMSEO and release it faster, the core development team uses the agile-based practices: short incremental cycles, daily meeting, conception review before coding, review before integrating changes and software development forge. The source code comes with automated and mandatory testing (continuous integration), reproducible and isolated testing (automation project *tox*, Python environments, controlled dependencies), on all supported Python versions, on major target OS (Linux, Windows) and with high code coverage ($>$90%). All the tests must always pass on the reference source code (develop branch). In terms of code quality, automatic verification are made for every code changes (pre-commit hooks) and refactoring is done continuously. Lastly, documenting is done continuously, close to the source code and reviewed like the code.

1.3 GEMSEO and UQ

In the recent years, accounting for uncertainties in optimization problems has become a major research topic, including MDO problems [6]. Typically, physical constants usually assumed to be known are in fact uncertain, the optimum design (exact or found numerically) is not exactly the one used in practice, the initial conditions of the disciplines are poorly known, etc. And all these sources of uncertainty, whether they come from one or several disciplines, actually impact a large number of disciplines, since they are coupled. This leads to algorithmic

problems in easily propagating uncertainties, especially in the presence of strong couplings. For that reason, the GEMSEO team has developed uncertainty quantificatio (UQ) capabilities, using the framework of probability theory, and has completed its package `gemseo` with `gemseo-umdo` (documentation) and `gemseo-mlearning` (documentation), two specialized plugins (by GEMSEO, we mean `gemseo` and its plugins). These packages are based on specialized open source Python libraries, such as OpenTURNS [3] for UQ, SMT [5] and scikit-learn [17] for surrogate modelling and NumPy and SciPy for applied mathematics.

Figure 1 illustrates the main steps of an MDO study under uncertainty. The aim of GEMSEO is to carry out these steps with a unique software environment, to standardize the user experience, avoid the need to master different software libraries and facilitate the transition from MDO to UQ, which are two potentially complex field of applied mathematics in engineering departments. First (Step A), we defin the MDO problem under uncertainty: which input uncertainties, which statistical objective to optimize, which statistical constraints to satisfy, which design variables to control, which disciplines to compute the f, g and h from x and U and which MDO formulation to generate the execution process. Then (Step B), we need to defin the probability distributions characterizing the uncertain input variables. In a third stage (Step C), we have to propagate these uncertainties through the multi-disciplinary process, using sampling typically. Possibly (Step C'), the contribution of the uncertain inputs to the quantities of interest is studied, variables are sorted and non-influence are set to nominal values. Lastly (Step D), Steps C and potentially C' are repeated in an optimization loop until a stopping criterion is met (hoping for optimum value). Note that Figure 1 is adapted from one classically used in the UQ literature [20], to which we have added MDO features (disciplines, MDO formulation, objective, constraints and design variables) and Step D.

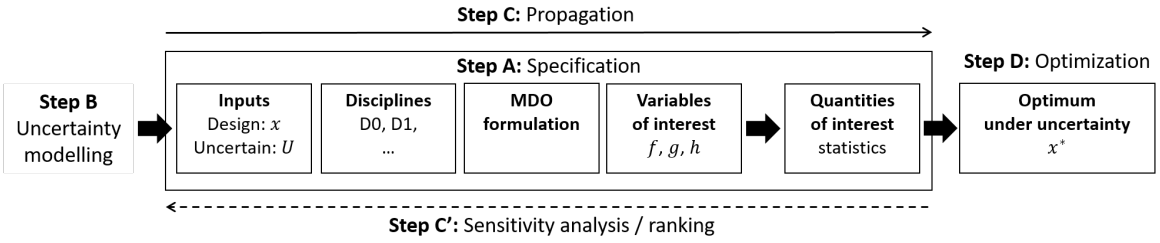

Figure 1: Breakdown of an MDO study under uncertainty (adapted from [20]).

1.4 Outline

In what follows, Section 2 presents the main concepts of GEMSEO on which the UQ elements have been built. Then, Section 3 shows how to defin and propagate uncertainties through a multi-disciplinary system, and study the sensitivity of quantities of interest to sources of uncertainty. Section 4 focuses on the definitio and resolution of an MDO problem under uncertainty, with different kinds of statistic estimators, illustrates these capabilities on an academic use case and proposes benchmark problems to compare U-MDO techniques. Finally, Section 5 recalls the main points and gives a vision of the GEMSEO roadmap concerning UQ and U-MDO.

2 BASIC CONCEPTS OF GEMSEO

This section provides a brief overwiev of the main Python classes of GEMSEO on which the UQ elements rely. It concludes with an UML class diagram summarizing these elements.

In this paper, the expressions *NumPy array* and *float number* will be replaced by *array* and *float* for the sake of brevity. In other respects, bear in mind that about everything in GEMSEO can be enriched by deriving a Python class, either directly in `gemseo` or in its existing plugins or in a new public or private plugin. Your new class will then be automatically recognized by GEMSEO, via a factory mechanism. As an example, if a `Color` class has an associated `ColorFactory`, then this factory can create an instance of any class derived from `Foo` without knowing its import path. This child class of `Color` can be define either in `gemseo`, in a public plugin or in a module of your Python path if you want to keep your development private. So, in practice, a `Square` class equipped with a `ColorFactory` can generate `Square` objects from any color class derived from `Color`. You only have to know the class name, and instantiate the class as `blue_square = Square("BlueSquare")`.

MDOFunction The most basic concept of GEMSEO is the `MDOFunction`, which wraps a callable object, e.g. a Python function, taking an array as argument and returning either an array or a float It can also wrap the function computing its Jacobian matrix expressed as an array from an array. An `MDOFunction` has methods to compute the output value (array or float) and the Jacobian matrix (array), from the input value (array). It is equipped with algebraic operators, namely addition, subtraction, multiplication, division and negation, in order to rewrite the functions of the optimization problem in a standardized way: the objective functions as cost functions, the inequality constraint functions as negativity constraints, and the constraint functions with a zero threshold.

DesignSpace A `DesignSpace` is a dictionary-like object to defin scalar and vector variables from a name, a lower bound, an upper bound, a type (integer or real) and a current value. A point in the design space can be represented either as a dictionary {`variable_name: variable_value, ...`}, where `variable_value` is an array, or as a full array concatenating the sub-arrays. It offers easy-to-use methods for switching from one representation to another, as well as others to express the point according to a particular transformation, such as normalization into the unit hypercube from bounds.

EvaluationProblem An `EvaluationProblem` is set of `MDOFunctions`, a `DesignSpace` and a method to evaluate these `MDOFunctions` using a mechanism to check that the evaluation point respects the bounds of the `DesignSpace`. It also proposes a method for replacing the original `MDOFunctions` by specifi `MDOFunctions`, called `ProblemFunctions`, that can count their evaluations, transform the evaluation point according to the `DesignSpace`, store their evaluations in a `Database` to avoid evaluating an `MDOFunction` twice at a same evaluation point. This `Database` can be converted into a `Dataset`, which is a multi-index pandas DataFrame [28] easier to handle.

OptimizationProblem An `OptimizationProblem` is an `EvaluationProblem` whose `MDOFunctions` are labelled as *objective*, *constraints* and *observables*.

DriverLibrary `BaseDriverLibrary` is the base class to interface a library of algorithms, called drivers. Once instantiated from an algorithm name, its `execute` method uses this algorithm to evaluate the `MDOFunctions` of an `EvaluationProblem` from algorithm settings. In particular, `BaseDOELibrary` is the base class to interface design of experiments (DOE)

algorithms, while `BaseOptimizationLibrary` is the base class to interface optimization algorithms and is limited to `OptimizationProblems`. GEMSEO gives access to various libraries, such as SciPy, pyDOE3 and OpenTURNS for DOEs and Sci Py, NLopt and pymoo for optimizers.

Discipline A discipline is define by a `Discipline` whose `execute` method computes labeled output data of the form {`output_name: output_value, ...`} from labeled input data of the form {`input_name: input_value, ...`}. The `Discipline` can also compute the Jacobian of the form {`output_name: {input_name: jac_value, ...}, ...`} when the discipline is differentiable, either analytically or by approximation, e.g. finit differences. The input and output variables of a discipline are define by grammars, in terms of names, types and compulsoriness. At execution, the `Discipline` checks that the input and output data are consistent with these grammars. If the input data are incomplete, the `Discipline` completes them with default input values.

DependencyGraph A `DependencyGraph` analyzes the input and output variable names of a collection of `Disciplines` to infer how these depend on each other, by using the convention: two homonymous variables represent the same variable. It produces the sequence of execution of the disciplines, the coupling variables between two disciplines and a directed coupling graph. `DependencyGraph` can also display this graph, either in full form, with one node per discipline and one edge by coupling variable, condensed form, with one node per weakly coupled discipline or group of strongly coupled discipline. The user is led to use the concept of `Discipline` rather than that of `MDOFunction` which is dedicated to the process layer.

BaseFormulation The base class `BaseFormulation` define the concept of MDO formulation, which is both the definitio of the design space and objective and constraints functions, and the overall organization of the process. This `BaseFormulation` is composed of an `OptimizationProblem`, created from a `DesignSpace`, a set of `Disciplines`, names of discipline output variables to minimize or maximize, and names of discipline output variables to to be constrained. More precisely, the `BaseFormulation` plugs `MDOFunctions` into the `OptimizationProblem` and these `MDOFunctions` wrap the `Disciplines` according to the overall process. This process can be composed of several levels, and in some cases, sub-processes can be parallelized. `MDF`, `IDF`, `BiLevel` and `DisciplinaryOpt` are concrete classes deriving from `BaseFormulation`. `DisciplinaryOpt` define a process to execute the disciplines in the order in which they are given, while the others implement the eponymous MDO formulations presented in Section 1.1 using a `DependencyGraph` to identify the coupling variables and potentially deduce the sequence of execution of the disciplines. In the latter case, the formulations MDF and BiLe vel also include sub-processes implemented as `ProcessDiscipline` objects, such as sub-optimizations and MDAs.

MDOScenario Finally, `MDOScenario` is the *key class* for the end user. It aims to gather all elements necessary to define analyze, solve and post-process an MDO problem: a `Design-Space`, a set of `Disciplines`, the names of the discipline output variables to minimize or maximize, the names of the discipline output variables to be constrained, the name and settings

of an MDO formulation, and the name and settings of a driver. Here is an example to defin and solve the Sellar problem [25] using the MDF formulation:

```
disciplines = [
    AnalyticDiscipline({"y1: (z1**2+z2+x-0.2*y2)**0.5"}, name="Sellar1"),
    AnalyticDiscipline({"y2": "abs(y1)+z1+z2"}, name="Sellar2"),
    AnalyticDiscipline(
        {
            "f": "x**2+z2+y1**2+exp(-y2)",
            "c1": "3.16-y1**2",
            "c2": "y2-24",
        },
        name="SellarSystem"
    )
]

design_space = DesignSpace()
design_space.add_variable("x", lower_bound=0.0, upper_bound=10.0)
design_space.add_variable("z1", lower_bound=-10.0, upper_bound=10.0)
design_space.add_variable("z2", lower_bound=0.0, upper_bound=10.0)

scenario = create_scenario(
    disciplines, "obj", design_space, formulation_name="MDF"
)
scenario.add_constraint("c1", constraint_type="ineq")
scenario.add_constraint("c2", constraint_type="ineq")
scenario.execute(algo_name="SLSQP", max_iter=10)
```

Figure 2 summarizes this entire section in the form of a class diagram where classes are represented by boxes. The abstract classes have dashed borders, the edges with diamonds represent composition relationships (the class with the diamond is made up of the class at the other end) and the edges with triangles represent inheritance relationships (the class with the triangle is the parent of the class at the other end). The edges with arrows represent a certain causal relationship (a non-pointedclass uses a pointed class or generates an instance of this class).

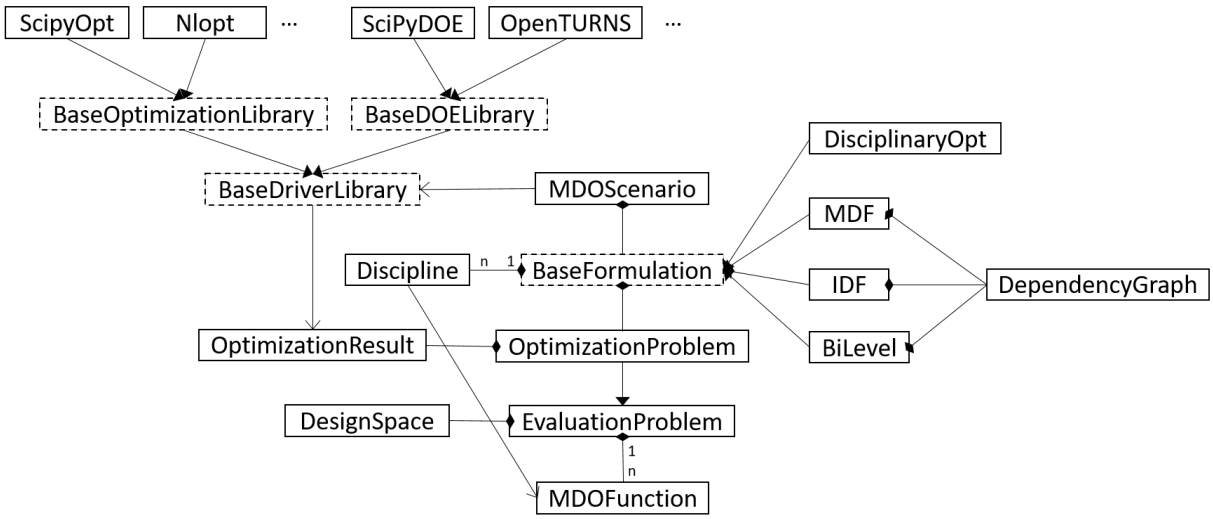

Figure 2: Class diagram of the basic concepts of GEMSEO.

3 UQ IN MULTI-DISCIPLINARY SYSTEMS

Although its main area of applied mathematics is optimization, as illustrated by Figure 2 with the `OptimizationProblem` and `OptimizationResult` classes for example, the technological core of GEMSEO is the automation of multi-disciplinary system creation, execution and reconfiguration This is illustrated in Figure 3, where an `EvaluationProblem` is automatically created from a `DesignSpace` and a set of `Disciplines`. This makes it possible to evaluate any output of a multi-disciplinary system from the value of one of its inputs, which multiplies the possibilities of use in other mathematical field (simple evaluation of the multi-disciplinary system, surrogate model of this system, visualization of an output of interest, sensitivity of the output of a discipline with respect to the input of another discipline, etc.). In particular, the sub-package `gemseo.uncertainty` was originally added not to optimize in presence of uncertainties, as we will see in Section 4, but to estimate statistics for structural composite models define as modular discipline chains [13], using the MDF formulation, as illustrated in Figure 4.

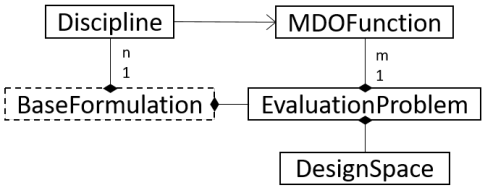

Figure 3: Minimal class diagram for definin a multi-disciplinary evaluation problem.

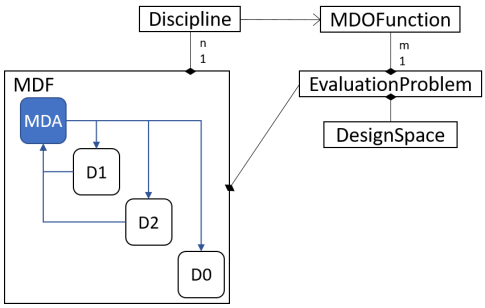

Figure 4: Minimal class diagram for definin a multi-disciplinary evaluation problem in an MDF way.

3.1 Uncertainty propagation through a multi-disciplinary system

Given a set of disciplines, an input variable space and a collection of output variables of interest, `gemseo` is able to generate a collection of `MDOFunctions` mapping from the input variable space to an output variable space and evaluate them, as explained in Section 2. This is typically what is done when the user creates an `MDOScenario` or calls the `sample_disciplines` function to sample a multi-disciplinary system over the input space when the notions of *objective* and *constraint* are not useful (in other words, we simply want to calculate "input values - output values" pairs). In the case of sampling, the input variables are usually define by bounds and thus a `DesignSpace` is perfectly suited to gather them. When these variables are random, a `ParameterSpace` deriving from `DesignSpace` is more appropriate as it can also defin scalar and vector variables from their probability distributions. `gemseo` offers an

interface to all the distributions available in OpenTURNS and SciPy, through the generic classes `OTDistribution` and `SPDistribution`, as illustrated in Figure 5. There are also specifi classes for specifi probability distribution, e.g. `OTNormalDistribution`. Note that the developer can interface other UQ libraries by deriving the base class `BaseDistribution`. The class will then be automatically recognized by `gemseo` and accessible via its name and arguments:

```
uncertain_space = create_parameter_space()
uncertain_space.add_random_variable("u", "OTNormalDistribution", sigma=0.8)
```

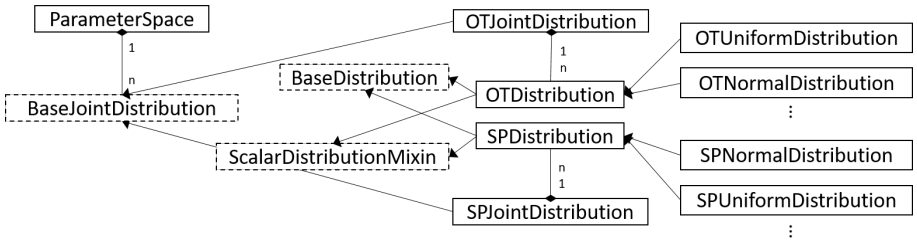

Figure 5: Class diagram for probability distributions and their use by `ParameterSpace`. OT and ST stand for OpenTURNS and SciPy respectively. `OTDistribution` can represent any probability distribution available in OpenTURNS. `OTNormalDistribution` represents the normal distibution available in OpenTURNS and `OTJointDistribution` is the probability distribution of a random vector whose marginals are `OTDistributions`.

As mentioned in Figure 5, random vectors can also be added to a `ParameterSpace`. By default, only the marginal probability distributions are define and the components of a random vector are stochastically independent. In the case of OpenTURNS distributions, the dependence structure can be define using a copula function, e.g. Gaussian copula.

Then, just as the `DesignSpace` can apply a geometric transformation to project a point into the unit hypercube, the `ParameterSpace` can apply the inverse transformation method to sample over the unit hypercube and project these intermediate points into the probabilized input space. Here is an illustration of sampling where the `sample_disciplines` function returns a `Dataset`:

```
samples = sample_disciplines(
    disciplines, uncertain_space, output_names, n_samples=100
)
```

Once the uncertainty sources propagated, the classes `EmpiricalStatistics` and `ParametricStatistics`, deriving from the base class `BaseStatistics`, come into play to estimate the statistics of interest, including tolerance intervals (especially A- and B-value). The firs one uses empirical estimators:

```
empirical_statistics = create_statistics(samples)
std = empirical_statistics.compute_standard_deviation()
```

while the second one uses the statistics of the best probability distributions, fitte from the output samples by either maximum likelihood or the method of moments, and selected from a list of candidates by a statistical hypothesis test or criterion:

```
parametric_statistics = create_statistics(
    samples,
    tested_distributions=["Exponential", "Normal", "Uniform"],
```

```
    fitting_criterion="Kolmogorov"
)
q10 = parametric_statistics.compute_quantile(0.1)
```

The `EmpiricalStatistics` class also provides the user with visualization tools, such as boxplots and plots of empirical probability density and cumulative probability functions:

```
boxplots = empirical_statistics.plot_boxplot()
```

All the figure have matplotlib [10] versions that can be modifie locally to match the user's graphical presentation of results (color, line styles, labels, etc.). This applies to all the figure that can be generated by GEMSEO. An effort is underway to port these figure to plotly in order to make them dynamic in a web page and improve the user experience.

3.2 Quantile estimation using active learning algorithms

Because of the slow convergence of the Monte Carlo method, the estimates of some statistics can be fl wed with a large variance, in particular when only a limited number of evaluations is available. A naive approach is to replace the function of interest by a surrogate model and sample it intensively. However, this requires a precise surrogate model, and so possibly a large training dataset. Another approach is to use active learning techniques that are iterative methods aiming to sequentially estimate various quantities of interest: optimas, contours, failure probabilities, quantiles, expected values, etc. These methods generally start by constructing a rough surrogate model (an approximation of a costly exact physical model much faster to run). Then, they iteratively look for new training points in order to update the surrogate model and increase the accuracy of the target quantity until a budget or accuracy criterion is reached. This search is guided by an acquisition criterion, also called infil criterion, to be minimized or maximized. This approach is inspired by the efficien global optimization (EGO) algorithm [12]. Another potential application of active learning techniques is the estimation of level-sets, i.e. the input values for which the model output is equal to a specifi value. In particular, this specifi value can be the α-quantile for a specifi $\alpha \in]0,1[$ [13].

`gemseo-mlearning` proposes the `ActiveLearningAlgo` class to perform an active learning study. This algorithm instantiates a `BaseAcquisitionCriterion` belonging to a `BaseAcquisitionCriterionFamily`, e.g. `EI` (expected improvement), `EF` (expected feasibility) or `U` (U-function) from the `Quantile` family [23, 19], in the case of quantile estimation. The `ActiveLearningAlgo` can also be used to explore the input space, optimize a function or estimate a level set. As illustrated in Figure 6, the `ActiveLearningAlgo` can use any DOE or optimization algorithm available in a `BaseDriverLibrary` to optimize the infil criterion and fin the next point to enrich the surrogate model. It should be noted that point acquisition can be carried out sequentially or in batch mode, to take advantage of parallelized calculations. Lastly, the surrogate model used by the `ActiveLearningAlgo` can be any `BaseRegressor`, which is the base class for regression models in GEMSEO, and interfaced from any machine learning library. In fact, even if most often active learning technique uses a Gaussian process regressor, (`gemseo` proposes `GaussianProcessRegressor` based on scikit-learn and `OTGaussianProcessRegressor` based on OpenTURNS), it can be applied to a wide range of surrogate models [4].

3.3 Sensitivity analyses

Once the uncertainties have been propagated and the quantity of interest estimated, the user may want to perform a sensitivity analysis (SA) to explain this quantity from the dif-

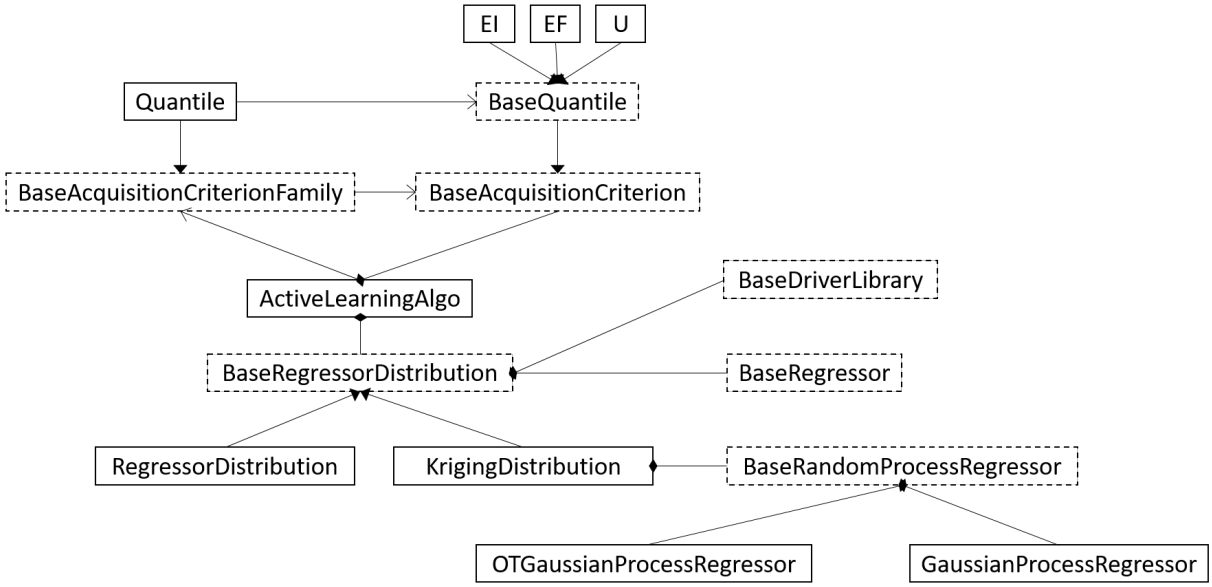

Figure 6: Class diagram of the active learning algorithm to estimate a quantile.

ferent sources of uncertainty. Figure 8 gives an overview of the classes implementing SA in `gemseo`. They derive from the base class `BaseSensitivityAnalysis`. Its method `compute_samples` is used to sample the multi-disciplinary system over an uncertain space using an `MDOFormulation`, a DOE which may be specifi to the type of SA and a maximum number of evaluations. This is the costly task in a SA, and for this reason it is advisable to save the samples generated by this method. This means you can later instantiate a new SA from these samples, without having to regenerate them. Note also that by default, the sampling is performed for all outputs of all disciplines, including coupling variables, so that we can calculate sensitivity indices for all these variables. However, it is possible to select a subset of variables of interest, to avoid storing data that we will never use. Then, the method `compute_indices` computes the sensitivity indices from these samples. The indices can be displayed using generic graphs, such as bar plot and radar chart, or specifi ones, whether related to the type of sensitivity analysis (see Figure 7) or to the functional nature of the quantity of interest. It is also possible to sort the sources of uncertainty, to prioritize uncertainty management, and compare the results of two different analyses on the same graph, to see whether or not they agree with the conclusions.

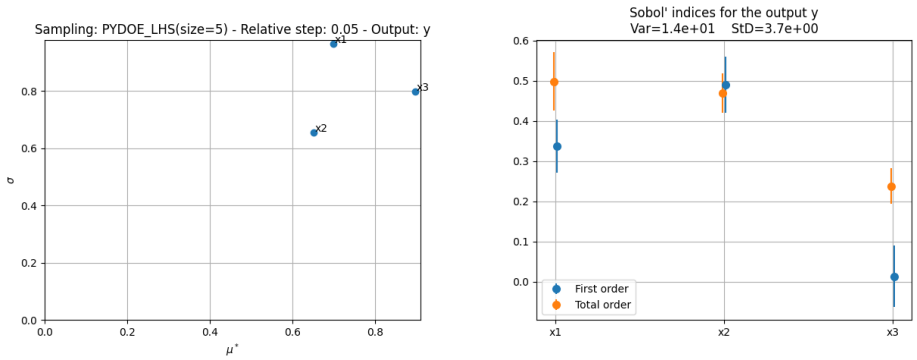

Figure 7: Visualization of the Morris indices (left) and Sobol indices (right) for the Ishigami function [11].

If we take a closer look, most of `SobolAnalysis`, `CorrelationAnalysis` and `HSICAnalysis` wrap features proposed by OpenTURNS while `MorrisAnalysis` is an in-house code. About the latter, you can perform a Morris analysis [15] using any DOE algorithm to place the trajectories in the input space, and must specify either the number of samples required, or the number of trajectories. You can also use the Morris sampling technique, or the underlying one-at-a-time (OAT) one in charge to generate a trajectory, independently of the `MorrisAnalysis`, thanks to the DOE algorithms `MorrisDOE` and `OATDOE`. Regarding `CorrelationAnalysis` and `HSICAnalysis`, they do not use a specifi DOE algorithm (just Monte Carlo sampling), which makes them popular methods. The method `compute_indices` of `HSICAnalysis` proposes three SA variants, namely global SA (default) to identify the input variables most likely to cause the output to deviate from a nominal value, target SA to identify the input variables most likely to cause the output to reach a certain domain, and conditional SA to identify the input variables most likely to cause the output to deviate from a nominal value under the condition that the considered output is in a certain domain. Lastly, `SobolAnalysis` proposes different state-of-the-art algorithms to estimate the first-orde , second-order and total indices from the samples. In the case of the Saltelli algorithm [24], the user can reduce the variance of the estimator at no extra cost, by providing cheap disciplines to be used as control variates, including surrogate models in the spirit of [7]. Using control variates is uncommonin UQ software libraries while the literature has demonstrated the value of these variance reduction methods, sometimes with several orders of magnitude.

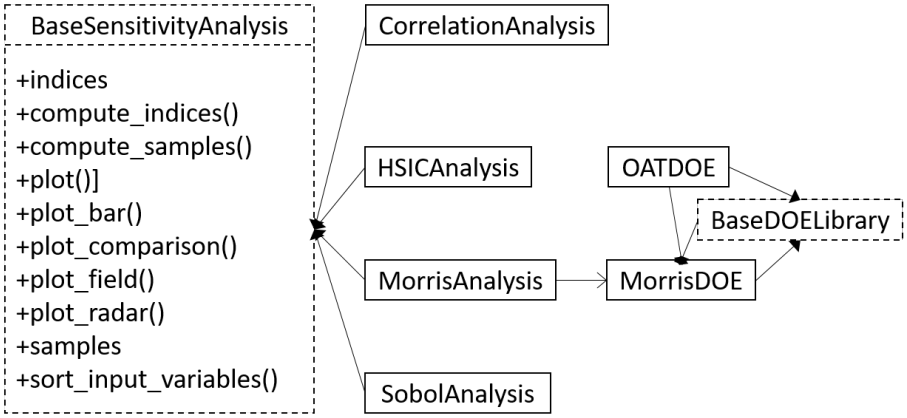

Figure 8: Class diagram of the sensitivity analysis algorithms.

3.4 Networks to visualize uncertainties

Calculating statistics is one thing, visualizing them is quite another, especially in high dimension, for example in presence of a large number of sources of uncertainty or many coupling variables. Networks are often used to represent this kind of data.

3.4.1 First-, second- and total-order Sobol' indices

Some ten years ago, [16] proposed the FANOVA graph to display the first- second- and total-order Sobol' indices at the same time. A node represents an uncertain variable whose name is written inside, followed by its first-orde and total-order Sobol' indices. The thickness of a node is proportional to the total-order Sobol' index of the variable while the thick-

ness of an edge is proportional to the second-order Sobol' index of the corresponding pair of variables. The `SobolGraph` class of the package `gemseo-umdo` implements the FANOVA graph, either from a `SobolAnalysis` or from the indices directly. The user can change the maximum thickness of a line and the significanc level below which a second-order index is not displayed. Figure 9 illustrates this feature using the Ishigami function $f(X_1, X_2, X_3) = \sin(X_1) + 7\sin(X_2)^2 + 0.1X_3^4\sin(X_1)$ where X_1, X_2 and X_3 are independent and identically distributed according to the uniform distribution over $[-\pi, \pi]$ [11]. This clearly shows the first order index of zero for X_3 and its interaction with X_1 which leads to a total index of 24%.

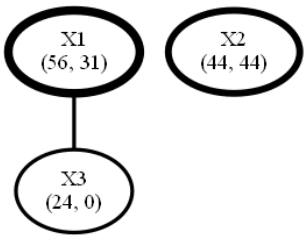

Figure 9: FANOVA graph for the Ishigami function.

3.4.2 Uncertain coupling variables

Usually, we perform a sensitivity analysis to identify the impact of uncertain inputs on a quantity of interest. In the case of an optimization problem, this quantity could be an objective or a constraint. In the case of a multi-disciplinary system, some disciplines may be less subject to uncertainties than others. Some disciplines might even be insensitive to uncertainties. Then, it would be wise not to re-evaluate them when sampling this system to estimate statistics of interest. For that reason, `gemseo-umdo` has an `UncertainCouplingGraph` class that identifie the coupling variables that are not impacted by the uncertain inputs, and so the disciplines that are not impacted by the uncertain inputs.

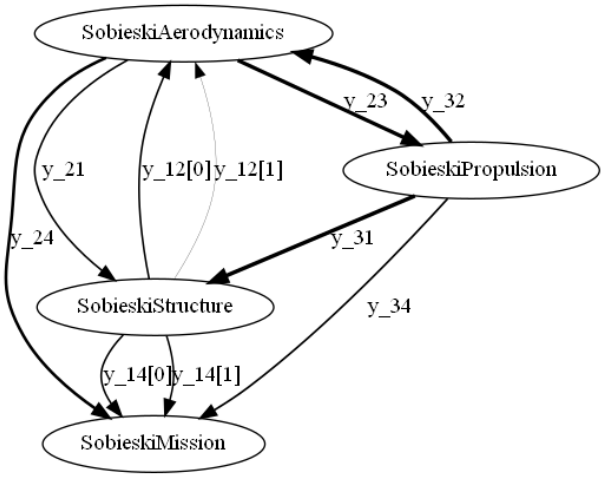

Figure 10: Uncertain coupling graph for the Sobieski' s SSBJ problem.

Figure 10 illustrates this concept from the Sobieski' s SSBJ problem [26] that includes three strongly coupled disciplines, called Aerodynamics, Propulsion and Structure, computing the

disciplinary constraints, and a fourth one, called Mission, computing the objective function from the others. In this example, the design variables are transformed into independent random variables and uniformly distributed over an interval of $\pm$ 5% around the optimum. In this network, the nodes represent the disciplines while the edges represent the coupling variables. The thickness of an edge is proportional to the absolute value of the dispersion of the corresponding coupling variable. A node connected only by ultra-thin edges, e.g. `y_12[1]`, will therefore be judged to be very insensitive to uncertain inputs. In terms of options, the user can select the dispersion measure (either the coefficien of variation or the quartile coefficien of dispersion) and the coupling variables of interest (displaying a lot of coupling variables can look like a plate of spaghetti).

4 MDO PROBLEMS UNDER UNCERTAINTY

When the sources of uncertainty have a significan impact on the objective and constraints, which are the variables of interest of the MDO problem, it is important to take them into account when definin the optimization problem. Originally, the GEMSEO team has created the `gemseo-umdo` plug-in to defin and solve optimization problems under uncertainty of the form

$$\begin{array}{lll} \min\limits_{x} & \mathbb{K}_f[f(x,U)] & \\ \text{s.t.} & \mathbb{K}_g[g(x,U)] & \leq 0 \\ & \mathbb{K}_{h^-}[h(x,U)] & \leq 0 \\ & \mathbb{K}_{h^+}[h(x,U)] & \geq 0 \\ & x \in \mathcal{X} & \end{array} \tag{2}$$

where $\mathbb{K}_f$, $\mathbb{K}_g$ and $\mathbb{K}_h$ are statistical operators, e.g. expectation or variance, and f, g and h are objective and inequality constrained functions define on both the design and uncertain spaces. Note that the uncertain parameters u are now replaced by random parameters U. The question of equality constraints under uncertainty remains an open problem, and we have chosen not to provide any particular capability to deal with them. Answering this question is part of the roadmap of `gemseo-umdo`. A naive approach is to replace them with both negativity and positivity constraints, as described in Problem (2). A more complex way would be to consider equality constraints for different realizations of U or for different statistics of $h(x,U)$ [6]. In the case of the IDF formulation, the consistency constraints would be addressed through this prism. However, this would increase significantl the dimension of the optimization, especially in the case of the approach with realizations.

4.1 An intuitive interface for UQ novices

The `UMDOScenario` class can be used to rewrite any MDO problem under uncertainty as Problem (2). Its interface is very similar to that of `MDOScenario` to ease the transition from a deterministic to an uncertain world, whether in terms of problem definition resolution or results management. As in the case without uncertainties, the user must specify

- the disciplines outputting the variables of interest,
- the disciplinary output f to minimize or maximize,
- the disciplinary outputs g and h to be constrained,
- the design space in which to perform the optimization,

- the MDO formulation to create the multi-disciplinary process to evaluate the functions f, g and h.

In the specifi context of uncertainties, the user must also specify:

- the uncertain space,
- the statistical operators for the constraints and objective,
- the statistics estimation technique (see Section 4.4).

As you can see in the following code, it is easy to go from an MDO problem without uncertainty (left) to one with uncertainty (right):

```
scenario = MDOScenario(
    [disciplines],
    "obj",
    design_space,


    formulation_name="MDF",



)  # min obj(x,u_bar) over x_space
scenario.add_constraint(
    "g"
)  # s.t. g(x,u_bar) <= 0
```

```
scenario = UMDOScenario(
    [disciplines],
    "obj",
    design_space,
    uncertain_space,
    "Mean",
    formulation_name="MDF",
    statistic_estimation_parameters={
        "n_samples": 100
    },
)  # min E[obj(x,U)] over x_space
scenario.add_constraint(
    "g", "Margin", factor=3.0
)  # s.t. E[g(x,U)]+3S[g(x,U)] <= 0
```

Once created, the user can execute this `UMDOScenario` with any optimization or DOE algorithm interfaced to `gemseo` and then post-process the results, in the same way as in the case without uncertainty:

```
scenario.execute(algo_name="NLOPT_COBYLA", max_iter=50)
scenario.post_process("OptHistoryView")
```

4.2 U-MDO formulation engine

Section 4.1 showed that the user must defin a pair of MDO formulation and statistics estimation technique. `gemseo-umdo` calls this duo a U-MDO formulation. It can be seen as a nested multi-disciplinary process. The inner process is orchestrated by the MDO formulation and generates the functions

$$u \mapsto f(x,u) \qquad u \mapsto g(x,u) \qquad u \mapsto h(x,u)$$

define over the uncertain space for a given value of x. The outer process is orchestrated by the statistics estimation technique and generates the functions

$$x \mapsto \widehat{\mathbb{K}}_f[f(x,U)] \qquad x \mapsto \widehat{\mathbb{K}}_g[g(x,U)] \qquad x \mapsto \widehat{\mathbb{K}}_h[h(x,U)]$$

where $\widehat{\mathbb{K}}_f$, $\widehat{\mathbb{K}}_g$ and $\widehat{\mathbb{K}}_h$ are the estimators of the statistics $\mathbb{K}_f$, $\mathbb{K}_g$ and $\mathbb{K}_h$ respectively. Finally, the estimators $\widehat{\mathbb{K}}_f$, $\widehat{\mathbb{K}}_g$ and $\widehat{\mathbb{K}}_h$ are plugged into the Problem (2) which is ready to be solved numerically. We can thus see that a so-called U-MDO formulation is indeed an MDO formulation in the MDO sense, namely both a workfl w to evaluate the objective and constraints functions

expressed as statistics estimators, and a datafl w to connect the disciplines and sub-processes using labelled data. Therefore, `UMDOScenario` is a unifie way to extend the MDO formulations usable with `gemseo` (MDF, IDF, multi-level ones) to the frame of U-MDO. In the same way as `gemseo` offers an MDO formulation engine, `gemseo-umdo` offers a U-MDO formulation engine, illustrated by the class diagram in Figure 11.

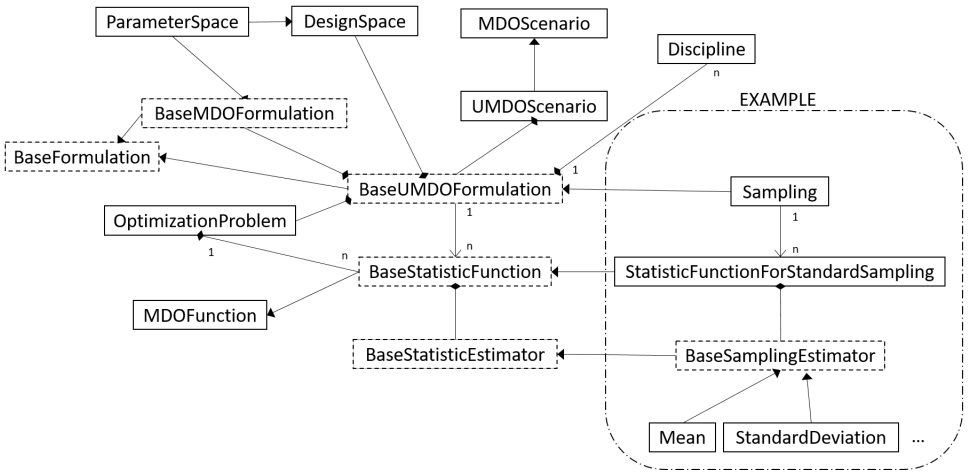

Figure 11: Class diagram of the U-MDO formulation engine. A `UMDOScenario` is a `MDOScenario` whose `BaseFormulation` is a `BaseUMDOFormulation`, e.g. `Sampling` when estimating the statistics using a sampling technique. This `BaseUMDOFormulation` is also composed of a `BaseMDOFormulation`, e.g. `MDF`. The `BaseMDOFormulation` includes a `ParameterSpace`, definin the uncertain variables, while the `BaseUMDOFormulation` includes a `DesignSpace`, defin ing the design variables. The `OptimizationProblem` composing the `BaseUMDOFormulation` uses `BaseStatisticFunctions` as `MDOFunctions`, estimating the statistics helped by a `BaseStatisticEstimator`. `BaseStatisticFunction` and `BaseStatisticEstimator` are subclassed for a specifi `BaseUMDOFormulation`, e.g. `StatisticFunctionForStandardSampling` and `BaseSamplingEstimator`. Finally, `BaseSamplingEstimator` derives into one class per statistic, e.g. `Mean` and `StandardDeviation`, whose class names are used as keywords by the end-user (see `"Mean"` and `"Margin"` in the previous code lines).

In an easier-to-read format, Figure 12 is a graphical view of this feature when combining Monte Carlo (MC) sampling (outer loop in orange) and an MDF formulation (inner loop in blue), with a Jacobi algorithm for the MDA. In this example, the MDO problem is unconstrained and the expectation is the statistical operator for the objective. In the absence of uncertainty, the outer loop would not appear and the discipline D_0 would directly return $f(x, u)$ to the optimizer at top left. In the next sub-section, we illustrate it with an application to the Sobieski' s Super-Sonic Business Jet (SSBJ) problem [26].

4.3 Illustration on an aeronautical application

In [26], the goal was to maximize the range y_4 of a SSBJ under the constraints $g_i \leq 0,\ 1 \leq i \leq 3$, with respect to different design variables, such as the shared design variables x_{shared}. In this study, we consider this design variable as uncertain and model by the random variable $X_{shared} = dv_{x_{shared}} + U_{x_{shared}}$ where $dv_{x_{shared}}$ is the design vector controlled by the optimizer and $U_{x_{shared}}$ is a random vector with zero mean. The minimum drag coefficien C_4 is also considered as uncertain. Now, the goal is to maximize the expectation of the range y_4 under the constraints $\mathbb{E}[g_i] + 2\mathbb{S}[g_i] \leq 0,\ 1 \leq i \leq 3$ and the uncertain variables $U_{x_{shared}}$ and C_4, with respect to different design variables. The following lines describe how to defin and solve this

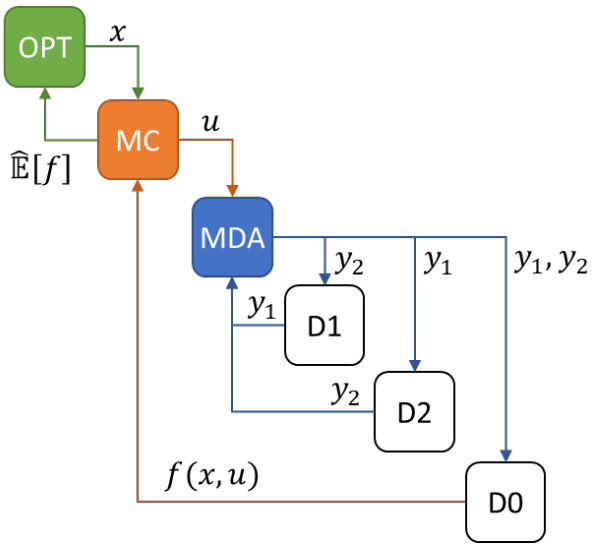

Figure 12: An optimization problem relying on a U-MDO formulation combining MC sampling and an MDF formulation, based on a Jacobi MDA algorithm, with $\widehat{\mathbb{E}}[f(x,U)] = \frac{1}{N}\sum_{i=1}^{N} f(x,u^{(i)})$.

U-MDO problem:

```
from gemseo import configure_logger
from gemseo.algos.parameter_space import ParameterSpace
from gemseo.problems.mdo.sobieski.core.problem import SobieskiProblem
from gemseo.problems.mdo.sobieski.disciplines import SobieskiAerodynamics
from gemseo.problems.mdo.sobieski.disciplines import SobieskiMission
from gemseo.problems.mdo.sobieski.disciplines import SobieskiPropulsion
from gemseo.problems.mdo.sobieski.disciplines import SobieskiStructure

from gemseo_umdo.scenarios.umdo_scenario import UMDOScenario

configure_logger()
```

Firstly, we instantiate the disciplines:

```
mission = SobieskiMission()
structure = SobieskiStructure()
propulsion = SobieskiPropulsion()
aerodynamics = SobieskiAerodynamics()
```

and the design space:

```
design_space = SobieskiProblem().design_space
```

Secondly, we defin the uncertain space:

```
uncertain_space = ParameterSpace()
```

including the uncertain physical property C_4 which is distributed as the triangular distribution $\mathcal{T}(0.85c_4, c_4, 1.15c_4)$ where $c_4 = 0.01375$:

```
uncertain_space.add_random_variable(
    "c_4", "OTTriangularDistribution",
    minimum=0.01375*0.85, mode=0.01375, maximum=0.01375*1.15
)
```

and the uncertain design variable $X_{shared} = dv_{x_{shared}} + U_{x_{shared}}$ where $U_{x_{shared}}$ is distributed as the triangular distribution $\mathcal{T}(-0.15, u_{x_{shared}}, 0.15)$ where $u_{x_{shared}} = 0$:

```
uncertain_space.add_random_variable(
    "u_x_shared", "OTTriangularDistribution",
    minimum=-0.15, mode=0.0, maximum=0.15
)
```

Then, we build a `UMDOScenario` to maximize the expectation of the range $\mathbb{E}[y_4]$ estimated by 50 samples:

```
scenario = UMDOScenario(
    [mission, structure, propulsion, aerodynamics],
    "y_4",
    design_space,
    uncertain_space,
    "Mean",
    formulation_name="MDF",
    # By default, UMDOScenario estimates the statistics by sampling.
    statistic_estimation_parameters={"n_samples": 50},
    maximize_objective=True,
    # There is a convention to make a design variable uncertain.
    uncertain_design_variables={"x_shared": ("+", "u_x_shared")},
)
```

while satisfying margin constraints of the form $\mathbb{E}[g_i] + 2\mathbb{S}[g_i] \leq 0$:

```
scenario.add_constraint("g_1", "Margin")
scenario.add_constraint("g_2", "Margin")
scenario.add_constraint("g_3", "Margin")
```

and execute it with the gradient-based optimizer SLSQP [18]:

```
scenario.execute(algo_name="NLOPT_COBYLA", max_iter=100)
```

The solution is feasible, with an objective equal to 3890.33 nautical miles, which is more conservative than the solution without uncertainties (3963.88).

Lastly, we can plot the optimization history view (see Figure 13) to analyze the convergence of the algorithm:

```
scenario.post_process(post_name="OptHistoryView", save=False, show=True)
```

4.4 Statistics estimation techniques

`gemseo-umdo` can use any type of statistics estimation technique and already offers some standards, such as sampling, surrogate models and variance reduction methods. The choice of the estimation technique applies to all statistics $\mathbb{K}_f$, $\mathbb{K}_g$ and $\mathbb{K}_h$. In the rest of the section, we will limit the mathematical expressions to the function f with scalar uncertain variable U for the sake of clarity.

4.4.1 Sampling

The most naive approach consists of using MC estimators. It is also possible to replace crude MC by a quasi MC approach, such a low-discrepancy sequence or Latin hypercube sampling. `gemseo` offers different implementations of these DOE techniques based on SciPy [22], pyDOE3 and OpenTURNS. The U-MDO formulation *Sampling* can use any of them to generate the input samples $\{u^{(1)}, \ldots, u^{(N)}\}$. Then, the user can choose to compute all the output samples before estimating the statistics, or to estimate the statistics iteratively by using new OpenTURNS features.

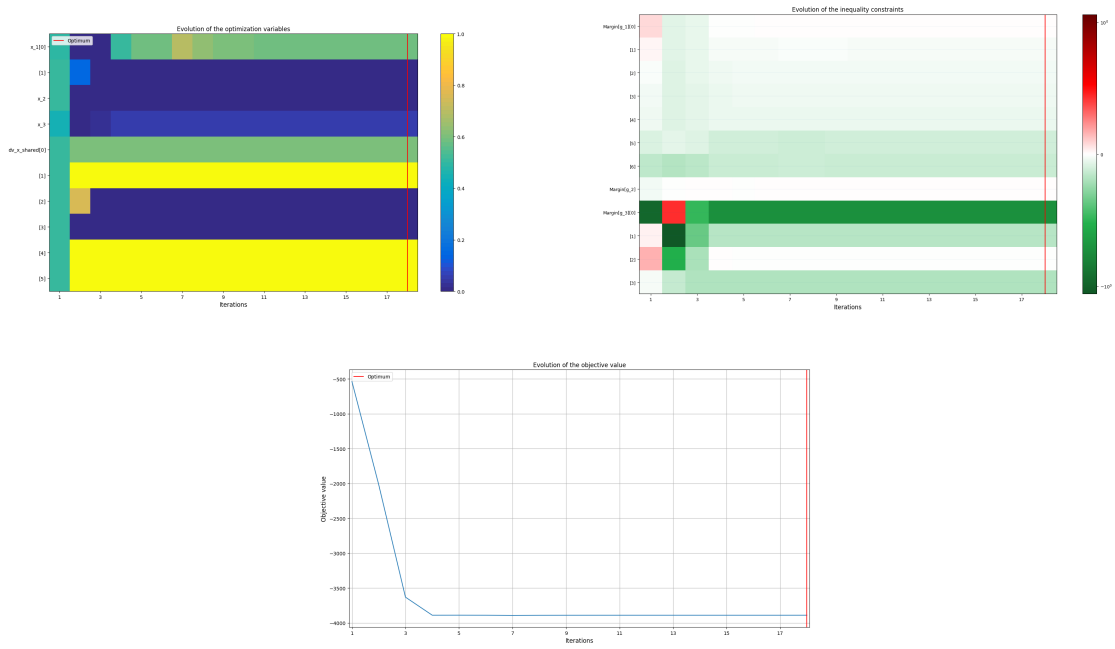

Figure 13: Optimization history view of the SSBJ' s U-MDO problem.

These estimators are often unbiased, which is an interesting property. However, if some certain disciplines are costly to evaluate, the number of samples N will be limited and the variance of the estimators large. This is the main limitation of the sampling techniques.

4.4.2 Taylor polynomial

When evaluating the inner loop of the U-MDO formulation is costly, approximating $u \mapsto f(x,u)$ by a first-orde Taylor polynomial at $\mu = \mathbb{E}[U]$ may be relevant, especially in the case of negligible uncertain sources U:

$$f(x,U) \approx \hat{f}(x,U) = f(x,\mu) + (U-\mu)\partial_u f(x,\mu).$$

Then, the expectation and variance reduce to $f(x,\mu)$ and $\mathbb{V}[U]\left(\partial_u f(x,\mu)\right)^2$ respectively, which requires only one evaluation and one differentiation of f at each iteration of the optimization process. If the partial derivative $\partial_u f$ is missing, it can be approximated by finit differences for example, which results in not 1 but $1+d$ evaluations of f per iteration, where d is the dimension of the design variable x. In low dimension, the U-MDO formulation *TaylorPolynomial* is still cheaper than sampling. The quality of these estimators could be improved if the second-order derivatives can be approximated. On the other hand, it can only be used for statistics combining expectations and variances.

4.4.3 Surrogate models

A Taylor polynomial can be considered as a linear surrogate model. If f is poorly linear with respect to u, the approximation can be quite poor and switching to a non-linear surrogate model can be a good option. `gemseo-umdo` offers the possibility to create a surrogate model of $x \mapsto f(x,u)$ over the uncertain space and estimate the statistics with intensive MC sampling at each iteration of the optimization loop. During the optimization process, information on the

quality of the surrogate models is logged, both in terms of learning and validation, to alert the user in the event of poor approximation.

The U-MDO formulation *Surrogate* allows the user to select any regression model interfaced to `gemseo`. The latter and its extension `gemseo-mlearning` provide access to several models from the libraries scikit-learn, OpenTURNS, SMT and SciPy, such as polynomial regression, Gaussian process regression, radial basis function regression and polynomial chaos expansion (PCE). In the special case of PCE, the U-MDO formulation *PCE* does the same thing, except that it estimates the statistics analytically from the PCE coefficients which limits its use to combinations of expectations and variances (the expectation of a PCE is equal to its offset term it is equal to the sums of the squares of the other coefficients)

4.4.4 Control variates

Most of the time, the surrogate model $\hat{f}$ guarantees that $\frac{1}{N}\sum_{i=1}^{N}\hat{f}(x,U^{(i)})$ is an unbiased estimator of $\mathbb{E}[f(x,U)]$, while it is an interesting statistic estimator property. On the other hand, the estimator $E_N[\hat{f};x]$ may be strongly correlated to $E_N[f;x]$, with $E_N[f;x] = \sum_{i=1}^{N} f(x,U^{(i)})/N$, and thus could be a good control variate (CV) [7]. CV techniques are reduction variance methods to make an MC estimator more accurate. In the case of expectation, the surrogate-based CV estimator is:

$$E_{CV,N}[f;x] = E_N[f,x] - \alpha_N\left(E_N[f;x] - \mathbb{E}[f(x,U)]\right)$$

with $\alpha_N = \frac{C_N[f,\hat{f};x]}{V_N[f;x]}$, where C_N and V_N are the MC estimator of the covariance between $\hat{f}(x,U)$ and $f(x,U)$ and the MC estimator of the variance of $f(x,U)$ respectively. The U-MDO formulation *ControlVariate* can estimate the statistics using surrogate-based control variate techniques. By default, the surrogate model is a first-orde Taylor polynomial.

4.4.5 Summary

Table 1 summarizes the different statistics estimation techniques. It is important to note that even though the sampling was done according to the uncertain space and the PCE was built on this same space, the statistics obtained with the U-MDO formulations *Sampling* and *PCE* are differentiable with respect to the design variables. This means that if the disciplines provide the derivatives and the formulation is differentiable, then you can use a gradient-based optimizer to solve the optimization problem without having to approximate the derivatives. This is particularly promising for using gradient algorithms to converge faster.

4.5 Benchmark problems

We have just seen that there are various statistical estimators, just as there are various optimizers, MDO formulations and MDA algorithms. All combined, it can be complicated to know which is the best combination for you. Faced with such a wealth of combinations, a question arises: which one to choose? Just as there is no free lunch for optimization algorithms, there is no a priori choice for the other components, and one way of doing this is to choose a combination by experience, or to select one based on a benchmark study. The literature on problems to benchmark optimization algorithms is much richer than the literature on problems to benchmark MDO algorithms. This difference is even greater in the case of MDO under uncertainties.

In the face of this reality, `gemseo` proposes different benchmark MDO problems that are scalable, in the sense that the user can chose the number of disciplines, the dimensions of the

	Sampling	Taylor	Surrogate	PCE	ControlVariate
Expectation	✓	✓	✓	✓	✓
Standard Deviation	✓	✓	✓	✓	✓
Variance	✓	✓	✓	✓	✓
Margin	✓	✓	✓	✓	✓
Probability	✓		✓		✓
Differentiable	✓			✓	
Cost	High	Small	Medium	Medium	Medium
Options	Iterative	Second-order	Model Settings Quality measure	Settings Quality measure	Model Settings

Table 1: Summary of the statistics estimation techniques to create a U-MDO formulation. The notion of evaluation cost is subjective and case-dependent. A greater number of U-MDO formulations will be differentiable in the future, especially *Surrogate* and *ControlVariate*.

design variables and the dimension of the coupling variables. In particular, it proposes an MDO problem in which the mathematical characteristics of disciplines reflec those of user disciplines, thanks to frugal sampling [29]. An extension of the scalable quadratic MDO problem proposed by [27] is also implemented, in which the user can defin uncertainties in the coupling equations in particular, as well as the percentage of feasible design space [2]. In addition, classic MDO problems are implemented, such as Sellar' s problem [25], supersonic business jet (SSBJ) problem [26] and propane combustion problem [27]. It is easy to extend them to the uncertain case, by replacing certain constants with random variables or considering some design variables as uncertain, as illustrated in Section 4.3.

Lastly, `gemseo` offers the `OptAsMDOScenario` class, which is an `MDOScenario` making a monodisciplinary optimization problem multi-disciplinary, by applying a method proposed by the authors of [2]. The original discipline must have as inputs at least three design variables and as outputs one or more objectives. It may also have as outputs one or more constraints. The MDO problem will include $2 + N$ disciplines, namely

- N strongly coupled disciplines computing the values of some coupling variables from the values of the other coupling variables and the values of the design variables,
- one weakly coupled discipline, called *link discipline*, computing the values of the design variables in the original optimization problem from the values of the design and coupling variables in the MDO problem,
- the original discipline.

By default, the strongly coupled disciplines and the link discipline are linear and the user can pass its own disciplines. For example, `gemseo-umdo` extends the `OptAsMDOScenario` class into `UOptAsUMDOScenario`, to make an optimization problem under uncertainty multi-disciplinary. This approach allowed us to rewrite the Rosenbrock optimization problem [21] in dimension 3 as a U-MDO problem with linear coupling equations:

```
discipline = AnalyticDiscipline(
    {"f": "100*(z_2-(u*z_1)**2)**2+(1-v*z_1)**2+100*(z_1-(u*z_0)**2)**2+(1-
   v*z_0)**2"},
    name="Rosenbrock",
```

```
)

design_space = DesignSpace()
design_space.add_variable("z_0", lower_bound=-1, upper_bound=1)
design_space.add_variable("z_1", lower_bound=-1, upper_bound=1)
design_space.add_variable("z_2", lower_bound=-1, upper_bound=1)

uncertain_space = DesignSpace()
uncertain_space.add_random_variable(
    "u", "OTNormalDistribution", mu=1., sigma=0.01
)
uncertain_space.add_random_variable(
    "v", "OTNormalDistribution", mu=1., sigma=0.01
)

scenario = UOptAsUMDOScenario(
    discipline,
    "f",
    design_space,
    uncertain_space,
    "Mean",
    formulation_name="MDF"
)
scenario.execute(algo_name="NLOPT_SLSQP", max_iter=100)
```

Figure 14 shows the discipline coupling graph associated with this problem, and the section ends with the start of the log associated with this resolution. You can notice that the optimization variables are renamed into x_0, x_1, x_2 because the variables z_0, z_1, z_2 are now coupling variables.

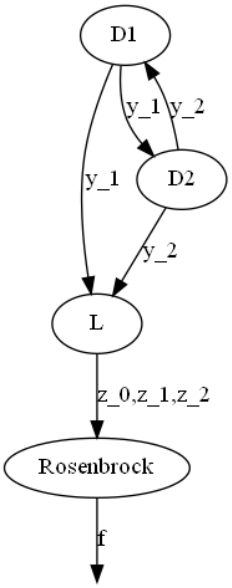

Figure 14: Discipline coupling graph in a multidisciplinary of the Rosenbrock problem under uncertainty, with the two strongly coupled discipline D_1 and D_2, the link discipline L and the standard Rosenbrock function.

```
INFO - 16:43:14: *** Start OptAsMDOScenario execution ***
INFO - 16:43:14: OptAsMDOScenario
```

```
INFO - 16:43:14:    Disciplines: D1 D2 L Rosenbrock
INFO - 16:43:14:    Formulation:
INFO - 16:43:14:       MDO formulation: MDF
INFO - 16:43:14:       Statistic estimation: Sampling
INFO - 16:43:14:    Uncertain space:
INFO - 16:43:14:       +------+---------------------------+
INFO - 16:43:14:       | Name |        Distribution       |
INFO - 16:43:14:       +------+---------------------------+
INFO - 16:43:14:       |  u   | Normal(mu=1.0, sigma=0.01) |
INFO - 16:43:14:       +------+---------------------------+
INFO - 16:43:14:       |  v   | Normal(mu=1.0, sigma=0.01) |
INFO - 16:43:14:       +------+---------------------------+
INFO - 16:43:14: Optimization problem:
INFO - 16:43:14:    minimize E[f(x_0, x_1, x_2)]
INFO - 16:43:14:    with respect to x_0, x_1, x_2
INFO - 16:43:14:    over the design space:
INFO - 16:43:14:       +------+-------------+-------+-------------+-------+
INFO - 16:43:14:       | Name | Lower bound | Value | Upper bound | Type  |
INFO - 16:43:14:       +------+-------------+-------+-------------+-------+
INFO - 16:43:14:       | x_0  |      -1     | -0.25 |      1      | float |
INFO - 16:43:14:       | x_1  |      -1     |  0.75 |      1      | float |
INFO - 16:43:14:       | x_2  |      -1     |  -0.9 |      1      | float |
INFO - 16:43:14:       +------+-------------+-------+-------------+-------+
INFO - 16:43:14: Solving optimization problem with algorithm NLOPT_SLSQP:
...
```

5 CONCLUSIONS

In this work, we have presented the extension of `gemseo` to UQ features, facilitated by the object-oriented paradigm intensively used.

First, we introduced the basic classes on which the UQ features rely. It was an opportunity to demonstrate the versatility of `gemseo`, capable to automate the creation of evaluation problems from user-friendly disciplines define by data labels and an MDO formulation that define the computational process.

Then, we introduced how to use `gemseo` and its extension `gemseo-umdo` to propagate uncertainties through a multi-disciplinary process and estimate the quantities of interest either empirically or parametrically. Graphical techniques were presented for uncertainty visualization in coupling graphs, in order to detect the paths of greatest variability.

We have also presented the sensitivity analysis techniques that separate the data generation process from the sensitivity indices estimation. This allows to save the samples and calculate the indices or perfect the graphs later, without having to resample the multi-disciplinary system. We have shown that it is possible to extend such techniques, e.g. the Morris method with new DOE algorithm to place the trajectories.

We also introduced the active learning capabilities available in `gemseo-mlearning`, with a focus on quantile estimation. This features can easily been extended to new machine learning libraries to create the surrogate models (typically Gaussian process regressors) [5] and new acquisition criteria.

Thirdly, we have tackled the extension of the MDO framework to account for uncertainties. To do this, we have shown how the new concepts are based on GEMSEO's basic classes, once again proving its genericity, and presented different estimation techniques for the statistics of objective and constraints. These features were illustrated on an academic use case from the

aeronautical litterature and the section concludes with benchmarks problems to compare U-MDO algorithms, combining both estimation techniques and MDO formulations.

In this work, we have also discussed the promising use of surrogate-based control variates to estimate the statistics more precisely, for whether Sobol' analysis or U-MDO problems, and highlighted that a surrogate can be a simple and cheap first-orde Taylor polynomial.

In the future, we would like to illustrate the application of the U-MDO formulation engine with a wide selection of MDO formulations, including IDF and bi-level ones. In particular, we will show how `gemseo-umdo` can implement the bi-level U-MDO formulation proposed by [1] and inspired by the uncertainty-free bi-level formulation [9]. This U-MDO formulation returns both the optimum of the global design variables as fi ed values and the optimum of the local variables as random variables. The latter depends on the sources of uncertainty that we hope will decrease during the design cycle and thus lead to better solutions. We can also propose a bi-level U-MDO formulation that seeks to optimize the local and global design variables at the same time. We also plan to increase the use of surrogate models in the UQ context, either trained offlin or online for active learning purposes. In this context, new use of gradient-based techniques would fin its place in U-MDO, to improve convergence and accuracy of optimization processes, e.g. gradient-enhanced and differentiable surrogate models. Bridging the gap between active learning techniques and U-MDO would also be a logical extension of the work done in GEMSEO. Lastly, there is an ongoing effort to improve interfaces and documentation, advise on the choice of U-MDO formulations depending on the computational budget and the properties of the problem, and thus democratize access to MDO techniques under uncertainty.

ACKNOWLEDGEMENTS

This study has been supported by the NEXTAIR project which has received funding from the European Union's Horizon Europe research and innovation program under grant agreement No 101056732. Views and opinions expressed are however those of the authors only and do not necessarily reflec those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

REFERENCES

[1] Aziz-Alaoui, A., *Contributions to multidisciplinary design optimization under uncertainty, with applications to aircraft design*, Thèse de doctorat, Université de Toulouse, 2025.

[2] Aziz-Alaoui, A., Roustant, O. and De Lozzo, M. *A scalable problem to benchmark robust multi-disciplinary design optimization techniques*. Optim Eng 25, 941–958, 2024.

[3] Baudin, M., Dutfoy, A., Iooss, B., Popelin, AL, *OpenTURNS: An Industrial Software for Uncertainty Quantification in Simulation*. In: Ghanem, R., Higdon, D., Owhadi, H. (eds) Handbook of Uncertainty Quantification Springer, Cham, 2015.

[4] Ben Salem, M., Roustant, O., Gamboa, F., and Tomaso, L., *Universal prediction distribution for surrogate models*. SIAM/ASA Journal on Uncertainty Quantification 5:, 12 2015

[5] Bouhlel, M. A., and Hwang, J. T., and Bartoli, N., and Lafage, R., and Morlier, J., and Martins, J. R. R. A., *A Python surrogate modeling framework with derivatives*. Advances in Engineering Software, 2019.

[6] Brevault, L. and Balesdent, L. *Uncertainty-Based multi-disciplinary Design Optimization (UMDO)*. In: Aerospace System Analysis and Optimization in Uncertainty. Springer Optimization and Its Applications, **156**, Springer, Cham, 2020.

[7] El Amri, M. R., Mycek, P., Ricci, S., and De Lozzo, M., *Multilevel surrogate-based control variates*. arXiv preprint arXiv:2306.10800.

[8] Gallard, F., Vanaret, C., Guénot, D, et al., *GEMS: A Python Library for Automation of multi-disciplinary Design Optimization Process Generation*. In : 2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, 2018.

[9] Gazaix, A., et al, *Industrial Application of an Advanced Bi-level MDO Formulation to Aircraft Engine Pylon Optimization*. In : AIAA Aviation 2019 Forum, 2019.

[10] Hunter, J. D., *Matplotlib: A 2D Graphics Environment*. Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007.

[11] Ishigami, T., and Homma, T, *An importance quantification technique in uncertainty analysis for computer models*. In First International Symposium on Uncertainty Modeling and Analysis, 1990.

[12] Jones, D., Schonlau, M., and Welch, W., *Efficient global optimization of expensive black-box functions*. Journal of Global optimization, 13:455–492, 1998.

[13] Laboulfie C., De Lozzo, M., Grotto, F., Barrière, L., Navarro, J.-P., Bocquet, S., *Parallel Active Learning for Quantile Estimation from Composites Damage Models*.

[14] Martins, J. R. R. A., and Lambe, A. B., *multi-disciplinary Design Optimization: A Survey of Architectures*. AIAA Journal Vol. 51 (9) , 2049-2075, 2013.

[15] Morris, M.D., *Factorial Sampling Plans for Preliminary Computational Experiments*, Technometrics. 33 (2): 161–174, 1991.

[16] Muehlenstaedt, T., Roustant, R., Carraro, L., and Kuhnt, S., *Data-driven Kriging models based on FANOVA-decomposition*. Statistics and Computing, 22:723-738, 2012.

[17] Pedregosa et al., *Scikit-learn: Machine Learning in Python*, JMLR 12, pp. 2825-2830, 2011.

[18] Powell, M.J.D., *A direct search optimization method that models the objective and constraint functions by linear interpolation*. in Advances in Optimization and Numerical Analysis, eds. S. Gomez and J-P Hennart, Kluwer Academic (Dordrecht), pp. 51-67, 1994.

[19] Ranjan, P., Bingham, D., and Michailidis, G., *Sequential experiment design for contour estimation from complex computer codes*. Technometrics, 50(4):527–541, 2008.

[20] de Rocquigny, E., Devictor, N., and Tarantola, S. (Eds.), *Uncertainty in industrial practice: a guide to quantitative uncertainty management*. John Wiley & Sons, 2008.

[21] Rosenbrock, H.H., *An automatic method for finding the greatest or least value of a function*. The Computer Journal. 3 (3): 175–184, 1960.

[22] Roy et al, *Quasi-Monte Carlo Methods in Python*. Journal of Open Source Software, 8(84), 5309, 2023.

[23] Roy, S., and Notz, W., *Estimating percentiles in computer experiments: a comparison of sequential-adaptive designs and fixed designs*.Journal of Statistical Theory and Practice, Springer, 8(1), 2014

[24] Saltelli , A., Annoni, P., Azzini, I., Campolongo, F., Ratto, R. and Tarantola, S., *Variance based sensitivity analysis of model output. design and estimator for the total sensitivity index*. Computer physics communications, 181(2):259–270, 2010.

[25] Sellar , R., Batill, S., and Renaud, J., *Response surface based, concurrent subspace optimization for multi-disciplinary system design*. 34th Aerospace Sciences Meeting and Exhibit, 1996.

[26] Sobieszczanski-Sobieski, J., Agte, J. S., and Sandusky, R. R. S., Jr., *Bi-Level Integrated System Synthesis (BLISS)*. Tech. Rep. TM-1998-208715, NASA, Langley Research Center, 1998.

[27] Tedford, N., and Martins, J. R. R. A., *Benchmarking multi-disciplinary design optimization algorithms*. Optimization and Engineering 11, 159–183, 2010.

[28] The pandas development team, 10.5281/zenodo.3509134, 2020.

[29] Vanaret, C., Gallard, F., and Martins, J. R. R. A., *On the Consequences of the "No Free Lunch" Theorem for Optimization on the Choice of an Appropriate MDO Architecture*. 18th AIAA/ISSMO multi-disciplinary Analysis and Optimization Conference, 2017.