TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Civil Engineering Institute for Structural Analysis

Diploma Thesis

FUZZY-ANALYSIS IN A GENERIC POLYMORPHIC UNCERTAINTY QUANTIFICATION FRAMEWORK

Bertram Richter

Institute for **Structural Analysis**

Diploma Thesis

Fuzzy-Analysis in a Generic Polymorphic Uncertainty Quantification Framework

Fuzzy-Analyse als Teil generischer polymorpher Unschärfe-Analyse

Bertram Richter

Supervised by:
Prof. Dr.-Ing. Wolfgang Graf, Univ.-Prof. Dr.-Ing. habil. Michael Kaliske

and

Dipl.-Ing. F. Niklas Schietzold

Submitted on 2021-05-12

TECHNISCHE UNIVERSITÄT DRESDEN

Institut für **Statik & Dynamik** der Tragwerke

Fakultät Bauingenieurwesen Institut für Statik und Dynamik der Tragwerke, Univ.-Prof. Dr.-Ing. habil. Michael Kaliske

Aufgabenstellung für die Diplomarbeit DA04/20

Name:	Bertram Richter
Vertiefung:	Konstruktiver Ingenieurbau (KI)

Thema: **Fuzzy-Analyse als Teil generischer polymorpher Unschärfe-Analyse**
(Fuzzy-Analysis in a Generic Polymorphic Uncertainty Quantification Framework)

Hintergrund und Zielsetzung:

Fuzzy-Größen sind Unschärfemodellierungen zur Beschreibung insbesondere epistemischer Unschärfecharakteristika, wie Unvollständigkeit und Ungewissheit. Die Entwicklung effizienter Methoden zur numerischen Unschärfe-Analyse von Fuzzy-Größen ist Teil aktueller Forschung. Dabei werden neben der Methode der α-Level Optimierung auch andere Strategien verfolgt.

Polymorphe Unschärfemodelle ermöglichen die Berücksichtigung sowohl epistemischer als auch aleatorischer Unschärfecharakteristika. Die zur numerischen Auswertung notwendige polymorphe Unschärfeanalyse ist eine Kombination aus stochastischen und fuzzy-basierten Analyse-Methoden.

Ziel dieser Arbeit ist es, aktuelle Methoden der Fuzzy-Analyse zu untersuchen und zu vergleichen. Dabei werden die Fuzzy-Analyse Algorithmen als Teil eines generisch geordneten Software-Rahmens implementiert, welcher für die Anwendung auf polymorphe Unschärfe-Analysen im Kontext von Strukturanalysen angelegt ist. Der Entwurf und die Implementation der Basis einer Datenstruktur für die polymorphe Unschärfe-Analyse erfolgt in enger Abstimmung und Zusammenarbeit mit der Diplomarbeit DA03/20 „Stochastische-Analyse als Teil generischer polymorpher Unschärfe-Analyse"

Die Ansätze der *state-of-the-art* Methoden zur Fuzzy-Analyse werden gegenübergestellt und daraus kombinierte Strategien werden entwickelt. Besonderer Fokus der Arbeit liegt auf der Stukturierung der Daten und der Analyse von Zusammenhängen zwischen Fuzzy-Ergebnisgrößen.

Arbeitsschritte:

1. Literaturrecherche zu den Themen: Polymorphe Unschärfeanalyse, Fuzzy-Analyse Methoden
2. Entwurf Datenstruktur und Rahmen für generische polymorphe Unschärfe-Analyse
3. Implementation und Vergleich der Ansätze zur numerischen Fuzzy-Analyse
4. Kombination der Ansätze zur Verbesserung der Effizienz von Fuzzy-Analysen
5. Implementation von Methoden zur Analyse von Abhängigkeiten zwischen Fuzzy-Ergebnisgrößen
6. Implementation von Darstellungsmöglichkeiten für die Präsentation von Analyse-Ergebnissen unter Einsatz geeigneter Benchmark-Beispiele.

Wiss. BetreuerInnen TU Dresden: Dipl.-Ing. F. Niklas Schietzold

ausgehändigt am:
einzureichen am:

Prof. Dr.-Ing. Wolfgang Graf	Prof. Dr.-Ing. habil. Michael Kaliske
Institut für Statik und Dynamik der Tragwerke	Institut für Statik und Dynamik der Tragwerke
Verantwortlicher Hochschullehrer	Verantwortlicher Hochschullehrer

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

Die besonderen Hinweise des Instituts für die Anfertigung der Diplomarbeit sind zu beachten.

Declaration of originality

I confirm that this assignment is my own work and that I have not sought or used inadmissible help of third parties to produce this work and that I have clearly referenced all sources used in the work. I have fully referenced and used inverted commas for all text directly or indirectly quoted from a source.

This work has not yet been submitted to another examination institution – neither in Germany nor outside Germany – neither in the same nor in a similar way and has not yet been published.

Dresden, 2021-05-12

Abstract

In this thesis, a framework for generic uncertainty analysis is developed. The two basic uncertainty characteristics aleatoric and epistemic uncertainty are differentiated. Polymorphic uncertainty as the combination of these two characteristics is discussed. The main focus is on epistemic uncertainty, with fuzziness as an uncertainty model. Properties and classes of fuzzy quantities are discussed. Some information reduction measures to reduce a fuzzy quantity to a characteristic value, are briefly debated. Analysis approaches for aleatoric, epistemic and polymorphic uncertainty are discussed. For fuzzy analysis α-level-based and α-level-free methods are described. As a hybridization of both methods, non-flat α-level-optimization is proposed.

For numerical uncertainty analysis, the framework PUQpy, which stands for "Polymorphic Uncertainty Quantification in Python" is introduced. The conception, structure, data structure, modules and design principles of PUQpy are documented. Sequential Weighted Sampling (*SWS*) is presented as an optimization algorithm for general purpose optimization, as well as for fuzzy analysis. Slice Sampling as a component of *SWS* is shown. Routines to update Pareto-fronts, which are required for optimization are benchmarked.

Finally, PUQpy is used to analyze example problems as a proof of concept. In those problems analytical functions with uncertain parameters, characterized by fuzzy and polymorphic uncertainty, are examined.

Acknowledgments

Firstly, I would like to thank my parents for their support and the assuredness, that this support is never-ending. I am glad and grateful for the comfortable study, that they made possible for me. I highly appreciate the caring interest and reassurance in me and the sometimes necessary push, which I have gotten from them.

Secondly, I want to thank Marcus Seidowski, my mate in this project, as he wrote the sibling to this work, thus being tightly involved in the matter. He endured sheer endless discussions with me regarding nearly every little aspect of the project, even and especially when it was way out of the scope of his work. I apologize for the harsh critique, that he had received sometimes, and thankful for the critique given back. Without him, this project would have never gotten off the ground.

I want to thank my supervisors F. Niklas Schietzold and Ferenc Leichsenring for helping me solving problems I could not wrap my head around by myself. Sometimes those problems existed in my head only. For every question I asked, an answer was found eventually, no matter, how difficult or stupid it has been, but all of them have been tiring for sure.

I want to thank my professors Wolfgang Graf and Michael Kaliske for the introduction to uncertainty, which I got in the lectures. It lighted the spark for my enthusiasm for uncertainty, leading directly to this diploma thesis, close to the current state of research. It is a high honor to me, to be able to contribute to the institutes work.

I would like to thank the staff of the Institute for Structural Analysis for the general support, tips and tricks I have gotten and the opportunity to write my diploma here.

Contents

1 Introduction

In engineering tasks, consideration of uncertainty is required for a realistic behavior assessment of the designed structures. Both safety, as well as efficiency of structures can be increased by accounting for uncertainties using appropriate uncertainty analysis approaches. By realistic modeling of uncertain quantities in the design process, risks to the safety of the designed structure can be estimated more exact, resulting is a safer design. Based on those more detailed risk assessments, potential overdimensioning by constructing on the conservative side can be avoided, reducing costs. Therefore the importance of epistemic uncertainty in engineering models and its incorporation into computations is rising. With fuzzy quantities, introduced by [100], uncertainty is modeled on a gradual scale of truth and belongingness. For dealing with aleatoric uncertainty, there is a wide variety of stochastical and statistical software. In comparison, available software, with which epistemic uncertainty can be dealt with, is rare. The combination of both uncertainty characteristics, polymorphic uncertainty, is lacking available software. Hence, in this work, the software PUQpy, which stands for "Polymorphic Uncertainty Quantification in Python" is introduced.

Theoretical aspects of this thesis are discussed in Chapter 2. In Section 2.1, uncertainty characteristics are differentiated. Aleatoric and epistemic uncertainty are explained as basic uncertainty models, polymorphic uncertainty is presented as the combination of aleatoric and epistemic uncertainty. Definitions of fuzzy quantities are given in Section 2.2. The basics of fuzziness are discussed, secondly several types of fuzzy quantities with their properties are presented. Methods for defuzzification are given in Section 2.2.6. These are used to reduce fuzzy quantities to characteristic values. In Section 2.3, approaches to analysis of fuzziness are discussed. As the two approaches to fuzzy uncertainty analysis, α-level-based, see Section 2.3.1, and α-level-free methods, see Section 2.3.2, are discussed. In Section 2.4, qualities of optimization are discussed. Pareto-fronts are explained and the performance of update methods for Pareto-fronts is measured and compared. The numerical algorithms used in this thesis are explained in detail in Section 2.5.

In Chapter 3, the objective and general structure of PUQpy is presented. The purpose of the implemented classes `Analysis` and sub-classes, `FundamentalSolution`, `Layer`, and `UncertaintyAnalysis` for uncertainty analysis explained in-depth. As a central design element of PUQpy, the connection of `Analysis` objects by using `Layer` objects is elaborated.

In Chapter 4, numerical examples with fuzzy input quantities and polymorphic input quantities are given as a proof of concept and usability of PUQpy. Finally, the results of the work are discussed in Chapter 5 and questions to be answered in further research are pointed out.

2 Theory

2.1 Uncertainty Models

As summarized in [71], there are two basic uncertainty models, aleatoric and epistemic uncertainty. Aleatoric uncertainty describes variability, that is caused by true random or uncontrollable processes, and features probabilistic, that is random, properties. Since this variability is inherent to the quantity, aleatoric uncertainty is irreducible [71].

Epistemic uncertainty is present, when dealing with limited data, vague knowledge, incertitudes, and impreciseness of the model, or data. According to [71] epistemic uncertainty can be reduced at least in theoretically by gathering more data, or less imprecise data, or by using less simplified models. In practice however, this may be difficult to impossible.

A quantity, which has either aleatoric or epistemic properties is called a monomorphic quantity [19; 35, p. 3]. Polymorphic quantities feature properties of both aleatoric and epistemic uncertainty [35, p. 3; 68, p. 1; 77, p. 9]. Thus, they can be used to account for both randomness and fuzziness at the same time. Every polymorphic quantity can be decomposed into a set of monomorphic quantities and their dependencies. Therefore, it is possible to assemble arbitrarily complex quantities, and the appropriate analysis methods from monomorphic ones.

The uncertainty model and the analysis approach is to be chosen depending on the available quantity and quality data [87]. If only low quantity or low quality data is available, an epistemic or polymorphic analysis approach may be better suited. According to [32; 33; 47; 51], a purely stochastic analysis approach to uncertainty may not be appropriate, since non-probabilistic uncertainty may occur in stochastic quantities due to

- uncertain distribution type,
- uncertain parameters of the distribution, due to statistical uncertainty,
- bias in the recorded data due to survey design,
- uncertain limit-state function.

Therefore, in the following discussion of different models, appropriateness in regard to the data situation is given to help choosing the right one for the use case. Firstly, in Section 2.1.2 and Section 2.1.1, analyses for monomorphic uncertainty are explained. Then, in Section 2.1.3 different types of polymorphic uncertainty are discussed.

2.1.1 Aleatoric Uncertainty

Aleatoric uncertainty can be modeled with stochastic quantities [34, p. 1; 86, p. 13]. For modeling and analysis of aleatoric uncertainty, software is available. Some examples of available software packages are:

- SCIPY, see [38; 99],
- UQPY, see [83],
- OPENTURNS, see [7] and
- OPENCOSSAN, see [71].

Stochastic Analysis Aleatoric or probabilistic uncertainty, shows random properties and is modeled with stochastic quantities [34, p. 1]. In case, that an event as the result of a random trial is almost always a crisp value, under an unlimited number of occasions, with constant boundary conditions, it is a pure stochastic quantity. Stochastic quantities follow the law of large numbers, according to which the relative frequency of an event approaches its probability [23, pp. 290 sqq.]. For this prerequisite to hold true, samples must be pairwise independent, identically distributed (*iid*) [98, p. 63]. If the boundary conditions are not constant, the *iid*-criterion is violated, or only a limited amount of events is available, the law of large numbers is violated, or the outcome is not crisp, it features additional uncertainty. Thus, the quantity is not a pure stochastic, but a polymorphic quantity. It is bad practice to ignore that fact and model it as a stochastic quantity nevertheless. In this case, it is advisable to use another uncertainty analysis, such as fuzziness or polymorphic uncertainty. [84, p. 379]

BAYESian Uncertainty The BAYESian approach, as introduced in [8], is used, when modeling the properties of a stochastic quantity as stochastic quantities. A realization of the stochastic quantity is a stochastic quantity again. In statistical hypothesis testing, the validity of a stochastic model is determined by means of probability, since the population of experimentees is a subset of the statistical universe shows random variation [23, pp. 369 sqq.].

2.1.2 Epistemic Uncertainty

Epistemic uncertainty can be modeled using the uncertainty model fuzziness [34, p. 1; 31, p. 1]. According to [34; 50] epistemic uncertainty may originate in

- linguistic uncertainty,
- missing data due to unknown data or data loss,
- imprecise and inaccurate data, for example due to tolerances or measure errors,
- imprecise or inaccurate functional models,
- expertise or
- if reasons for variance are unknown.

Fuzziness

Classes in the real world do not have precisely, and sharply defined parameters, as postulated in [100]. Thus, most objects are not binarily assigned to classes in "does belong" and "does not belong" to the class, but rather belong to a point in a continuum between these two extremes. Human thinking often is based on vague properties, especially, in pattern recognition to account for possible deviation from the anticipation, communication to give room for interpretation, and unforeseeable events, and abstraction [100]. The purpose of fuzzy quantities is to describe and quantify vagueness. An introduction to the theory of fuzziness is given in [37]. In [98], general, including non-convex, fuzzy numbers are discussed, and it is focused on the incorporation of fuzzy and stochastic analyses. Fuzzy quantities are discussed in Section 2.2.

Fuzzy-based Fuzziness (*ff*)

Fuzzy-based fuzzy quantities have two possible use cases. Firstly, if the assessment of uncertainty itself is doubtful. The properties of a fuzzy quantity are modeled as fuzzy quantities, thus a realization of the outer fuzzy quantity is a fuzzy quantity again. This is the case when reading a membership function off a plot or if limits of the quantity are only vaguely available.

Second use case is the examination of epistemic uncertainty, when different uncertainty sources are considered [68]. The resulting membership function of a single, even multivariate fuzzy analysis describes the combined uncertainty sensitivity of all fuzzy input quantities. On the basis of α-levels, it can be understood how the uncertainty of the result is reduced by reducing the uncertainty of all uncertain input quantities to this specific α-level. An insight on the influence of the individual quantities is not easily possible. By dividing the design space in sub-spaces, the objective space is divided too, as shown in [68]. Mapping the components from input to output spaces, makes it possible to isolate the uncertainty influence of several input quantities on the result quantity from each other. Therefore it is possible to see the uncertainty sensitivity for each of the quantities separately. The user is enabled to identify the quantity, that imposes the highest uncertainty onto the result. This insight can be harnessed to reduce the result's overall uncertainty by conducting further targeted surveys. In this approach the quantities are put into individual analyses, but are independent of each other.

2.1.3 Polymorphic Uncertainty

Fuzzy Probability Based Randomness (*fp-r*) Fuzzy probability based randomness (*fp-r*) is used, if data is sparse and the estimates for the stochastic parameters cannot be considered to be accurate, see [30, p. 39]. The properties of a stochastic quantity are modeled as fuzzy quantities. A realization of the fuzzy quantity is a stochastic quantity again, thus a certain trajectory in the bunch of distribution functions. This is the case for experiments, where only a few samples can be taken. Modeling as pure stochastic is not permissible, since the law of large numbers cannot be considered fulfilled.

Fuzzy Randomness (*fr*) Fuzzy randomness (*fr*) is used, if data base is sufficient, but each sample shows impreciseness in itself. Thus, the data cannot be trusted, see [30, p. 39]. The properties of a fuzzy quantity are modeled as stochastic quantities. A realization of the stochastic quantity is a fuzzy quantity again. This may be the case, in experiments, where a lot of data has been measured hastily.

Fuzzy Probability Based Fuzzy Randomness (*fp-fr*) Fuzzy probability based fuzzy randomness (*fp-fr*) is to be used, if data is scarce, yet imprecise and involvement of randomness is assumed, see [30, p. 39]. Therefore, the properties of the stochastic quantity are doubtful, thus modeled as fuzzy quantities. The outcome of a single trial itself is yet uncertain again, thus a fuzzy quantity. Three quantities are necessary. The most inner fuzzy quantity models the uncertainty of a single trial. Its properties are determined by the means of a stochastic quantity. The properties of the stochastic quantity are yet again uncertain and modeled with a fuzzy quantity. All previous cases are covered as special cases in *fp-fr*.

2.2 Fuzzy Quantities

2.2.1 Membership function, α-Cut and α-Level

A fuzzy quantity X^{f} is a set whose members x are assessed by their *possibility*, or *membership* value $\mu(x)$ [34, p. 1; 98]. The *membership* indicates, how much an item belongs to a group or set, see Example 2.1.

> **Example 2.1:** The perception of temperature is highly subjective. Depending on a lot of factors, a person may rate temperatures by their comfort. The comfort of a person at a given temperature is somewhere on the gradual spectrum between perfectly fine, and life-threatening unbearable. The assessment process, of comfort for specific temperatures is the evaluation of the membership function.

The term *possibility* is a measure to express the degree of truth of a statement or the degree to which an event may occur [20, p. 2]. Possibility theory is the epistemic counterpart to probability theory [22; 39; 50]. Since in this thesis fuzzy quantities are understood as a generalization of the classical set theory, the term *membership* is used rather than the term *possibility*.

Let be $\mathbb{X}$ a space and $\mathcal{F}(\mathbb{X})$ the class of fuzzy sets in $\mathbb{X}$, then $X^{\mathrm{f}} \in \mathcal{F}(\mathbb{X})$ is a fuzzy quantity [88]. The function $\mu_{X^{\mathrm{f}}} : \mathbb{X} \to [0, 1]$ maps the degree of membership of the element $x \in \mathbb{X}$ to the unity interval $[0, 1]$, see [66, p. 1]. Therefore, $\mu_{X^{\mathrm{f}}}$ is referred to as the membership function of a fuzzy quantity X^{f}. The short notation $\mu(x)$ the quantities name X^{f} is omitted.

An α-level is defined as the set for whose members the membership is at least the specified value α [30, p. 31; 89, p. 5].

> **Example 2.2:** For the assessment of comfort, see Example 2.1, some zones based on the level of comfort may be defined, like the livable zone, bearable zone, workable zone, comfort zone, and the perfect temperature zone. The ranges of those comfort zones can be comprehended as α-levels.

An α-level is obtained by the inverse function to μ, the α-cut or α-level-cut. In literature α-cut and α-level are sometimes used synonymously. In this thesis α-cut denotes the method of discretization, yielding a set of intervals, the α-levels. The weak α-cut is defined as

$$C_\alpha = \{x \in \mathbb{X} \mid \mu(x) \geq \alpha\} \ : \ \alpha \in \,]0, 1] \,. \tag{2.1}$$

If the membership is strictly greater than the specified α,

$$C_\alpha = \{x \in \mathbb{X} \mid \mu(x) > \alpha\} \ : \ \alpha \in \,]0, 1] \tag{2.2}$$

is called the strong α-cut [46; 37, p. 19]. In this work, the weak α-cut is used, if not stated otherwise.

In this thesis, only convex, normalized, and bounded quantities are discussed. For convex fuzzy quantities, each α-level-set fulfills the inclusion property

$$C_{\alpha_k} \subseteq C_{\alpha_i} \,\forall\, \alpha_i \leq \alpha_k \ : \ \alpha_i, \alpha_k \in \,]0, 1] \,, \tag{2.3}$$

which means, that every α-level is contained in all lower α-levels [89, p. 5]. The set $C_0(X^{\mathrm{f}}) = \{x \in X^{\mathrm{f}} \mid \mu(x) > 0\}$ is referred to as *support*. For the numerical representation in this thesis, the *support* is a closed interval, thus the very limits, for which $\lim \mu(x) = 0$, are considered members of the *support*. According to [46], the set of members with the highest membership in a quantity is referred to as *kernel*. In [73, p. 11], a normalized fuzzy quantity $X^{\mathrm{f}}_{\mathrm{norm}}$ is defined by the restriction

$$\sup \mu_{X^{\mathrm{f}}_{\mathrm{norm}}}(x) = 1 \Leftrightarrow C_1(X^{\mathrm{f}}_{\mathrm{norm}}) \neq \emptyset \wedge \nexists\, x \in X^{\mathrm{f}}_{\mathrm{norm}} \;:\; \mu(x) > 1. \tag{2.4}$$

The closed set $C_1(X^{\mathrm{f}}) = \{x \in X^{\mathrm{f}} \mid \mu(x) = 1\}$ is called *core* of a fuzzy quantity. For normalized fuzzy quantities *kernel*, and *core* are identical, and the latter term will be used in this thesis. Which means, that the core must not be empty, thus at least one x for which $\mu(x) = 1$ holds, and the maximum membership is limited to 1. For bounded fuzzy quantities, the support C_0 is bounded. Thus, all α-levels of such a fuzzy quantity are compact, closed, and bounded sets, see [89, p. 1].

Fuzzy quantities can be subclassified into analytical and empirical fuzzy quantities. Analytical fuzzy quantities are represented by their analytical membership function [37, p. 15], see Section 2.2.4. Among these, they can be defined in different ways. The membership can be given by a piece wise defined function [49] or by a single closed function over the complete space. For a family of functions, characteristic values, for example support, core, or a moment can be used for parameterization of the membership function [27; 49]. In [21, p. 618] L-R fuzzy numbers are introduced as a representation by a parameterized function family for the left and right slope of the fuzzy quantity. Another possible representation of a fuzzy quantity is a set of stacked, assessed intervals, which are α-levels and therefore referred to as α-level-based fuzzy quantities.

Empirical fuzzy quantities consist of a set of elements and their respective membership values [37, p. 15]. Those elements can live in any space, also in non-continuous spaces, but $\mathbb{R}^n$ is focus of this thesis. Empirical fuzzy are discussed in Section 2.2.5.

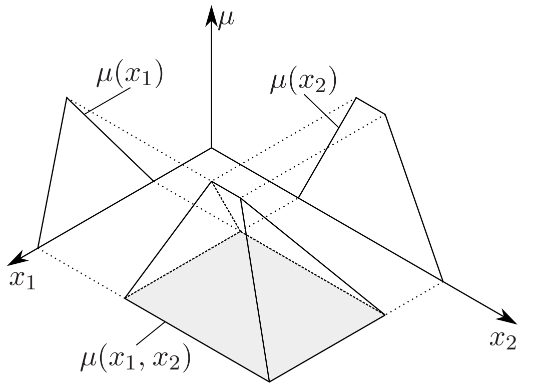
2.2.2 Extension Principle, Multi-Dimensional Fuzzy Quantities

The extension principle is defined by [21, p. 615; 100, p. 346] for two fuzzy quantities, but can be expanded to arbitrarily many input quantities. Let $Z^{\mathrm{f}} = f(X^{\mathrm{f}}_1, \dots, X^{\mathrm{f}}_n)$ be a fuzzy result quantity of an operation on the $n \in \mathbb{N}^+$ fuzzy quantities X^{f}_1 to X^{f}_n. Each of these quantities can be of any dimension. Deterministic samples, or members of those quantities are noted as x_1 to x_n, and z and have the same dimension as their respective fuzzy quantity. The deterministic solution of a set of samples is noted as $z = f(x_1, \dots, x_n)$. The membership of Z^{f} can be obtained by means of the extension principle

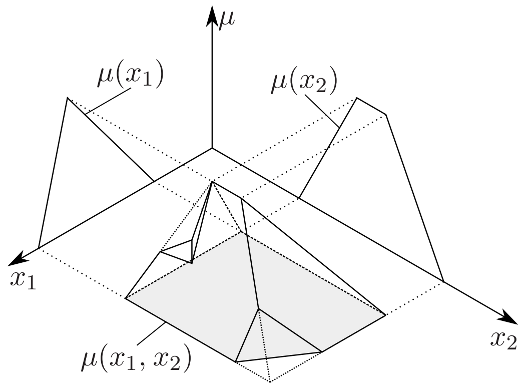
$$\mu(z) = \sup_{z=f(x_1,\dots,x_n)} \min\left(\mu(x_1), \dots, \mu(x_n)\right). \tag{2.5}$$

The whole procedure is an optimization problem [54, p. 2]. For continuous quantities, optimization for the most extreme values of z, while maximizing the membership $\mu(z)$, is done. For discrete quantities the combinations of all x_i, which yield the maximum membership for z, are to be found.

Using the CARTESian product $X_1 \times \ldots \times X_n$, a single higher-dimensional compound quantity X_{CARTES} can be constructed from $n \in \mathbb{N}^+$ sub-quantities. The dimension of the compound is equal to the sum of all its sub-quantities dimensions $\dim(X_{\mathrm{CARTES}}) = \sum_{i=1}^{n} \dim(X_i)$. The joint membership function of the compound quantity may be either given explicitly or constructed from the self-contained sub-quantities by means of the extension principle [84, p. 381], using the functional operation $f(x_1, \dots, x_n) = (x_1, \dots, x_n)$. For compound quantities, additional interactions may be imposed upon pairwise combinations of x_i and x_j, which limit the joint membership function. Inherent relations between the sub-quantities are described with these interactions. Interactions between fuzzy quantities interpretable as counterpart to correlation in stochastic quantities [31]. In α-level-based analyses, those interactions are translated on each α-level to constraints for the design space of the α-level-optimization, restricting the possible combinations of component values [78]. For α-level-free methods this translation is required only on the level of support to exclude sample combinations of the support's bounding box, which are not members of the support. The smallest possible hyperrectangle, containing the entirety of its contents, is called bounding box. Only axis aligned bounding boxes considered in this thesis, that is hyperrectangles, whose edges are parallel to the axes of a CARTESian coordinate system. Therefore it is possible to describe the bounding box as the CARTESian product of the intervals of minimum to maximum for each component. In Figure 2.1 examples for CARTESian products between an fuzzy triangular number and a fuzzy trapezoidal number are shown. Figure 2.1(a) shows a CARTESian product without interactions, Figure 2.1(b) with two interactions. The support of the respective CARTESian product is filled in light gray.



(a) CARTESian product of two fuzzy quantities.



(b) CARTESian product of two fuzzy quantities with interactions.

Figure 2.1: CARTESian product of two one-dimensional quantities.

2.2.3 Generic Representation of α-Levels

For one-dimensional, convex fuzzy quantities, an α-level is defined as the compact interval between two crisp limits. It can be obtained by an α-cut as given in Equation (2.1). α-level of a CARTESian product of one-dimensional quantities without interactions are hyperrectangles, but depart from it under consideration of interactions [57]. The postulation of convexity set is not necessarily fulfilled anymore with non-convex interaction functions. An example of a non-convex α-level in $\mathbb{R}^2$ is given in Figure 2.2.

In PUQPY, by default, an α-level is represented by the lower and upper diagonal corner of its bounding box, thus as a hyperrectangle. Alternatively, the α-level can be represented by its outline or hull. In PUQPY, the outline consists of discrete data points and is found using PARETO-fronts. Due to the usage of PARETO-fronts, see Section 2.4.3, voids in the α-level can not be respected. Thus, a one-dimensional slice of the α-level parallel to one of the axes is e a compact straight-line segment. Therefore the spaces outlined by the α-level's hull is coherent. For one-dimensional or non-interacting convex fuzzy quantities, the representation of an α-level as a hyperrectangle and the outline are equal. The entirety of members with membership values greater than the given value will be called α-cap.

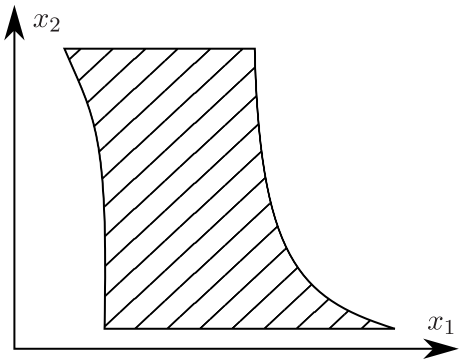
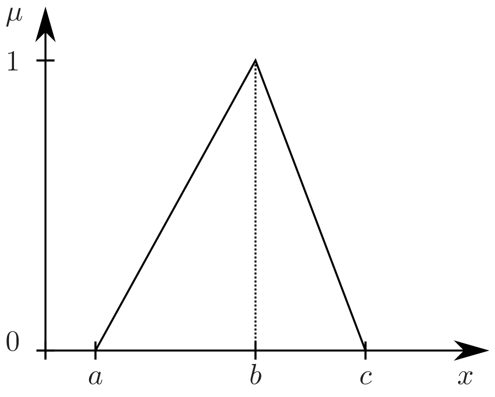


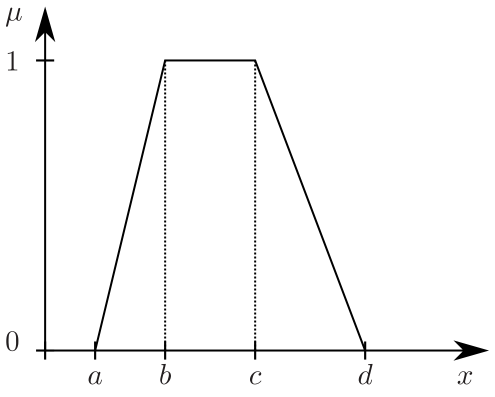
Figure 2.2: Example for an α-level in $\mathbb{R}^2$.

2.2.4 Analytical Fuzzy Quantities

An analytical fuzzy quantity is given by an analytical membership function. To be a valid membership function it needs to fulfill the assumptions made in Section 2.2.1 to be convexity, normalization, and boundedness. In $\mathbb{R}$, a fuzzy quantity is normalized, if the membership $\mu(x)$ is a continuous mapping to the closed interval $[0, 1]$ and $\exists\, x : \mu(x) = 1$, see [93, p. 2]. A fuzzy quantity in $\mathbb{R}$ is bounded, if the support is given by two real numbers $C_{0,\,\mathrm{l}} \leq C_{0,\,\mathrm{u}}$ and $\mu(x) = 0\,\forall\, x \notin C_0$, see [93, p. 2]. The fuzzy quantity is convex, if the inclusion property is fulfilled, see Equation (2.3). This is the case, if the membership function is monotonically increasing from $C_{0,\,\mathrm{l}}$ to $C_{1,\,\mathrm{l}}$, and likewise monotonically decreasing on the other side from $C_{1,\,\mathrm{u}}$ to $C_{0,\,\mathrm{u}}$, see [93, p. 2]. Some types of analytical fuzzy quantities, are given in this section.



(a) Example for a fuzzy triangular quantity.



(b) Example for a fuzzy trapezoidal quantity.

Figure 2.3: Fuzzy triangular and trapezoidal quantity.

Fuzzy Singleton, Crisp Sets and Interval Quantities

Crisp numbers, which feature no uncertainty, are special cases of fuzzy quantities, so called fuzzy singletons [37, p. 50; 21, p. 614]. The support of a fuzzy singleton is a crisp number. Ordinary sets, which are a special case of fuzzy quantities. According to [41, p. 35], the membership function of an ordinary set is the indicator function

$$\mu(x) = \begin{cases} 1 & x \in X, \\ 0 & x \notin X. \end{cases} \tag{2.6}$$

Thus, the support and the core of an ordinary set are equal, see Section 2.2.1. Being an ordinary set in $\mathbb{R}$, interval quantities are a special case to fuzzy trapezoidal numbers.

Fuzzy Triangular Number

A fuzzy number has a triangular shaped membership function and is notated in this thesis as $X^{\mathrm{f}}_{\mathrm{tri}} = \langle a, b, c \rangle$. There are several different conflicting notations proposed in [21; 37, p. 46; 34, p. 198; 76]. In this thesis, the membership function, in conformity with [34, p. 198], is defined as

$$\mu(x) = \begin{cases} \frac{x-a}{b-a} & a < x < b, \\ 1 & x = b, \\ \frac{c-x}{c-b} & b < x < c, \\ 0 & \text{otherwise.} \end{cases} \tag{2.7}$$

The support of a fuzzy triangular number is a crisp interval $C_0(X^{\mathrm{f}}_{\mathrm{tri}}) = \{[a, c] \mid a \leq b \leq c \mid a, b, c \in \mathbb{R}\}$. The core of a fuzzy triangular number consists of exactly one element $C_1(X^{\mathrm{f}}_{\mathrm{tri}}) = \{b \in \mathbb{R} \mid \mu(b) = 1\}$. A schematic of the membership function is shown in Figure 2.3(a). The α-cut $C_\alpha = \{x \mid \mu(x) \geq \alpha\}$ for the given membership $\alpha : \alpha \in [0, 1]$ is

$$\begin{aligned} x_{\mathrm{l},\,\alpha} &= a + \alpha \cdot (b - a), \\ x_{\mathrm{u},\,\alpha} &= c - \alpha \cdot (c - b), \\ C_\alpha &= [x_{\mathrm{l},\,\alpha}, x_{\mathrm{u},\,\alpha}]. \end{aligned} \tag{2.8}$$

Fuzzy Trapezoidal Number

A fuzzy trapezoidal number $X^{\mathrm{f}}_{\mathrm{tra}} = \langle a, b, c, d \rangle$ with the membership function, analogously to Section 2.2.4, see [1, p. 19; 62, p. 412; 76, p. 260],

$$\mu(x) = \begin{cases} \frac{x-a}{b-a} & a < x < b, \\ 1 & b \leq x \leq c, \\ \frac{d-x}{d-c} & c < x < d, \\ 0 & \text{otherwise.} \end{cases} \tag{2.9}$$

A schematic of the membership function is shown in Figure 2.3(b). If $b = c$, the fuzzy trapezoidal number reduces to a fuzzy triangular quantity. The inverse, yielding the α-cut $C_\alpha = \{x \mid \mu(x) \geq \alpha\}$ for the given membership $\alpha : \alpha \in [0, 1]$ is

$$\begin{aligned} x_{\mathrm{l},\,\alpha} &= a + \alpha \cdot (b - a), \\ x_{\mathrm{u},\,\alpha} &= d - \alpha \cdot (d - c), \\ C_\alpha &= [x_{\mathrm{l},\,\alpha}, x_{\mathrm{u},\,\alpha}]. \end{aligned} \tag{2.10}$$

Piecewise Linear Fuzzy Quantities

A piecewise linear fuzzy quantity $X^{\mathrm{f}}_{\mathrm{pwl}}$ is a special case of a fuzzy quantity with a piecewise defined membership function. It is constructed by a set of data points P containing the x-value and the according membership. The membership is calculated as shown in Algorithm 1. Firstly the two data points, that are closest to the sample x to the left, and

Algorithm 1 Membership function for a piecewise linear fuzzy quantity

```
procedure MEMBERSHIP(x)
    for $P \in X^{\mathrm{f}}_{\mathrm{pwl}}$ do
        if $P_x = x$ then
            return $P_\mu$
    $d_{\mathrm{l}} \leftarrow -\infty$
    $d_{\mathrm{u}} \leftarrow \infty$
    $\mu_{\mathrm{l}} \leftarrow$ None
    $\mu_{\mathrm{u}} \leftarrow$ None
    for $P \in X$ do
        $d \leftarrow P_x - x$
        if $d < 0.0 \wedge d > d_{\mathrm{l}}$ then
            $d_{\mathrm{l}} \leftarrow d$
            $\mu_{\mathrm{l}} \leftarrow P_\mu$
        else if $d > 0.0 \wedge d < d_{\mathrm{u}}$ then
            $d_{\mathrm{u}} \leftarrow d$
            $\mu_{\mathrm{u}} \leftarrow P_\mu$
    if $\mu_{\mathrm{l}} =$ None $\vee$ $\mu_{\mathrm{u}} =$ None then          ▷ is outside of the support
        return $0.0$
    return $\mu_{\mathrm{l}} \cdot \frac{d_{\mathrm{u}}}{d_{\mathrm{l}}+d_{\mathrm{u}}} + \mu_{\mathrm{u}} \cdot \frac{d_{\mathrm{l}}}{d_{\mathrm{l}}+d_{\mathrm{u}}}$
```

right are to be identified. If the sample has the same value as one of the data points, the data point's membership is returned. If for any of those two no point is found, the sample is considered outside the support, thus the membership of zero is returned. Secondly, after successfully finding the containing interval $[a, b]$, the membership can be calculated via linear interpolation to

$$\mu(x) = \mu(a) \cdot \frac{b-x}{b-a} + \mu(b) \cdot \frac{x-a}{b-a}. \tag{2.11}$$

The calculation steps for the α-cut for this quantity type are shown in Algorithm 2. Firstly, the data points on the inner side, which is on the side with the higher membership are found using the α-cut-method. Those have equal or higher membership than the specified α. If the membership of a data point is equal to α, then the respective bound of the α-level is already found. Otherwise, the next outer data point has to be found as the closest data point to the inner one with a lower membership. Since the membership function can have discontinuities, the x-value of the inner and outer point can be equal. If none is found, the inner data point is considered to be already the outermost data point in that direction, that is the border of the support. Given an outer data point is found, the x-value for the lower boundary of the α-level is interpolated with the equation

$$x = x_{\mathrm{o}} \cdot \frac{\mu_{\mathrm{i}} - \alpha}{\mu_{\mathrm{i}} - \mu_{\mathrm{o}}} + x_{\mathrm{i}} \cdot \frac{\alpha - \mu_{\mathrm{o}}}{\mu_{\mathrm{i}} - \mu_{\mathrm{o}}}. \tag{2.12}$$

For the upper α-level boundary the formula is analogue.

Algorithm 2 α-cut for a piecewise linear fuzzy quantity

```
1: procedure GET_ALPHACUT($\alpha$)
2:     $P_l, P_u \leftarrow \alpha$-cut                          ▷ $\alpha$-cut with data points
3:     $\mu_i \leftarrow P_{l,\mu}$
4:     if $\mu_i = \alpha$ then                                  ▷ lower $\alpha$-level bound
5:         $x_l \leftarrow P_{l,x}$                              ▷ bound found
6:     else
7:         $x_i \leftarrow P_{l,x}$
8:         $x_o \leftarrow -\infty$
9:         for all $P \in X$ do                      ▷ find data point to the left of $x_u$
10:            if $x_o < P_x \le x_i \wedge P_\mu < \mu_i$ then
11:                $x_o \leftarrow P_x$
12:                $\mu_o \leftarrow P_\mu$
13:        if no outer point is found then
14:            $x_l \leftarrow P_{l,x}$
15:        else
16:            $x_l \leftarrow$ Equation (2.12)
17:    if $\mu_i = \alpha$ then                                  ▷ upper $\alpha$-level bound
18:        $x_u \leftarrow P_{u,x}$                              ▷ bound found
19:    else
20:        $x_i \leftarrow P_{u,x}$
21:        $x_o \leftarrow \infty$
22:        for all $P \in X$ do                     ▷ find data point to the right of $x_u$
23:            if $x_o > P_x \ge x_i \wedge P_\mu < \mu_i$ then
24:                $x_o \leftarrow P_x$
25:                $\mu_o \leftarrow P_\mu$
26:        if no outer point is found then
27:            $x_u \leftarrow P_{u,x}$
28:        else
29:            calculate $\alpha$-level bound using Equation (2.12)
30:    return $[x_l, x_u]$
```

α-Level-based Fuzzy Quantities

α-level-based fuzzy quantities are defined by a set of intervals. For each interval a membership level is specified. The membership for a point is determined by the highest nominal value of all intervals, in which the point is contained. The α-cut is given by the biggest interval, which has at least the specified membership. A schematic of the membership function is shown in Figure 2.4.

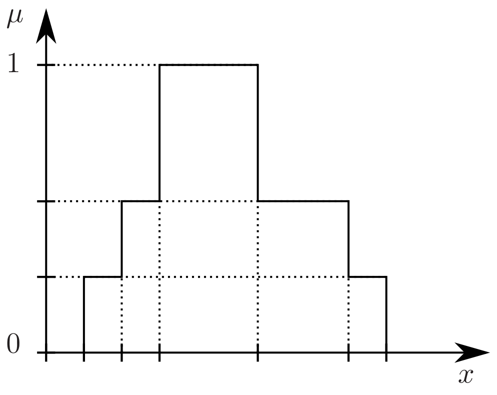


Figure 2.4: Example for an α-level-based quantity.

2.2.5 Empirical Fuzzy Quantities

Empirical fuzzy quantities are not defined by an analytical continuous membership functions, but by discrete data points with their respective membership values. Result quantities of fuzzy analysis, obtained by means of optimization are empirical fuzzy quantities.

Interpolation of Membership

To sample from an empiric quantity continuously, an interpolation technique is required. Piecewise linear fuzzy quantities, as shown in Section 2.2.4, employ linear interpolation between the data points. A one-dimensional empiric fuzzy quantity can be converted to a piecewise linear one. As empiric fuzzy quantities can be of any dimension, interpolation methods are required to sample continuously. Since multidimensional interpolation is a research topic on its own, only the simplest method, the nearest neighbor interpolation is implemented in PUQpy. The available methods are set up to be easily extensible. The methods must be able to interpolate on arbitrary point clouds, without relying on any form of grid.

In nearest neighbor interpolation the nearest data point is found, and its properties, except the coordinates in design space are copied to the new data point. In Voronoi-diagrams this can be visualized by filling cells with solid colors [11, p. 5].

An approach to natural neighbor interpolation is given in [45]. It is based on Voronoi-diagrams and Delauney-triangulation. The point to be sampled x will introduce a new cell into the diagram, which takes parts from all n neighboring cells. The value to the sampling point $\mu(x)$ is calculated as summation over the weighted values of the neighboring cells

$$\mu(x) = \sum_{i=1}^{n} w_i(x) \cdot \mu(x_i). \tag{2.13}$$

The SIBSON-weights of a neighboring cells are assigned in proportion to how much hypervolume V the neighboring cell loses to the new cell [42; 45]

$$w_i(x) = \frac{V(x_i)}{V(x)}. \tag{2.14}$$

According to [11, p. 5; 42, p. 5], the LAPLACE-weights, or non-SIBSONian weights are assigned in proportion to the shared interface $l(x_i)$, which is a hyperarea shared by the neighboring cell and the new cell. The distance $d(x_i)$ from the new point to the neighboring cell as

$$w_i = \frac{\frac{l(x_i)}{d(x_i)}}{\sum_{k=1}^{n} \frac{l(x_k)}{d(x_k)}}. \tag{2.15}$$

Calculation of α-Cuts of Empiric Quantities

For empiric fuzzy quantities two α-cut methods are available in PUQPY. The first, and default method for empiric quantities, shown in Algorithm 3, returns the bounding box of the α-level. For all components of the objective space, that is the dimension of the quantity,

Algorithm 3 α-cut yielding a hyperrectangular α-level

```
1: procedure α-CUT($\alpha$)
2:     $\boldsymbol{x}_l \leftarrow \infty$                         ▷ all elements
3:     $\boldsymbol{x}_u \leftarrow -\infty$                        ▷ all elements
4:     for $P \in X^{\mathrm{f}}$ do                    ▷ compare all data points
5:         $\boldsymbol{x}_l \leftarrow \min(\boldsymbol{x}_l, P_x)$           ▷ element wise
6:         $\boldsymbol{x}_u \leftarrow \max(\boldsymbol{x}_u, P_x)$           ▷ element wise
7:     return $\boldsymbol{x}_l, \boldsymbol{x}_u$
```

the minimum, and maximum is found across all data points. The second method shown in Algorithm 4 returns the actual outline of the α-level, consisting of data points. Firstly, 2^n PARETO-front objects are set up, one for each combination of minimum and maximum for all n dimensions of the quantity. After that, the PARETO-fronts are queried for all data points, thus finding the data points contributing to the respective outline segment. The procedure is described in Section 2.4.3. Finally, the data points of all PARETO-fronts are united, yielding the complete outline.

Algorithm 4 α-cut yielding the α-levels outline

```
1: procedure α-CUT-OUTLINE($\alpha$)
2:     set up PARETO-objects
3:     for $P \in X^{\mathrm{f}}$ do                    ▷ compare all data points
4:         if $P_\mu \geq \alpha$ then
5:             update all PARETO-objects with $P$
6:     outline $\leftarrow \bigcup P \in$ PARETO-objects
```

Projection of Multi-Dimensional Empirical Fuzzy Quantities

There are two different approaches available in PUQPY for reducing dimensionality of empiric fuzzy quantities through projection. In both approaches, inter-dependencies are dismissed, thus the fuzziness of the quantity is overestimated, see [31]. The data points stay unchanged, the data of the data point is not reduced in dimensionality.

For the first approach, PARETO sorting is done while the chosen axis is not considered as an optimization target, and thus ignored. Additionally the range of values along the chosen axis can be restricted. This can be used to extract slices with a certain thickness from the quantity. A quantity of one dimension less is obtained if the component of the chosen axis is ignored or deleted afterwards. The other technique is to use a PARETO sorting to only optimize for the given axis and the membership. All other components are ignored, thus finding only the hull along this axis. An one-dimensional quantity is obtained, if the components of all other axes are ignored, or deleted afterwards. For two dimensions the result is the same, if the first technique is applied to one axis, and the second technique to the other axis.

2.2.6 Information Reduction Measures

Information reduction measures, for fuzzy quantities called defuzzification, can be applied to reduce a fuzzy quantity to a crisp, real valued representation [37, pp. 139 sqq.]. There are two types of defuzzification measures. Performance oriented measures, such as characteristic values are in defined in Section 2.2.6. The second ones are robustness oriented measures, which quantify the uncertainty in a fuzzy quantity. Those are discussed in Section 2.2.6. For a holistic evaluation of the uncertain results of an uncertainty analysis, both types are required [31, p. 4]. According to [74], continuity is an important feature of a defuzzification measure, thus small changes to the fuzzy quantity result in small changes in the result of defuzzification.

In the PUQPY, defuzzification is done after the completion of a fuzzy analysis. The necessary methods are provided by the returned quantity.

Performance-oriented Defuzzification Methods

Performance oriented defuzzification methods reduce a fuzzy quantity to a characteristic value [46]. This characteristic value gives information about the approximate location of the fuzzy quantity in the space, but not on its uncertainty.

> **Example 2.3:** Coming back to the subjective perception of temperature illustrated in Example 2.1. This defuzzification task may be worded as “Give me the temperature, you consider comfortable.”

Lower and upper bounds of an arbitrary α-levels bounding box are retrieved by the α-cut-method

$$C_\alpha(X^{\mathrm{f}}) = [\boldsymbol{x}_{\alpha,\mathrm{l}}, \boldsymbol{x}_{\alpha,\mathrm{u}}]. \tag{2.16}$$

Those can be used directly as a measure or further calculation can be done. The most prominent may be using the core as a basis. If the core of the fuzzy quantity is a singular point, its value can be referred to as modal value x_{mod} [37]. In other cases, the modal value may be obtained for example by the mean of the core, random of maxima, first of maxima or last of maxima [46].

Bounding box center of a certain α-level, which is computed element-wise as

$$\boldsymbol{x}_{\alpha,\mathrm{bbc}} = \frac{\boldsymbol{x}_{\alpha,\mathrm{l}} + \boldsymbol{x}_{\alpha,\mathrm{u}}}{2}. \tag{2.17}$$

Especially interesting are $\boldsymbol{x}_{0,\,\mathrm{mean}}$ as mean of the support and $\boldsymbol{x}_{1,\,\mathrm{mean}}$ as the mean of the core, also called mean of maximum [26].

The expected value of a fuzzy quantity is an interval $[E_*, E^*]$. In accordance to [22, p. 292; 49, p. 2; 66, p. 3; 82] the lower expected value E_* and the upper expected value E^* can be calculated by

$$E_* = \int_0^1 \inf C_\alpha(X^{\mathrm{f}})\,\mathrm{d}\alpha \;= x_{1,\,\mathrm{l}} - \int_{x_{0,\,\mathrm{l}}}^{x_{1,\,\mathrm{l}}} \mu(x)\,\mathrm{d}x \tag{2.18}$$

$$E^* = \int_0^1 \sup C_\alpha(X^{\mathrm{f}})\,\mathrm{d}\alpha = x_{1,\,\mathrm{u}} + \int_{x_{1,\,\mathrm{u}}}^{x_{0,\,\mathrm{u}}} \mu(x)\,\mathrm{d}x\,. \tag{2.19}$$

The first version measures the area between $x = 0$ and the lower, respective upper, slope of the quantity by integrating along the vertical axis α, using α-cuts. The second version measures the area under the lower, respective upper, slope between the support and core of the quantity, using the membership function. In both versions, the measured area is implicitly converted to rectangle with the same area. Explicit division of the area by the height of the rectangle is omitted due to it being 1. This can be assumed, because of the premise of normalized fuzzy quantity.

The defuzzified value according to [37, pp. 140 sq.], based on bounding boxes of $m + 1$ equidistant distributed α-levels with $\alpha_j = \frac{j}{m}$ is given as

$$x^\diamond = \frac{1}{2^n(m+1)} \sum_{j=0}^{m} \sum_{k=1}^{2^n} \boldsymbol{x}_{j,\,k}. \tag{2.20}$$

Here k is a counter for all 2^n hyperrectangle corners and j is a counter for the α-level. This is the mean over all components of all α-level corners and yields a scalar value. For one-dimensional fuzzy quantities, this is equal to the mean of centroids and can be written by using $\boldsymbol{x}_{\alpha,\,\mathrm{bbc}}$ from Equation (2.17) as

$$x^\diamond = \frac{1}{(m+1)} \sum_{j=0}^{m} \boldsymbol{x}_{\alpha_j,\,\mathrm{bbc}}. \tag{2.21}$$

The centroid method according [26; 53] calculates the center of gravity of the membership function as

$$x_{\mathrm{cog}} = \frac{\int \mu(x) x\,\mathrm{d}x}{\int \mu(x)\,\mathrm{d}x} = \frac{\sum_i \mu(x_i) x_i}{\sum_i \mu(x_i)}. \tag{2.22}$$

This can be done element-wise for multidimensional quantities. For empiric quantities the second part may be employed, but it must be kept in mind that density concentrations are likely to drift it from the real value. If that is to be considered, the density must be factored in, such that clustered members get a smaller weight.

The center of area x_{coa} given by

$$\int_{x_{\mathrm{coa}}}^{\infty} \mu(x)\,\mathrm{d}x = \int_{-\infty}^{x_{\mathrm{coa}}} \mu(x)\,\mathrm{d}x \tag{2.23}$$

is the point, where the areas of both sides are equal [53]. Generalization to multidimensional and empiric quantities is not as easy as for the ones shown above.

Robustness-oriented Defuzzification Methods

Robustness oriented defuzzification methods give insight on how uncertain, that is how inaccurate, the quantity is. Information about the absolute location of the quantity in space is not gained with these approaches.

> **Example 2.4:** Coming back to the subjective perception of temperature illustrated in Example 2.1. This defuzzification task may be worded as “Give me the temperature span, your tolerance width, you consider comfortable.”

Absolute spread is defined as element-wise difference of the upper and the lower bound of the supports hyperrectangle

$$\boldsymbol{s}_{\alpha,\,\mathrm{abs}} = \boldsymbol{x}_{\alpha,\,\mathrm{u}} - \boldsymbol{x}_{\alpha,\,\mathrm{l}}. \tag{2.24}$$

Thus, absolute spread is the width of the fuzzy quantities bounding box [37, p. 90]. Relative spread is defined as the ratio of absolute spread $\boldsymbol{s}_{\alpha,\,\mathrm{abs}}$ to the characteristic modal value $\boldsymbol{x}_{1.0,\,\mathrm{bbc}} \neq 0$ as

$$\boldsymbol{s}_{\mathrm{rel}} = \frac{\boldsymbol{s}_{\mathrm{abs}}}{\boldsymbol{x}_{1.0,\,\mathrm{bbc}}}. \tag{2.25}$$

The absolute imprecision of a fuzzy quantity is the approximation of the cardinality of the set defined by the quantity $\mathrm{card}(X^{\mathrm{f}})$. Its geometric interpretation is the volume enclosed by the membership function [37, pp. 141 sq.]. It can be calculated by

$$\mathrm{imp}_{\mathrm{abs}}(X^{\mathrm{f}}) = \mathrm{card}(X^{\mathrm{f}}) = \int \mu(x)\,\mathrm{d}x\,. \tag{2.26}$$

For a bounding box based discretization of $m+1$ α-levels, this can be written as

$$\mathrm{imp}_{\mathrm{abs}}(X^{\mathrm{f}}) = \frac{1}{2m}\sum_{j=0}^{m-1}\left(\boldsymbol{s}_{\alpha_j,\,\mathrm{abs}} + \boldsymbol{s}_{\alpha_{j+1},\,\mathrm{abs}}\right). \tag{2.27}$$

Where j is a counter for the α-level with the value $\alpha_j = \frac{j}{m}$. The spread $\boldsymbol{s}_{\alpha,\,\mathrm{abs}}$, as calculated in Equation (2.24).

The relative imprecision of a fuzzy quantity is analogously to relative spread calculated for $\boldsymbol{x}_{1.0,\,\mathrm{bbc}} \neq 0$ by [37, p. 142]

$$\mathrm{imp}_{\mathrm{rel}}(X^{\mathrm{f}}) = \frac{\mathrm{imp}_{\mathrm{abs}}(X^{\mathrm{f}})}{\boldsymbol{x}_{1.0,\,\mathrm{bbc}}}. \tag{2.28}$$

The eccentricity $\mathrm{ecc} = x^{\diamond} - \boldsymbol{x}_{1.0,\,\mathrm{bbc}}$ is defined as the signed difference between the defuzzified value $x^{\diamond}$ and the modal value $\boldsymbol{x}_{1.0,\,\mathrm{bbc}}$ [37, p. 143]. The specific eccentricity is calculated by dividing the eccentricity by the imperfection.

2.3 Fuzzy Analysis

Fuzzy analysis is used to examine possibilistic and epistemic uncertainty. As defined in [31, p. 2], a fuzzy analysis is a computation of fuzzy result quantities Z^{f} from input fuzzy quantities X^{f} using the mapping $\xi^{\mathrm{f}} : X^{\mathrm{f}} \mapsto Z^{\mathrm{f}}$. The result membership function $\mu_{Z^{\mathrm{f}}}$ and therefore the result fuzzy quantity can be found by optimization. There are two numerical approaches to the optimization task of fuzzy analysis, α-level-based methods and α-level-free methods.

2.3.1 α-Level-based Methods

In α-level-based methods, the fuzzy input quantities are discretized vertically into a set of α-levels in preparation to the actual analysis [57]. Therefore, the quantity is sliced using α-cuts, as described in Equation (2.1) and depicted in Figure 2.5. The general procedure is shown in Algorithm 5. The analysis itself will be carried out based on those α-levels.

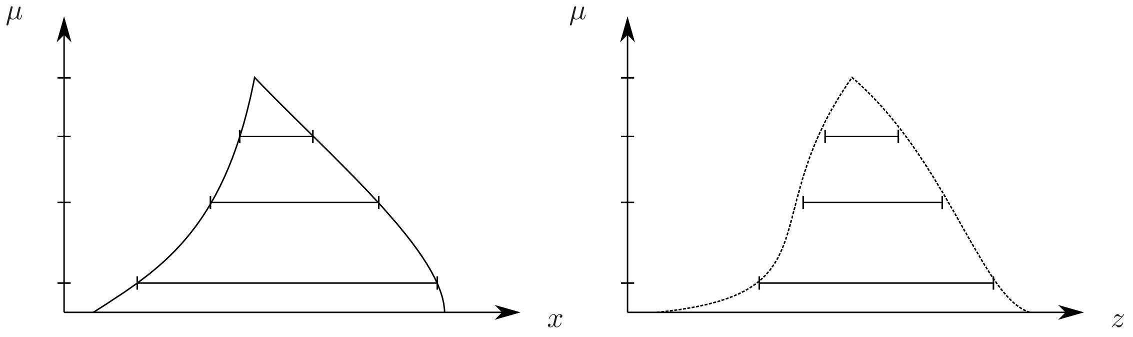


Figure 2.5: Discretization in α-level-based methods. Left: input quantity with α-levels; Right: result quantity; Dashed: the exact result.

Such an α-level-based analysis technique is α-level-optimization. A single α-level serves as the design space to a search for the global minima and maxima, which replaces the min-max-operator of the extension principle [84]. The most extreme outcomes of the fundamental solution will be searched on these individual α-levels. For multidimensional fuzzy result quantities, the outline of the α-level can be found by multi-objective optimization, see [31]. In total, 2^n Pareto-fronts are required for a n-dimensional fuzzy result quantity. The n-dimensional objective vector of a single Pareto-front is populated a by the objective, whether this component is to be of minimized $\downarrow (z)$ or maximized $\uparrow (z)$. This is one combination of the 2^n combinations $\{\uparrow, \downarrow\} \times (z_i, \ldots, z_n)$.

After all α-level are calculated, the output quantity is reassembled by stacking the α-levels. In this step, the convexity will be ensured, by checking, that a higher valued interval is fully part of all lower valued intervals. In case of a violation, the calculation needs to be restarted [57, p. 558].

The preferred calculation order of the α-levels depends on the optimization and data re-usage strategy employed. If no data is reused, the α-levels can be calculated independently, thus in arbitrary order or in parallel.

In [37, p. 105], it is suggested to move top-down in membership due to the assumption, that extremes in already calculated higher α-levels may approximate extremes in the current one. For lower α-levels the space to search may be larger in comparison to higher ones by

Algorithm 5 α-level-optimization

```
1: procedure α-LEVEL-OPTIMIZATION
2:     Discretize all quantities into $\alpha$-levels
3:     for all $\alpha$-levels do
4:         find maximum and minimum
5:     if not convex then                                  ▷ Check for convexity
6:         Restart optimization
7:     Reconstruct output quantity
```

order of magnitudes. Thus, it must be assumed, that the optimization algorithms knows little to nothing about the new design space and has to explore it completely again for each α-level. Surrogate models may be extremely inaccurate by extrapolating far from the trained data, thus unreliable to gain inference over the bigger space. The global optimum of a sub-space may be a local optimum in the containing space Thus, the search direction of this approach is local to global. Convexity of the fuzzy result quantity is implicitly ensured by using the result value of the next higher α-level as reference value to be improved on by the optimization in the current one. The result of higher α-level is guarantied to be included in all lower α-levels. Thus, each α-level needs to be calculated only once and post-computation can be skipped. The search for an optimum in a new α-level is started in a local optimum and the global optimum is to be found. If the fundamental solution is non-linear, it is not guaranteed to be found using neighborhood search with arbitrary big neighborhood. Vicinity based optimization techniques may fail by getting stuck at the local optimum. Thus, non-small steps need to be taken, essentially the design space needs to be fully discovered for each α-level. If fundamental solution is smooth and monotonous, it is guaranteed to be successful.

By treating the α-levels bottom-up, the widest α-level is treated first. Since higher α-levels are sub-spaces of lower ones, the search space is shrinking with increasing nominal membership value and its space is at least sparsely covered by the optimization on previous level. Discovering the design space on every α-level may be unnecessary. In the worst case, repeated rediscovery of the design space for every α-level is required, but in the best case only a single discovery is sufficient. The search direction of this approach is from global to local, since the global optimum may not be part of the sub-space. If the α-level space is sufficiently covered by data from previous α-levels, further optimization may be skipped completely. In this case, the current α-level's result can be found among the already known data points from previous α-levels. With the assumption of coverage, it is less likely, that neighborhood based optimization routines will converge to a local optimum, instead of to the global optimum. Surrogate models can be used to infer the behavior of the fundamental solution since data is over the space is available. If in post-computation, the result value in a higher α-level is found better than the result value of a lower one, all lower α-levels need to be reevaluated. This can be done by updating to the found better value or restarting optimization.

2.3.2 α-Level-free Methods

In α-level-free methods, the fuzzy quantity is not discretized into α-levels. Discretization happens by sampling and the sample's membership and the fundamental solution will be evaluated for every sample individually. This approach can be seen as a direct multi-objective optimization approach to the extension principle. In total, 2^n PARETO-fronts are required for a n-dimensional fuzzy result quantity. The first component of the $(n+1)$-dimensional objective vector of a single PARETO-front is set to maximize the membership $\uparrow (\mu(z))$. The rest of the n components are populated by the objective, whether this component is to be of minimized $\downarrow (z)$ or maximized $\uparrow (z)$. This is one combination of the 2^n combinations $\{\uparrow, \downarrow\} \times (z_i, \ldots, z_n)$.

A visual representation is given in Figure 2.6. Note, that in Figure 2.6, the resulting

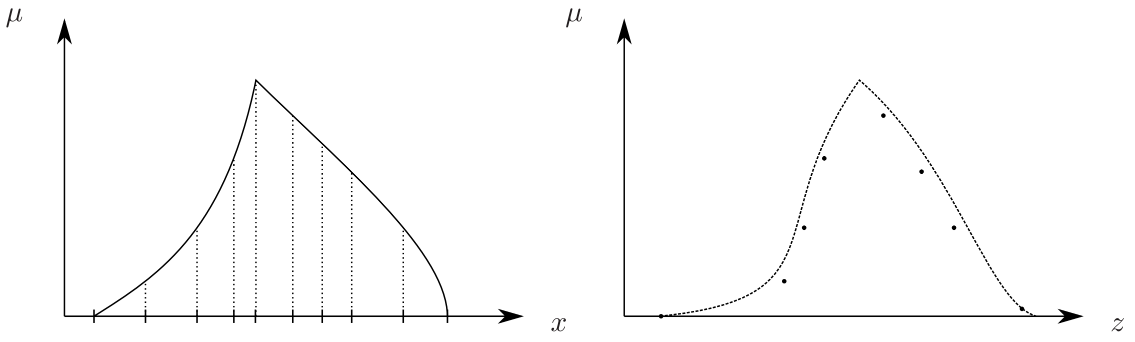


Figure 2.6: Discretization in α-level-free methods. Left: input quantity with samples; Right: result quantity; Dashed: the exact result.

quantity consists of fewer points than originally sampled. This loss is due to sub-optimal realizations being ejected from PARETO-fronts. A brute point mesh would be numerically extremely expensive [84, p. 382]. This issue can be overcome with appropriate sampling algorithms. One of those is *SWS*, see Section 2.5.3.

Algorithm 6 Naive α-level-free fuzzy analysis approach

```
1: procedure α-LEVEL-FREE METHOD
2:     Generate samples $s$
3:     for all $s$ do
4:         Evaluate $\mu(s)$
5:         Evaluate $z(s)$
6:     Reconstruct output quantity
```

2.3.3 Non-Flat α-Level-Optimization

Non-flat α-level-optimization is a hybrid of the classical α-level-optimization approach, described in Section 2.3.1 and the α-level-free method in Section 2.3.2. It combines both techniques by carrying out the optimization based on α-levels as in the classical α-level-optimization. The input quantities' α-levels are used as design space and are searched for minima and maxima in objective spaces. But for each sample, the membership is evaluated individually, instead assigning the level's nominal value, which may hold some advantages.

Consideration of constraints for the optimization due to interactions is built-in by requiring the individual samples membership to be at least the nominal value of the α-level. Therefore, an explicit translation to a constraint prior to the optimization is unnecessary.

In post-processing a higher information content is suspected. After the first calculation of an α-level, some net points of the result quantities' membership function on higher memberships are known. Thus, not only the performance function over the design space is know, but also the behavior of its membership function. Obviously, this can only be taken advantage of, if α-levels are evaluated in ascending order and data points are held in memory until the end of the analysis.

2.3.4 Comparison of Approaches

Both approaches aim to find the most extreme outcomes, with the highest membership value at the same time, thus fulfilling the extension principle as in Equation (2.5). In both approaches, Pareto-optimal solutions are found, as described in Section 2.4.

α-level-optimization is a multi-criteria optimization with an a predefined chosen value for the membership. This reduces the dimension of the objectives by one, similar to the ε-constraint method in [16, pp. 420 sqq.]. There, an objective is a priori converted to an equality or inequality constraint, thus reducing the dimensionality of objective space to be optimized by one dimension. The membership of an α-level becomes a constraint to be greater or equal the current α. The last step of an α-level-optimization is a post-computation to ensure the convexity of the result. This is an ordered Pareto-check. If a higher level dominates a lower one, the search for the lower one is restarted.

α-level-optimization is well suited, if coarse output quantities with predefined α-levels are wanted. That is the case, if only a few levels are needed to adequately reconstruct the resulting membership function with. A densely sampled membership function may not be of use in a nested analysis project, where performance oriented information reduction measures are used in-between the staged analyses. Drawbacks of α-level-optimization are the discretization of the objective space and the resulting membership function. It is less capable of representing dependencies of quantities in design space, as well as in objective space.

α-level-free techniques are able to deal with almost all design space dependencies and are able to represent the dependencies of multivariate result quantities in the objective space. On the other hand, sampling may loose efficiency due to the curse of dimensionality [49, p. 3]. α-level-free methods may yield a denser output membership function. How equally distributed the mesh points of the results membership function are, is entirely depended on the optimizer. For multidimensional α-level-free methods are to be preferred fuzzy output and implicit problems [36, pp. 51 sq.]. It is better suited, when high density, high resolution output quantities are wanted. Retrieval of exact α-levels is not guaranteed and may be done in post-computation. Distributive information reduction measures may yield a better result due to the denser result membership function.

2.4 Optimization

Optimization is the process of finding the best possible solution among available solutions [14, p. 6]. Search algorithms are used to maximize or minimize the outcome of a fundamental solution, the objective function [6, p. 23; 14, p. 6]. If the properties of the fundamental solution are not exploited, a problem is treated as a blackbox, [4, p. 425; 5]. Choosing the best suited optimization algorithm is an optimization problem in itself [10, p. 4]. How well an algorithm performs on a given problem is measured by benchmarking.

A plenitude of optimization algorithms exist. Introduction to the topic is given for example in [14; 40; 70]. In [10], a variety of algorithms and frameworks is given. Problem classes according to [17; 18] are

- single-objective, uni-global with a single global optimum,
- single-objective, multi-global optimization problems with multiple global optima,
- multi-objective, uni-global optimization problems with a single PARETO-front,
- multi-objective, multi-global optimization problems with multiple PARETO-fronts.

In [75] parallelization, hybridization, time continuation, evaluation relaxation, and surrogate models are identified as possible strategies to enhance the efficiency. As shown in [4, p. 425; 5], parallelization for blackbox problems can be done by

- evaluating several blackboxes in parallel,
- evaluating parallelized blackboxes sequentially,
- using mixed parallelism, thus running several parallelized blackboxes in parallel.

2.4.1 Components of an Optimization Algorithm

An optimization algorithm consists of a combination of several components:

1. Search method: Sample generation and heuristic,
2. Feasibility measure,
3. Fundamental solution evaluation,
4. Performance measure and domination trial,
5. Data management.

Sampling can be interleaved with the heuristic techniques, to increase the general usefulness of samples. A heuristic technique is used to find patterns in the models response, thus focusing on promising regions in design space [14, p. 8]. In a surrogate model the fundamental solution is abstracted into a function whose evaluation is less expensive [5]. Samples are either evaluated alternating on the real model and on the surrogate model or entirely on the surrogate model, as done in [12].

Although it is preferable to focus on the best regions, the whole input space needs to be explored. Thus, diversity of samples against focus on performant samples is a difficult balance in optimization algorithms [16, p. 433; 44, p. 2], since for both enhancement is only achievable by increasing the number of evaluations. The precision of the result can be improved by more evaluations close to the optimum. On the other hand, reliability can be improved by more diverse samples and a more general coverage of the design space, increasing the number of sub-optimal solutions. Generally, it is wanted to keep the number of unprofitable evaluation as low, as possible.

The feasibility of samples is measured with constraints. There are hard constraints, that have to be satisfied and soft constraints, that are wanted to be satisfied, but are not essential

[14, p. 5]. Infeasible solution can be rejected, thus not passed to the fundamental solution or attempted to be repaired [80, p. 204]. Checking feasibility of sample before evaluation of the fundamental solution prevents impossible or implausible combinations to be evaluated. Vicinity constraints can be applied to ensure higher diversity among the samples and avoid reevaluation of nearly identical samples.

The performance of the samples is measured based on the result of the fundamental solution. Whether a solution is an improvement or a miscarriage over previous attempts regarding to the objective, is assessed in a domination trial [16, pp. 410 sqq.]. Samples, that are winners of this trial are considered contributing to the solution and can be used to draw new generations of samples.

Required data is stored, which can be used in heuristic techniques, vicinity checks or generation of the samples. Finally, the result of the optimization is constructed from the storage.

2.4.2 Run-Time Performance Indicators

Depending on the fundamental solution and its expense, different heuristics may yield the best run-times. For easy fundamental solutions, the time spent in the heuristic may exceed the time spent for the evaluation of the fundamental solution, making the heuristic ineffective. In this case, a less expensive heuristic or a brute force approach may yield better performance. For long running fundamental solutions, the run-time scales approximately linear with n_{tot}. Thus, the proportion of evaluations contributing to the final result in comparison to the total evaluation count is

$$p_{\text{eff}} = \frac{n_{\text{contrib}}}{n_{\text{tot}}}. \tag{2.29}$$

To be able to tell, when a fundamental solution is long running, an indicator for the spent time is required to evaluate the efficiency. In the simplest case, the cumulative time spend in parts over the course of a complete optimization run could is interesting. The cumulative time spend in routines can be measured with profiling tools. Python comes with a handful of very powerful profiling libraries. With the time spend in the heuristic t_{heu} and t_{fund} in the fundamental solution respectively, the ratios are calculated to

$$t_{\text{tot}} = t_{\text{heu}} + t_{\text{fund}}, \tag{2.30a}$$

$$r_{\text{heu}} = \frac{t_{\text{heu}}}{t_{\text{heu}} + t_{\text{fund}}}, \tag{2.30b}$$

$$r_{\text{fund}} = \frac{t_{\text{fund}}}{t_{\text{heu}} + t_{\text{fund}}}. \tag{2.30c}$$

Based on this indicator, it is possible to decide, how complex the heuristic can be. By combining of both indicators, it is possible to deduce, how many fundamental solutions the heuristic is worth.

2.4.3 Pareto-Fronts

In multi-objective optimization, Pareto-fronts and Pareto-sets play an important role [25]. A Pareto-front is a set of non-dominated solutions in the objective space. A Pareto-set denotes the set of corresponding points in the design space. On a Pareto-front improving one feature is not possible without worsening another. All members of the front are equally optimal, but trade one aspect against the others [16, p. 405]. A solution is non-dominated when there is no other known solution, that dominates it. Domination is defined as follows. A solution $\boldsymbol{x}$ dominates another solution $\boldsymbol{y}$ if and only if

$$\boldsymbol{x} \prec \boldsymbol{y} \iff \forall\, \boldsymbol{x}_i \nsucc \boldsymbol{y}_i \;\wedge\; \exists\, \boldsymbol{x}_i \prec \boldsymbol{y}_i. \tag{2.31}$$

Here $\prec$ denotes "better", the tip is pointing towards the better solution. That means $\boldsymbol{x}$ is in all components not worse, that is better or equal, than $\boldsymbol{y}$ and at least one component of $\boldsymbol{x}$ is actually better than $\boldsymbol{y}$ [16, p. 412]. Each component can have its own objective, and either be minimization, maximization or even disregarded. To update a front, it is required to compare the candidate solution to all members of the front. Early abortion is possible, if any member dominates the candidate by rejecting the candidate. If the candidate is non-dominated, each member needs to be removed, that is dominated by the candidate.

As test runs early in development showed, most time was spend in updating the Pareto-fronts, the number of domination checks as the most important factor. To cut these calls, the following algorithms are compared. In Algorithm 7, the first stage is to check whether the candidate is inferior to any member. If so, the algorithm is aborted and the front stays unaltered. The second stage is only reached, if the candidate is not dominated by any member, thus will become a member itself. All members, that are inferior to the candidate are removed.

Algorithm 7 Approach 1 to updating a Pareto-front

```
1: for all members do                                          ▷ first stage
2:     if dominates(member, candidate) then
3:         abort
4: for all members do                                          ▷ second stage
5:     if dominates(candidate, member) then kick member
6: add candidate
```

In Algorithm 8, a different approach is taken. The front is iterated over just once. But for each member a bilateral domination check is done. If the candidate is inferior, the procedure is aborted again. If however, the candidate dominates a member, thus is a qualified member itself, all remaining checks, to rule out the candidate are skipped.

Algorithm 8 Approach 2 to updating a Pareto-front

```
1: approved ← False
2: for all members do
3:     if approved == False then
4:         if dominates(member, candidate) then
5:             abort
6:     else if dominates(candidate, member) then
7:         kick member
8:         approved ← True                  ▷ skip all remaining checks in Line 4
9: add candidate
```

Algorithm 7 is more efficient in rejecting candidates, especially, when the candidate is loosing against a later member. On the other hand, Algorithm 8 could be faster, if the candidate is accepted as soon as possible. For adding a non-dominated candidate, both algorithms evaluate domination equally often. Best and worst call frequencies for a single candidate check against a front with n member is broken down in Table 2.1.

Table 2.1: Number of domination check calls.

	Rejecting		Accepting	
	Approach 1	Approach 2	Approach 1	Approach 2
Best case	1	1	$2n$	$n+1$
Worst case	n	$2n-1$	$2n$	$2n$

To test, which of those is more performant, the example given in Section 4.1 is used as a benchmark. In total, 2 200 data points are queried against two fronts to be sorted into, resulting in the call frequencies shown in Table 2.2. The approach described in Algorithm 7 is the more effective one for this use case.

Table 2.2: Number of domination check calls.

	Approach 1	Approach 2
Run 1	179 501	199 245
Run 2	177 988	233 628
Run 3	180 600	214 699
Run 4	165 423	220 008

Further investigations showed, that the initial implementation showed in Algorithm 9 is ineffective, due to comparing repeatedly entire arrays. All operations are carried out element-wise with NUMPY-routines on the entire arrays [38]. Whereas $\boldsymbol{a}$ is the array, whose domination over $\boldsymbol{b}$ is evaluated, $\boldsymbol{o}$ is an array of truth values, whether the component should be minimized (`False`) or maximized (`True`). Finally $\boldsymbol{i}$ is an array, which specifies, whether the component is to be ignored or taken into comparison.

Algorithm 9 Initial `dominates()`

```
1: procedure DOMINATES($\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{o}, \boldsymbol{i}$)
2:     mask components of $\boldsymbol{a}, \boldsymbol{b}$, that are specified in $\boldsymbol{i}$
3:     $\boldsymbol{l} \leftarrow \boldsymbol{a} < \boldsymbol{b}$
4:     $\boldsymbol{g} \leftarrow \boldsymbol{a} > \boldsymbol{b}$
5:     $\boldsymbol{q} \leftarrow \boldsymbol{a} == \boldsymbol{b}$
6:     where $\boldsymbol{o}$ == True: $\boldsymbol{r}_{\text{better}} \leftarrow \boldsymbol{g}$; where $\boldsymbol{o}$ == False: $\boldsymbol{r}_{\text{better}} \leftarrow \boldsymbol{l}$
7:     $\boldsymbol{r}_{\text{notworse}} \leftarrow \boldsymbol{q} \vee \boldsymbol{r}_{\text{better}}$
8:     return whether $(\exists(\boldsymbol{r}_{\text{better}}) \wedge \forall(\boldsymbol{r}_{\text{notworse}}))$
```

In benchmarks, Algorithm 10 outperformed Algorithm 9 by give or take 100 times. The elements of the objective array $\boldsymbol{o}$ are -1 for minimization, 1 for maximization and 0 for ignoring the component. The actual value of a component of $\boldsymbol{r}$ is not of interest, but only its sign is important. Here NUMPY-routines operate element-wise on entire arrays again.

Algorithm 10 More efficient `dominates()`

```
1: procedure DOMINATES($\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{o}$)
2:     $\boldsymbol{r} \leftarrow \boldsymbol{o} \cdot (\boldsymbol{a} - \boldsymbol{b})$
3:     return whether $(\exists(\boldsymbol{r}_i > 0) \wedge \forall(\boldsymbol{r}_i \nless 0))$
```

With Algorithm 11, the performance can be increased further by using static data types and pre-compilation with Cython. For documentation on Cython, see [9]. This performance increase may be more noticeable with many objectives, due to the early abortion. It is iterated over the elements of the arrays explicitly. The variables are as in Algorithm 10. The variable d holds the domination status and becomes only `True`, if Equation (2.31) is fulfilled. The length of the compared arrays is noted as n.

Algorithm 11 Cython – fast `dominates()`

```
1: procedure DOMINATES($\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{o}$)
2:     $d \leftarrow$ False
3:     for $i$ in range($n$) do
4:         $r \leftarrow \boldsymbol{o}_i \cdot (\boldsymbol{a}_i - \boldsymbol{b}_i)$
5:         if $r < 0$ then
6:             return False                                    ▷ early abortion
7:         else if $r > 0$ then
8:             $d \leftarrow$ True
9:     return $d$
```

2.5 Algorithms

2.5.1 Slice Sampling

Slice sampling is a Markov-Chain Monte-Carlo (*MCMC*) based sampling method, introduced in [64; 65] and able to sample from arbitrary distributions. Recommendations regarding sampling using *MCMC* are given in [28]. For more details on *MCMC*, see [13; 24; 29; 43; 63].

Slice sampling is able to sample directly from a weight function w, which is proportional to the probability density function (*pdf*) of the desired distribution. An auxiliary variable is used, the inverse cumulated density funtion (*icdf*) is not necessary. Therefore, it is possible to sample from arbitrary distributions, for which the *icdf* is not easily available or not available at all. The general procedure is as follows and shown in Algorithm 12.

Algorithm 12 Slice sampling

```
 1: procedure SliceSampling($n_{\max,\mathrm{xp}}$, $\boldsymbol{s}$)
 2:     pick initial point $\boldsymbol{x}_0$                     ▷ generate or use the latest found point
 3:     $y \leftarrow \mathcal{U}(0, w(\boldsymbol{x}_0))$                                    ▷ draw slice level
 4:     $\boldsymbol{h}_l \leftarrow \boldsymbol{x}_0 - \mathcal{U}(\boldsymbol{0}, \boldsymbol{s})$          ▷ place hyperrectangle $\boldsymbol{h}$ randomly around $\boldsymbol{x}_0$
 5:     $\boldsymbol{h}_u \leftarrow \boldsymbol{h}_l + \boldsymbol{s}$
 6:     inbound ← True
 7:     while $(n_{\mathrm{xp}} \leq n_{\max,\mathrm{xp}}) \wedge$ inbound do          ▷ expand until the slice is covered
 8:         if $w(\boldsymbol{x}_i) < y$ then
 9:             inbound ← False
10:         else
11:             expand $\boldsymbol{h}$
12:     outside ← True                                         ▷ hyperrectangle found
13:     while outside do                          ▷ until a point inside the slice is found
14:         $\boldsymbol{x}_i \leftarrow \mathcal{U}(\boldsymbol{h}_l, \boldsymbol{h}_u)$              ▷ pick a random point from the hyperrectangle
15:         if $w(\boldsymbol{x}_i) \geq y \wedge$ acceptable then   ▷ evaluate weight function and acceptability
16:             outside ← False
17:             $\boldsymbol{x}_0 \leftarrow \boldsymbol{x}_i$              ▷ set the newly found point as the next initial point
18:             return $\boldsymbol{x}_i$                        ▷ stop the algorithm for this iteration
19:         else
20:             shrink $\boldsymbol{h}$
```

As it being a *MCMC* method, it needs a starting state, which is the point x_0. The starting state may be given by the user or picked arbitrarily from the design space. Then, the weight of the current state is evaluated. After that, a weight level y is drawn uniformly between zero and the just calculated weight of the current state. Similar to an α-cut, see Equation (2.1), the set of points, whose weight is at least y, is called *slice*. The next task is to find a fitting hyperrectangle $\boldsymbol{h}$, which preferably covers the whole slice. After a finding such a hyperrectangle is found, points are drawn uniformly from it, until a satisfactory point is found. For each point, the weight function is evaluated and compared to the slice level y. If the candidate is found outside the slice, shrinking might be applied to the hyperrectangle. A candidate is found inside the slice is accepted, if no further necessary checks are violated and becomes the new state of the *MCMC*.

The *stepping-out procedure* takes the initial step size and expands this first into one direction, for example to the left, until the boundary is found outside slice. Then, it expands to the other side analogously. It needs to be pointed out, that the proportion, how many expansion to either side can be taken is randomly assigned beforehand. Thus, if the slice is wider than $n_{\text{max, xp}} \cdot \boldsymbol{s}$, the final interval is still placed randomly.

The *doubling procedure* expands the interval by its own length in a random direction, until both boundary points are outside the slice. Since it is possible, that the slice is concave, another part of the slice could be found, from which the initial point can not be reached. Thus, an additional test must be carried out before excepting the new point.

If the drawn candidate is not acceptable, a new one needs to be tried. Shrinking the searching hyperrectangle enhances the efficiency by increasing the chance, finding an acceptable one. Although it is not compulsorily, it is highly recommended to use. The simplest approach is to shrink all components to the rejected candidate. Thus, making it a corner of the search hyperrectangle, that contains the initial point $\boldsymbol{x}_0$. Advanced algorithms may abuse the gradient or other know attributes of the weight function to enhance the efficiency further by suppressing small steps. Since the search hyperrectangle always contains and shrinks towards $\boldsymbol{x}_0$, it is guaranteed to terminate with a valid new state point $\boldsymbol{x}_i$.

Multiple approaches to multivariate slice sampling are possible:

- omitting the expansion and immediately start shrinking from a sufficiently large initial hyperrectangle,
- breaking the n-dimensional space into n one-dimensional axes, originating at the initial point and expand along these individually, recombine them after finding an interval on each axis,
- creating random walk, as only updating one component at any given step [91, p. 3],
- expanding in all directions at the same time,
- advanced direct multivariate expansion techniques, expanding a hyperrectangle in various directions.

In [64; 65] the previously explained algorithm is described for only one dimension in detail. But as pointed out, for multidimensional design spaces the procedure may be more costly. It is emphasized, that the randomness is crucial for the methods correctness. In $\mathbb{R}^1$ the proportion of expansions to either side are assigned randomly in-before the actual expansion. In [65, p. 722] is pointed out, that even though it is desirable to find the smallest interval (hyperrectangle), that contains the whole slice, one could also be contend with a hyperrectangle, that contains a part of the slice. Taking a sufficiently big initial interval, omitting the expansion and directly start shrinking will keep the procedure valid. For a more sophisticated, than this naive approach, more actions need to be taken. Early experiments showed, that when using too small initial hyperrectangles the method tend to get stuck in a particular region of the design space, which is not desirable for obvious reasons. In case, that not all corners are required to be outside the slice, the initial hyperrectangle must not be too small [65, p. 722]. Picking a bigger initial size is general on the safe side. For bounded design spaces it could be the whole design space, but exceeding the design space is not worthwhile. Cropping the hyperrectangle at the border of the design space is valid, since possibly drawn points outside the design space would be rejected in any case and shrinking would be applied. Approaches to multivariate Slice Sampling are given in [48; 61; 64; 65; 67; 90; 91; 92].

2.5.2 Evolu

According to [55; 56; 57; 58; 59; 60], Evolu is a modified genetic algorithm. It is described as hybrid of random-walk, hill-climb and Monte-Carlo-Sampling (*MCS*), in a phase of lacking improvement. The procedure of a single optimization chain is shown in Algorithm 13. The search region is a hyperrectangle around the parent point. In

Algorithm 13 evolu

```
1: procedure EVOLU($n_{\text{off}}, n_{\text{ref}}$)
2:     set search region
3:     $n_{\text{ref}} \leftarrow 0$
4:     $x_{\text{P}} \leftarrow x_0$                          ▷ draw starting point from design space
5:     while $j < n_{\text{ref}}$ do
6:         while $i < n_{\text{off}}$ do
7:             $x_i$: draw offspring              ▷ from a region around the parent-point
8:             Bring back to the feasible region, if ended up outside
9:             check vicinity            ▷ prevents computing of nearly identical points
10:            if is_valid($x_i$) then                          ▷ Checking constraints
11:                $z_i \leftarrow z(x_i)$                    ▷ evaluate fundamental solution
12:                if $z_i \prec z_{\text{P}}$ then                   ▷ improvement achieved
13:                    $x_{\text{P}} \leftarrow x_i$            ▷ offspring becomes new parent-point
14:                    $i \leftarrow 0$                          ▷ reset offspring counter
15:            else
16:                $i \leftarrow i + 1$
17:        refine search region
18:        $n_{\text{ref}} \leftarrow n_{\text{ref}} + 1$
19:    post-computation to ensure convexity
```

refinement stages, the algorithm shows recursive behavior by reducing the search region's size. Efficiency enhancements can be achieved, if the starting point is picked according to knowledge about already computed points. The individual chains are embarrassingly parallelizable.

2.5.3 Sequential Weighted Sampling (*SWS*)

Sequential Weighted Sampling (*SWS*) as described in [49] is an advanced *MCMC* based genetic sampling algorithm. The aim is denser sampling in promising areas than in less promising ones. This is achieved by assigning a *pdf* given in Equation (2.32) to the sampling space, that is rating regions closer to contributing points higher. For a sample $\boldsymbol{x}$ and the contributing data points P_j and its location in design space $P_{\boldsymbol{x},j}$ in all Pareto-fronts $\mathbb{P}$, the weighting function is given as

$$w = \max_{P_j \in \mathbb{P}} \exp\left(-\delta(\boldsymbol{x} - P_{\boldsymbol{x},j})^2\right) \mathbb{1}_{\mathcal{A}}(\boldsymbol{x}) \tag{2.32a}$$

with the characteristic function

$$\mathbb{1}_{\mathcal{A}}(\boldsymbol{x}) = \begin{cases} 1, & \text{if } \boldsymbol{x} \in \mathcal{A} \\ 0, & \text{else} \end{cases} \tag{2.32b}$$

to restrict the sampling space to the hyperrectangle of the design space $\mathcal{A}$. The parameter δ scales the falloff of the weight function. Since this *pdf* may become a difficult distribution, for which an analytical cumulated density function (*cdf*) and *icdf* might be nonexistent, a sampling technique able to sample from *pdf* is necessary. Slice sampling, described in Section 2.5.1, is such a technique and will be used in this thesis.

The algorithm is as shown in Algorithm 14. The data structure P_i are data points, holding the coordinates in design space $\boldsymbol{x}_i$, objective space $\boldsymbol{z}_i$ and its membership μ_i. The index i indexes the list of data points, samples and solutions, not the sample's components. Samples from the design space are denoted as $\boldsymbol{x}_i$. They are of the same dimension as $\mathcal{A}$, which can be of any dimension. A data point P is considered contributing, if it is a member of one of the PARETO-fronts $\mathcal{P}$. The fundamental solution is denoted as $z()$, and the membership function as $\mu()$.

Algorithm 14 Sequential weighted Sampling

```
 1: procedure SWS($n_{\text{init}}, n_{\text{generations}}, n_{\text{iter}}, \delta$)
 2:     $\boldsymbol{x}_i$: draw $n_{\text{init}}$ initial points          ▷ distribute over design space
 3:     $P_i \leftarrow (\boldsymbol{x}_i, z(\boldsymbol{x}_i), \mu(\boldsymbol{x}_i))$   ▷ evaluate fundamental solution and membership
 4:     sort $P_i$ into PARETO-fronts
 5:     store $P_i$
 6:     for $n_{\text{generations}}$ do
 7:         update $w$                     ▷ implicitly done in the implementation
 8:         $\boldsymbol{x}_i$: draw $n_{\text{iter}}$ new samples using Slice Sampling     ▷ see Section 2.5.1
 9:         $P_i \leftarrow (z(\boldsymbol{x}_i, \boldsymbol{x}_i), \mu(\boldsymbol{x}_i))$   ▷ evaluate fundamental solution and membership
10:         sort $P_i$ into PARETO-fronts
11:         store $P_i$
12:     return an empiric fuzzy quantity
```

The initial population is drawn by traditional MONTE-CARLO (*MC*)-Sampling from an uniform distribution over the design space. Other space-filling techniques, such as Latin Hypercube Sampling, see [52] or SOBOL-patterns, see [85] may be implemented as well. By passing a set of points to be the initial generation, the initial *MC*-Sampling can be skipped. Those data points can be obtained in a previous run, thus being located in a promising area.

After evaluating the n_{init} initial points' fundamental solution and membership, they are stored and sorted into the set of PARETO-fronts, see Section 2.4.3. For each $n_{\text{generations}}$ generations n_{iter} points are drawn according to the distribution function w, given in Equation (2.32). Since it is iterated over all points in the set of PARETO-fronts, the function is not updated explicitly, but with each update to a PARETO-front implicitly. After the sampling of new points, all of those are evaluated, stored and queried against all PARETO-fronts again. In the process, w evolves in each generation.

To inspect the convergence behaviour of *SWS*, a SHEKEL-equation according to [70, p. 8] is used as a benchmark function. The function is defined as

$$f(\boldsymbol{x}) = -\frac{0.1}{0.14 + 20\left((x_1 - 0.45)^2 + (x_2 - 0.55)^2\right)}. \tag{2.33}$$

A global minimum is found at $f\,(0.101\,714\,22, 0.101\,003\,89) = -1.015\,106\,55$ by *SWS* with a total of 1×10^6 samples. A local minimum is present at $(0.45, 0.55)$. The convergence with $n_{\text{init}} = 1$, $n_{\text{generations}} = 1\,000$, $n_{\text{iter}} = 1$ on this function is shown in Figure 2.7. Plotted are the minimum (green), mean (black) and maximum (red) relative error across 100 independent optimization chains and behavior of a single chain (blue) over the course of 1 000 generations. The relative error is calculated to $\left|\frac{\text{min}_{\text{current}} - \text{min}_{\text{abs}}}{\text{min}_{\text{abs}}}\right|$. Due to the undirected sampling, the improvement does not have a steady rate but happens in separate events. The algorithm may not yield an improvement for an unforeseeable number of generations an the improvements gains itself are not predictable. Thus, a conversion criterion is not easily determinable, although for this example function convergence is achieved consistently at roughly 500 fundamental solution evaluations.

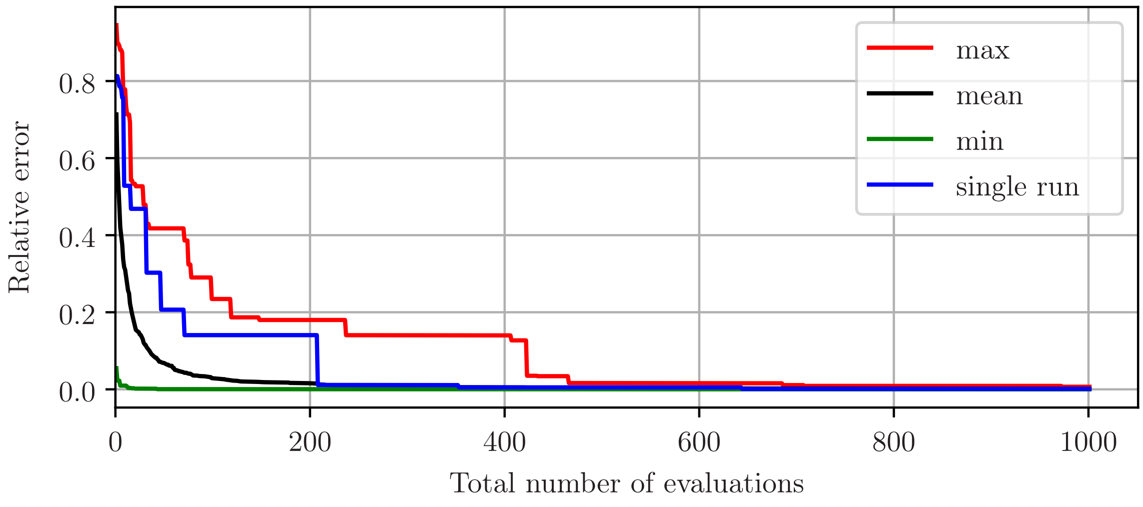


Figure 2.7: Convergence of *SWS*.

2.5.4 Comparison of Evolu and SWS

Evolu finds points also directly on the border of the design space. *SWS* usually does not find a point directly on the design space border due to the non-directed *MC* nature around a contributing single point. As pointed out in [49, p. 8], other optimization strategies may be more efficient, if the problem is monotone or gradients are known. While Evolu can only find as many members of a point, as chains are run, the member count after a run of *SWS* is not easily predictable and may depend on the problem.

Across various runs of *SWS* on the example given in Section 4.1, of 2 200 total evaluations, consistently roughly 400, give or take a few dozen, ended up in final solution. This makes the efficiency measure, given in Equation (2.29) approximately $p_{\text{eff}} \approx 0.2$. In comparison, a crude *MC*, without evolutionary procedure achieved barely $p_{\text{eff}} \approx 0.1$. This can be considered a doubling in efficiency.

3 PUQpy – Structure and Uncertainty Analysis

In this chapter, the general objective and structure of PUQpy, which stands for "Polymorphic Uncertainty Quantification in Python" is laid out. After that, analysis of uncertainty problems with PUQpy is discussed.

3.1 Requirements and Objective

PUQpy is a framework for monomorphic and polymorphic uncertainty analysis. First demand is a flexible structure, that makes it possible to combine arbitrary analyses, as the problem requires. Thus, a modular approach is being taken. The need of generic data types and interfaces to analyses and objects arises, analyses are to be nested in every possible combination. As assumed by experience, the examined problems are computational expensive, efficient strategies need to be applied in order to give the best result with the least fundamental solution evaluations possible. The flexibility, which the framework is aiming for, is suggesting a script based workflow. Thus a project is built by writing a Python script. This script defines the procedures and subjects of investigation. Incorporation with other packages is possible.

The software framework PUQpy accompanying this work is implemented in Python3 due to its user friendliness and flexibility. Python3 is a high level general purpose programming language. It supports object oriented programming as well as functional programming. It comes with a wide variety of well developed, maintained and documented libraries, such as numpy, see [38] and scipy, see [99]. Therefore, it is used widely, from system administration tools to computational intense programming, also in scientific computation, from small scripts to complex end-user software. Python code is easy to develop and to read, thus beginner friendly, as well as powerful. Pure Python code is platform independent, thus usable on all operating systems [95; 96; 97].

Due to its highly dynamic nature and the associated overhead, Python is slow in low-level numerical calculation. With the programming language Cython, it is possible to use both the flexibility of Python, as well as the speed of C code [9]. This is made possible by compiling Python code into plain C code, Cython takes care of the conversion. It is possible to gradually transform Python code by using Cython syntax, eventually obtaining native C code. Significant speedup is gained in numerical loops by statically typing variables, as done in C.

The documentation of the project is generated using doxygen [94]. The automatic generation of documentation from source code eases the burden of writing and keeping the reference up to date, even in a rapidly changing code base structure. doxygen supports a wide range of programming languages.

3.2 Structure

Modeling of uncertainty can be done in three ways. The first models uncertain quantities and the necessary analyses are implicitly derived from the given quantities. Shortcoming is the lack of control in the applied analysis methods. As well as the procedural structure cannot be easily deduced from the script.

The second is to define a set of stacked analyses and quantities are derived implicitly through the analyses' structure. Drawback is the obscure data structure, which is not evident to the reader of a script.

The third is a combination of the previous ways. Quantities and analyses are explicitly defined. Quantities are assigned to analyses as their matter. The explicitness makes it easy to follow the structure of a projects workflow and data structure. Quantities and analyses are tightly coupled and depend on each other. Quantities are defined as generic as possible to serve as input quantities and output quantities. It is possible to use output quantities as input quantities to compatible analyses.

The framework is organized into the following modules.

- `Analysis`, see Section 3.3,
- `Distributions`, see Section 3.7,
- `Interdependencies`, see Section 3.4,
- `Library_miscellaneous`
- `Optimization`, see Section 3.5,
- `Quantity`, see Section 3.6,
- `QuantityEpistemic`, see Section 3.6,
- `QuantityAleatoric`, see Section 3.6,
- `Sampling`, see Section 3.7.

Each of those is explained subsequently. To differentiate between the general terms and a class or an instance of the class, the latter ones are set in `teletype`, `Classes` are capitalized, `attributes` and `methods()` are lowercase, with methods having parentheses.

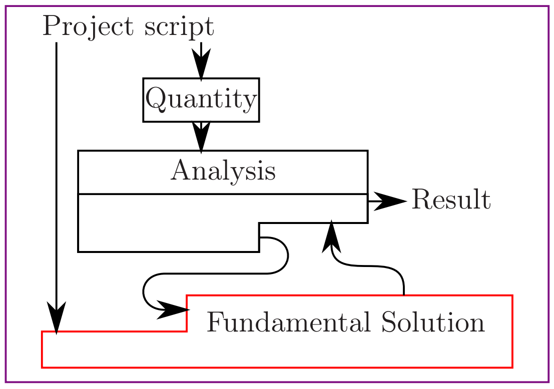
3.3 Module: `Analysis`

To analyze uncertainty, the following three major classes are necessary:

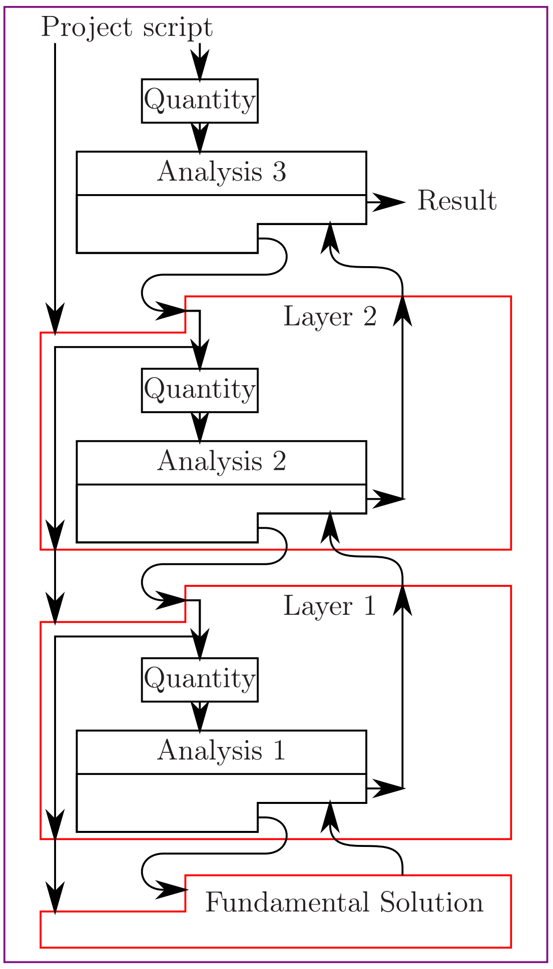
1. `UncertaintyAnalysis`, see Section 3.3.1,
2. `Layer`, see Section 3.3.2,
3. `FundamentalSolution`, see Section 3.3.3.

The uncertainty analyses are divided into two separate modules for fuzzy (`AnalysisFuzzy`) and stochastic (`AnalysisStochastic`) analyses. For fuzzy analysis, `ALO` for α-level-optimization and `SWSForFuzzy` as α-level-free method are implemented in PUQpy. Inheritance of sub-classes of `Analysis` is shown in Figure 3. In this framework only monomorphic analyses are implemented. By using `Layer` objects, polymorphic analyses are assembled by nesting monomorphic ones. This modular structure is especially useful, because the highly specialized uncertainty analyses can be put together arbitrarily. Therefore, it is possible for the user to customize the whole structure to fit perfectly for the project. A given analysis has exactly a single child and a single parent. `Layer` and `Analysis` interlock by the exposed interfaces. An `Analysis` must be embedded in a `Layer` and must call a

`Layer`. Project scripts and `FundamentalSolution` fulfill the role of a `Layer`. In Figure 3.1 the structure of an uncertain analysis project is shown, `Layer` frames are drawn in red. In Figure 3.1(a), the project structure of a monomorphic analysis is shown. In Figure 3.1(b), the project structure of a polymorphic analysis with three analysis layers is shown.



(a) Scheme of a monomorphic analysis.



(b) Scheme of a polymorphic analysis.

Figure 3.1: Schemes of a monomorphic and a polymorphic analysis.

3.3.1 Class: `UncertaintyAnalysis`

An analysis has the following major components:

- uncertain quantity,
- sampling method,
- fundamental solution,
- heuristic technique.

In Figure 3.2 a schematic of the `Analysis` class is shown. The uncertain quantity is the subject-matter of the analysis. An `UncertaintyAnalysis` object holds exactly one `quantityobject`. If the `quantityobject` changes or is replaced, the `UncertraintyAnalysis` must be reinitialized, that is reset. Otherwise stored data, that is derived from the previous quantity would contaminate the run with the next quantity.

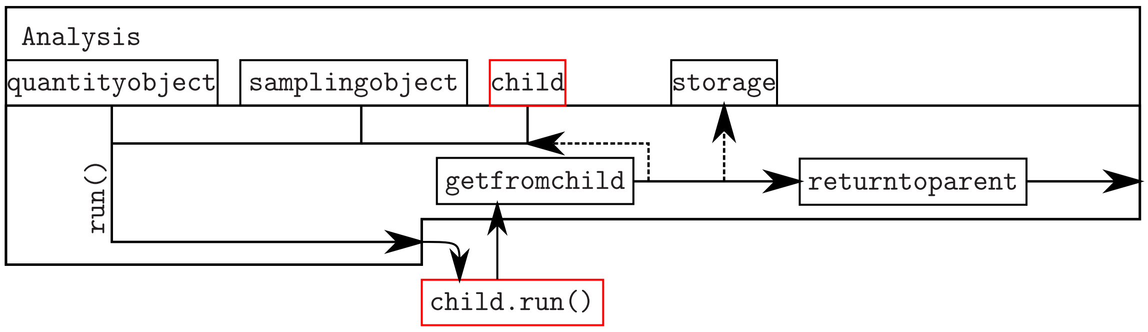


Figure 3.2: Data flow and management inside `UncertaintyAnalysis` object.

Various sampling methods are implemented in PUQPY to generate samples. These deterministic samples are evaluated by the fundamental solution or processed further by a subordinated `Layer`. The heuristic is used to aid and guide the sampling algorithm to draw points more efficiently. Evolutionary algorithms combine heuristics and sampling. These may be implemented in separate modules and communicating with each other via interfaces or merged into a single routine. This modular approach makes high scalability and flexibility possible.

Instances of `UncertaintyAnalysis` and `Layer` are aimed to be nested alternating. The `child` is the matter of its parent `UncertaintyAnalysis` and will be called repeatedly by the `run()` method of the `UncertaintyAnalysis`. The parent `Layer` object provides its `UncertaintyAnalysis` object with a `quantityobject` and a `child` and gathers the result of the analysis. A `child` must be either a `FundamentalSolution` or a `Layer`. The `run()` method exposed by `UncertaintyAnalysis`, expects no arguments. Since analyses generate much data, non-vital results may be either discarded or saved for later use, after the analysis terminated. Data required for the operation of the analysis are held in memory. After the analysis loop of sampling an evaluation is finished, a `Quantity` object of the same nature as the `Analysis` is instantiated and returned. Instances of `UncertaintyAnalysis` expose interfaces with data types as shown in Table 3.1. `UncertaintyAnalysis` objects are always defined inside a `Layer`. Since the project script serves as a `Layer`, the outermost `Analysis` object, can be defined directly in the main project script.

Table 3.1: Interface data types of `Analysis`.

	Receive from	Pass to
Child `Layer`	deterministic array	deterministic array
Parent `Layer`	`Quantity` object	`Quantity` object

Example 3.1: To illustrate the concept, consider an example. A research project investigates the behavior of concrete. They are especially interested in the mixtures impact on the strength. The uncertain design quantity is therefore the mixture ratio of aggregate, cement, water and additives. Manufacturing a set of probes can be seen as sampling. Measuring the strength of the probes in experiments is the fundamental solution, which yields a set of data. Heuristics are used to readjust the procedure, if new knowledge becomes available. The conclusion is done after the experiment, such as establishing design rules.

3.3.2 Class: `Layer`

The `Layer` class is used for

- quantity construction,
- analysis management,
- uncertainty reduction and
- data storage.

`Layer` objects manage the subordinated `Analysis` objects with their `Quantity` objects. Thus, they are the connecting pieces between analyses, see Figure 3.1(b). The next subordinated `Layer` is known to the current `Layer` as `child`. In Figure 3.3 a schematic of the `Layer` class is shown.

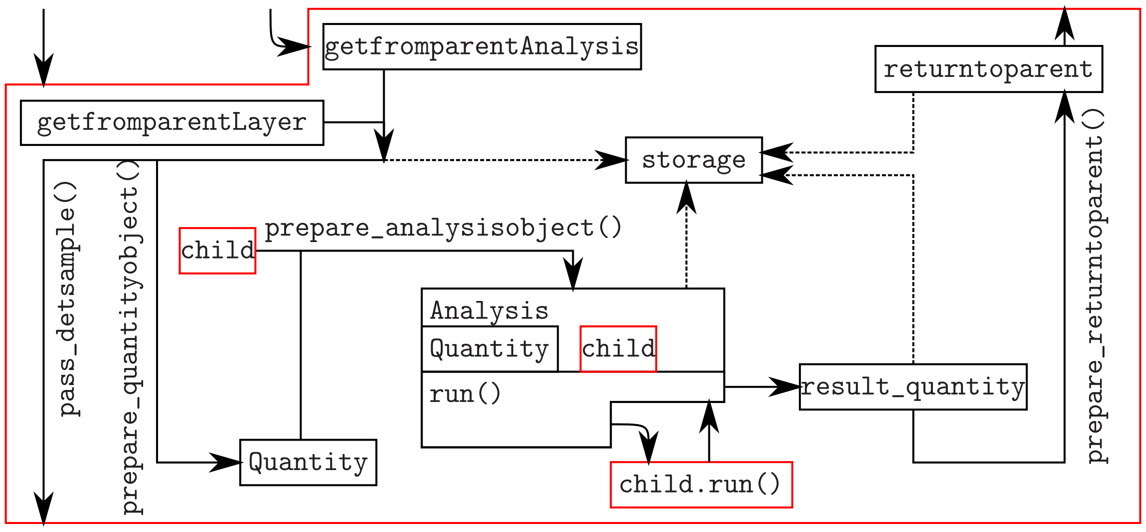


Figure 3.3: Data flow and management of `Layer.run()`.

The `run()` method exposed by `Layer` and `FundamentalSolution` objects expects and returns a deterministic array. In Algorithm 15 the succession of `Layer.run()` is shown, which is always the same. The individual methods are to be implemented by the user for the specific use case. They cannot be given universally due to the wide variety of possible use cases. It is highly recommended to re-implement the necessary methods in a sub-class of `Layer` inside the project script.

Algorithm 15 Succession of the calls inside the `.run()` method

```
1: procedure RUN(getfromparentAnalysis)
2:     self.getfromparentAnalysis = getfromparentAnalysis
3:     self.prepare_quantityobject()
4:     self.prepare_analysisobject()
5:     self.pass_detsample()
6:     self.result_quantity = self.analysisobject.run()
7:     self.returntoparent = self.prepare_returntoparent()
8:     self.store_result()
9:     return self.returntoparent
```

Firstly, the argument passed to the `run` method is made available object-wide to the other methods through writing it to the attribute `getfromparentAnalysis`. Then a `Quantity` object is constructed in the method `prepare_quantityobject()` and stored into the attribute `quantityobject`. The modules `QuantityEpistemic` and `QuantityAleatoric` provide some classes for such quantities (see Section 3.6). It is preferable to implement modifications in sub-classes inside the project script.

After that, the method `prepare_analysisobject()` is used to define an analysis. Already implemented classes can be found in the modules `AnalysisFuzzy` and `AnalysisStochastic` (see Section 3.3). If the analysis depends on drawing samples from the quantity, a sampling object is needed, as provided by the module `Sampling` (see Section 3.7). The previously initialized objects for sampling and the `quantityobject` are passed to the constructor. The associated `Analysis` instance is initialized and put into the attribute `analysisobject` of the `Layer` object. As the `Layer` and therefore the subordinated `Analysis`, is called repeatedly, the `Analysis` object is reset or overwritten repeatedly.

Since analyses can only work on a single monomorphic quantity, deterministic samples cannot be tunneled through `Analysis` objects. Hence, those deterministic samples can be branched off to be bypassed until being used further down the stack. This is useful for a component from a superior analysis, that is not being used in the current analysis, but in the fundamental solution or in a subordinated `Layer`. Deterministic samples are passed to the subordinated `Layer` object by the method `pass_detsample()`. The data is stored in the `getfromparentLayer` attribute, before invoking the `run()` method.

Running the analysis is done by invoking `analysisobject.run()`. The result of the analysis is assigned to attribute `result_quantity`. Since the calling analysis expects a deterministic array to be returned, the post-processing of the result quantity is to be implemented in `prepare_returntoparent()`. Thus the result quantity of the analysis needs to be reduced to a deterministic array. For stochastic quantities, this may be quantile, a moment of the distribution or any other characteristic value, for fuzzy quantities some defuzzification methods are discussed in Section 2.2.6.

Finally, the prepared deterministic result is returned to the calling analysis. If enabled, all relevant data is being stored for possible later use.

To be able to run consecutive analyses with different settings for the `Analysis` object, post-processing or other settings, a `Layer` provides the argument `additionalsettings` to pass those as a list.

Types of data passed through exposed interfaces of the `Layer` module are shown in Table 3.2. The main project script and the fundamental solution serve as a `Layer` object. Both share

Table 3.2: Interface data types of a `Layer` object.

	Receive from	Pass to
Child `Analysis`	`Quantity` object	`Quantity` object
Parent `Analysis`	deterministic array	deterministic array
Subordinated `Layer`	None	deterministic array
Superior `Layer`	deterministic array	None

the same interfaces, with the exception, that the former does not need to report to a superior `Analysis` object and the latter has no subordinated one. All `Layer` are defined toplevel in the project script and are connected, that is stacked, by setting the `child` attribute of their `Analysis`. The initializing order of `Layer` and `Analysis` objects is innermost to outermost.

3.3.3 Class: `FundamentalSolution`

The class `FundamentalSolution` defines a wrapper around the model for the basic deterministic problem. In Figure 3.4 a schematic of the `FundamentalSolution` class is shown. Fundamental solutions mark the innermost `Layer` in the analysis stack. The basic model

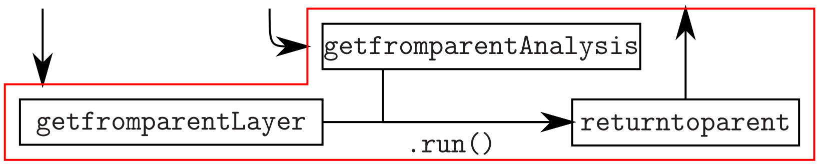


Figure 3.4: Data flow scheme inside a `FundamentalSolution` object.

itself can be arbitrarily complex. It can range from a simple analytic equation to a complex Finite-Element-Method (*FEM*). A `FundamentalSolution` object is called by an `Analysis` object as its `child`, but has no `child` itself. As a `FundamentalSolution` object serves as a `Layer` object, its `run()` method expects and returns a deterministic array. Types of data passed through exposed interfaces of the `FundamentalSolution` module are shown in Table 3.3.

Table 3.3: Interface data types of a `FundamentalSolution` object.

	Receive from	Pass to
Parent	deterministic array	deterministic array
Superior `Layer`	deterministic array	None

3.4 Module: `Interdependencies`

In the module `Interdependencies`, classes for interaction, constraints and correlation between input quantities are implemented. Correlation of stochastic quantities is not in the scope of this work.

A constraint may be used to define whether areas in design space are permissible or invalid combinations. In essence, a constraint holds information, on how one or more quantities interact. It consists of references to the involved quantities and an admissibility function. This function states, whether a combination of design space coordinates is legal or illegal. Only data points within the permissible areas are allowed to be generated and passed to the child.

Interactions are used to limit the membership of a combinations between fuzzy quantities realizations. On a specific α-level, including the support, an interaction becomes a constraint for the optimization. Inheritance for classes in `Interdependencies` is shown in Figure 2.

3.5 Modules: `Optimization` and `OptimizationPareto`

In this module, the class definitions of optimization algorithms, such as *SWS*, see Section 2.5.3, are located. Class definitions for PARETO-fronts, on which the optimization routines are dependent, are implemented in the module `OptimizationPareto`. Section 2.4 is dedicated to discuss this topic in detail.

3.6 Modules: `Quantity`, `QuantityEpistemic`, `QuantityAleatoric`

Ih these modules, classed for deterministic and uncertain quantities are implemented. In the module `Quantity`, the base class `Quantity` and deterministic quantities are implemented. `QuantityEpistemic` serves as the base class for fuzzy quantities. Fuzzy quantities and their properties are described in detail in Section 2.2. `QuantityAleatoric` serves as the base class for stochastic quantities. Most of them are built to estimate a failure probability. The implementation and conception of stochastic quantities is not scope of this work, but its sibling work [81]. Inheritance graphs of classes implemented in PUQpy are shown in Figure 4.

3.7 Modules: `Sampling` and `Distributions`

Sampling algorithm are found here. Sampling is the procedure of drawing a number of deterministic samples from a distribution. Slice Sampling is described in Section 2.5.1 and implemented in `SliceSampling`. The weight function of *SWS* is implemented in the sub-class `SliceSamplingForSWS`. More sampling algorithms implemented in PUQpy are documented in [81]. Inheritance of sub-classes of `Sampling` is shown in Figure 1.

In the module `Distributions`, stochastic distributions are defined. These are wrapper classes around SCIPY classes and documented in [81].

3.8 Distributed Computing

As the tasks to be done scale horrendously, computing on a single machine, even with parallel usage of all available cores may not be sufficient. Thus, distributed computing is necessary. This is to utilize many cores on many machines across a network. Distributed software becomes complex easily and the requirements for it are heavy. There are quite a lot of frameworks available to be used with Python. To be considered usable for this project, the framework for distributing the work has to provide

- easy set up (user friendly),
- support for any number of machines with each any number of processors and cores, heterogeneous architecture,
- plugging in and unplugging machines at any time,
- cross platform (Linux, Windows, Mac),
- load scheduling/balancing,
- monitoring,
- low bandwidth usage,
- low management overhead,
- secure network traffic,
- low to moderate effort to integrate with PUQpy.

Since it fulfills all termed requirements, `dask` is used [15; 72].

To parallelize the computation on `Layer` level, subordinated `Layer` objects can be run in concurrency by calling the individual `Layer` object's `run` method in parallel. On the other hand some part of an analysis' body can be run concurrently. Combination of both are possible.

3.9 Uncertainty Analysis in PUQpy

Fuzzy Analysis Fuzzy analysis is used to examine possibilistic and epistemic uncertainty. This is one of the two most simple use cases, due to its monomorphic nature. No `Layer` object is needed, since the project script serves as the `Layer`, but the general sequence is the same as described in Section 3.3.2. The first step is to define a fundamental solution. Best practice is to implement the actual model of the problem as a sub-class of `FundamentalSolution` (see Section 3.3.3). Then, a fuzzy quantity is to be constructed, using either classes from the module `QuantityEpistemic` (see Section 3.6) or sub-classing one. Some analyses depend on sampling methods, as `SWS` depends on `SliceSamplingForSWS`, which are provided by the module `Sampling`, see Section 3.7. To modify a method, sub-classing is preferable. After that, a fuzzy analysis is to be defined. Already implemented analyses can be found in the module `AnalysisFuzzy`. The initialization happens according to the dependencies. First a `FundamentalSolution` object, then the fuzzy `Quantity` object and if needed the `Sampling` object are initialized, before they are passed to the constructor of the fuzzy `Analysis` object. Then the `run()` method of `Analysis` object is invoked. After termination, the resulting quantity may be post-processed, according to the goals of the study. The general structure of a fuzzy analysis is shown in Figure 3.5.

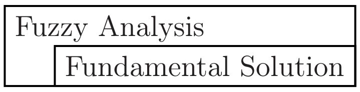


Figure 3.5: Structure of a fuzzy analysis.

Stochastic Analysis Stochastic analysis is used to examine probabilistic and aleatoric uncertainty. This is one of the two most simple use cases, due to its monomorphic nature. No `Layer` object is needed, since the project script serves as the `Layer`, but the general sequence is the same as described in Section 3.3.2. The first step is to define a fundamental solution. Best practice is to implement the actual model of the problem as a sub-class of `FundamentalSolution`, see Section 3.3.3. Then a stochastic quantity is to be constructed, using either classes from the module `QuantityAleatoric`, see Section 3.6, or sub-classing one. Sampling methods to draw from a stochastic quantity are provided by the module `Sampling`, see Section 3.7. To modify a method, sub-classing is preferable. After that, a stochastic analysis is to be defined. Already implemented classes can be found in the module `AnalysisStochastic`. The initialization happens according to the dependencies. First, a `FundamentalSolution` object, then the stochastic `Quantity` object and the `Sampling` object are initialized, before they are passed to the constructor of the stochastic `Analysis` object. Then the `run()` method of `Analysis` object is invoked. After termination, the resulting quantity may be post-processed, according to the goals of the study. The general structure of a stochastic analysis is shown in Figure 3.6.

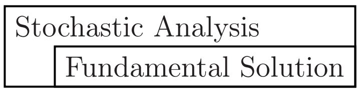


Figure 3.6: Structure of a stacked fuzzy analysis.

Fuzzy-based Fuzziness (*ff*) Fuzzy-based fuzziness (*ff*) analysis is used, when modeling the properties of a fuzzy quantity as fuzzy quantities. In PUQpy, this type is composed by using a fuzzy analysis as the fundamental solution to a fuzzy analysis. At least one `Layer` object is necessary, if the project script acts as `Layer` around the outer fuzzy analysis. For clarity, each uncertainty analysis is embedded in its own `Layer`, using two `Layer` objects. The first step is to define a fundamental solution. Best practice is to implement the actual model of the problem as a sub-class of `FundamentalSolution`, see Section 3.3.3. Then, a `Layer` for each analysis is defined by sub-classing `Layer`, as provided by the module `Analysis`, see Section 3.3.2. For each of the two `Layer` objects, the general sequence as described in Section 3.3.2 is used. Since each `Layer` performs a full fuzzy analysis, Section 2.1.2 may be helpful for understanding. Post-processing is necessary in the inner `Layer` to return a deterministic array back to the superior analysis loop, while it is optional in the outer `Layer`. The fuzzy result quantity of the inner fuzzy analysis needs to be reduced to a deterministic value. Some methods for defuzzification can be found in Section 2.2.6.

After both `Layer` classes are set up, the initialization of objects happens according to the dependencies. In general, the `Layer` objects are initialized from the innermost to the outermost one. Therefore initialization order is

1. `FundamentalSolution` object,
2. `Layer` object for inner fuzzy analysis,
3. `Layer` object for outer fuzzy analysis.

Then, the `run()` method of the outer `Layer` object is invoked. After termination, the resulting quantity may be post-processed, according to the goals of the study. The general structure of a stacked fuzzy analysis is shown in Figure 3.7.

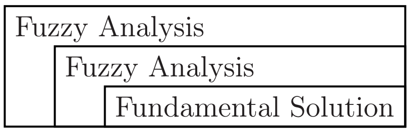


Figure 3.7: Structure of a stacked fuzzy analysis.

BAYESian Uncertainty BAYESian analysis is used, when modeling the properties of a stochastic quantity as stochastic quantities. In PUQPY, this type is composed by using a stochastic analysis as the fundamental solution to a stochastic analysis. At least one `Layer` object is necessary, if the project script acts as `Layer` around the outer stochastic analysis. For clarity, each uncertainty analysis is embedded in its own `Layer`, using two `Layer` objects. The first step is to define a fundamental solution. Best practice is to implement the actual model of the problem as a sub-class of `FundamentalSolution`, see Section 3.3.3. Then, a `Layer` for each analysis is defined by sub-classing `Layer`, as provided by the module `Analysis`, see Section 3.3.2. For each of the two `Layer` objects the general sequence as described in Section 3.3.2 is used. Since each `Layer` performs a full stochastic analysis, Section 2.1.1 may be helpful for understanding. Post-processing is necessary in the inner `Layer` to return a deterministic array back to the superior analysis loop, while it is optional in the outer `Layer`. The stochastic result quantity of the inner stochastic analysis needs to be reduced to a deterministic value. This may be a quantile, a moment of the distribution or any other characteristic value.

After both `Layer` classes are set up, the initialization of objects happens according to the dependencies. In general, `Layer` objects are initialized from the innermost to the outermost one. Therefore initialization order is

1. `FundamentalSolution` object,
2. `Layer` object for inner stochastic analysis,
3. `Layer` object for outer stochastic analysis.

Then, the `run()` method of the outer `Layer` object is invoked. After termination, the resulting quantity object may be post-processed, according to the goals of the study. The general structure of BAYESian analysis is shown in Figure 3.8.

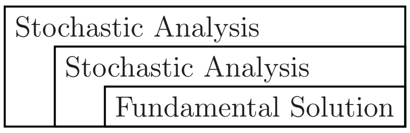


Figure 3.8: Structure of a BAYESian analysis.

Fuzzy Probability Based Randomness (*fp-r*) Fuzzy probability based randomness (*fp-r*) analysis is used, when modeling the properties of a stochastic quantity as fuzzy quantities. In PUQPY, this type is composed by using a stochastic analysis as the fundamental solution to a fuzzy analysis. At least one `Layer` object is necessary, if the project script acts as `Layer` around the outer fuzzy analysis. For clarity, each uncertainty analysis is embedded in its own `Layer`, using two `Layer` objects. The first step is to define a fundamental solution. Best practice is to implement the actual model of the problem as a sub-class of `FundamentalSolution`, see Section 3.3.3. Then, a `Layer` for each analysis is defined by sub-classing `Layer`, as provided by the module `Analysis`, see Section 3.3.2. For each of the two `Layer` objects the general sequence as described in Section 3.3.2 is used. Since the inner `Layer` performs a full stochastic analysis and the outer `Layer` performs a full fuzzy analysis, both Section 2.1.2 and Section 2.1.1 may be helpful for understanding. Post-processing is necessary in the inner `Layer` to return a deterministic array back to the superior analysis loop, while it is optional in the outer `Layer`. The stochastic result quantity of the inner stochastic analysis needs to be reduced to a deterministic value. This may be a quantile, a moment of the distribution or any other characteristic value.

After both `Layer` classes are set up, the initialization of objects happens according to the dependencies. In general, the `Layer` objects are initialized from the innermost to the outermost one. Therefore initialization order is

1. `FundamentalSolution` object,
2. `Layer` object for inner stochastic analysis,
3. `Layer` object for outer fuzzy analysis.

Then, the `run()` method of the outer `Layer` object is invoked. After termination, the resulting quantity may be post-processed, according to the goals of the study. The general structure of *fp-r*-analysis is shown in Figure 3.9 [84, p. 382].

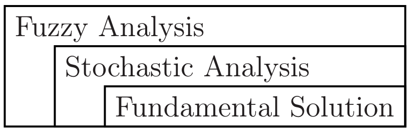


Figure 3.9: Structure of a *fp-r* analysis.

Fuzzy Randomness (*fr*) Fuzzy randomness (*fr*) analysis is used, when modeling the properties of a fuzzy quantity as stochastic quantities. In PUQpy, this type is composed by using a fuzzy analysis as the fundamental solution to a stochastic analysis. At least one `Layer` object is necessary, if the project script acts as `Layer` around the outer stochastic analysis. For clarity, each uncertainty analysis is embedded in its own `Layer`, using two `Layer` objects. The first step is to define a fundamental solution. Best practice is to implement the actual model of the problem as a sub-class of `FundamentalSolution`, see Section 3.3.3. Then, a `Layer` for each analysis is defined by sub-classing `Layer`, as provided by the module `Analysis`, see Section 3.3.2. For each of the two `Layer` objects the general sequence as described in Section 3.3.2 is used. Since the inner `Layer` performs a full fuzzy analysis and the outer `Layer` performs a full stochastic analysis, both Section 2.1.2 and Section 2.1.1 may be helpful for understanding. Post-processing is necessary in the inner `Layer` to return a deterministic array back to the superior analysis loop, while it is optional in the outer `Layer`. The fuzzy result quantity of the inner fuzzy analysis needs to be reduced to a deterministic value. Some methods for defuzzification can be found in Section 2.2.6.

After both `Layer` classes are set up, the initialization of objects happens according to the dependencies. In general, the `Layer` objects are initialized from the innermost to the outermost one. Therefore initialization order is

1. `FundamentalSolution` object,
2. `Layer` object for inner fuzzy analysis,
3. `Layer` object for outer stochastic analysis.

Then, the `run()` method of the outer `Layer` object is invoked. After termination, the resulting quantity may be post-processed, according to the goals of the study. The general structure of *fr*-analysis is shown in Figure 3.10.

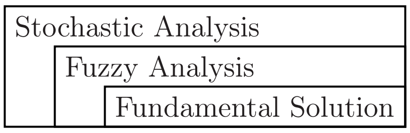


Figure 3.10: Structure of a *fr* analysis.

Fuzzy Probability Based Fuzzy Randomness (*fp-fr*) *fp-fr* analysis is the most complicated of all the shown polymorphic uncertainty analyses, as it encapsulates all of them as special cases. In PUQPY, this type is composed by stacking a fuzzy, a stochastic and a fuzzy analysis, which can be seen as a merge of *fp-r* and *fr*. At least two `Layer` object are necessary, if the project script acts as `Layer` around the outer fuzzy analysis. For clarity, each uncertainty analysis is embedded in its own `Layer`, using three `Layer` objects. The first step is to define a fundamental solution. Best practice is to implement the actual model of the problem as a sub-class of `FundamentalSolution`, see Section 3.3.3.

Then, classes for the `Layer` objects around the inner fuzzy analysis, the stochastic analysis in the middle and the outer fuzzy analysis are to be implemented. For each of those three `Layer` objects, the general sequence as described in Section 3.3.2 is used. Since the inner and outer `Layer` performs a full fuzzy analysis and the outer `Layer` performs a full stochastic analysis, both Section 2.1.2 and Section 2.1.1 may be helpful for understanding. Post-processing is necessary in the inner and middle `Layer` to return a deterministic array back to the superior analysis loop, while it is optional in the outer `Layer`. The fuzzy result quantity of the inner fuzzy analysis needs to be reduced to a deterministic value. Some methods for defuzzification can be found in Section 2.2.6. The stochastic result quantity of the stochastic analysis in the middle needs to be reduced to a deterministic value. This may be a quantile, a moment of the distribution or any other characteristic value.

After all three `Layer` classes are set up, the initialization of objects happens according to the dependencies. In general, the `Layer` objects are initialized from the innermost to the outermost one. Therefore initialization order is

1. `FundamentalSolution` object,
2. `Layer` object for inner fuzzy analysis,
3. `Layer` object for middle stochastic analysis,
4. `Layer` object for outer fuzzy analysis.

Then, the `run()` method of the outer `Layer` object is invoked. After termination, the resulting quantity may be post-processed, according to the goals of the study. The general structure of *fp-fr*-analysis is shown in Figure 3.11.

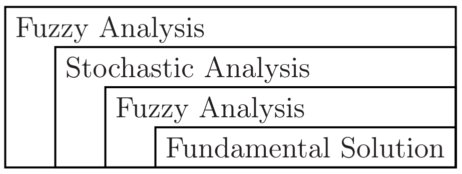


Figure 3.11: Structure of a *fp-fr* analysis.

4 Numerical Examples

4.1 Fuzzy Analysis with a Bivariate Fuzzy Input Quantity

In this section, flat α-level-optimization, non-flat α-level-optimization and α-level-free fuzzy analysis methods in PUQPY are compared based on the example given in [49]. As a fundamental solution serves

$$z = \phi(x_1, x_2) = (x_1 - 0.5)^2 + x_2. \tag{4.1}$$

The two-dimensional joint membership function is given by

$$\mu(x_1, x_2) = \min_{i=1,\,2} \max\left(4 \cdot \left(\frac{1}{2} - \left|x_i - \frac{1}{2}\right|\right)^2, 0\right). \tag{4.2}$$

Both functions are defined on the union square $[0, 1]^2$ design space. No further assumptions are made.

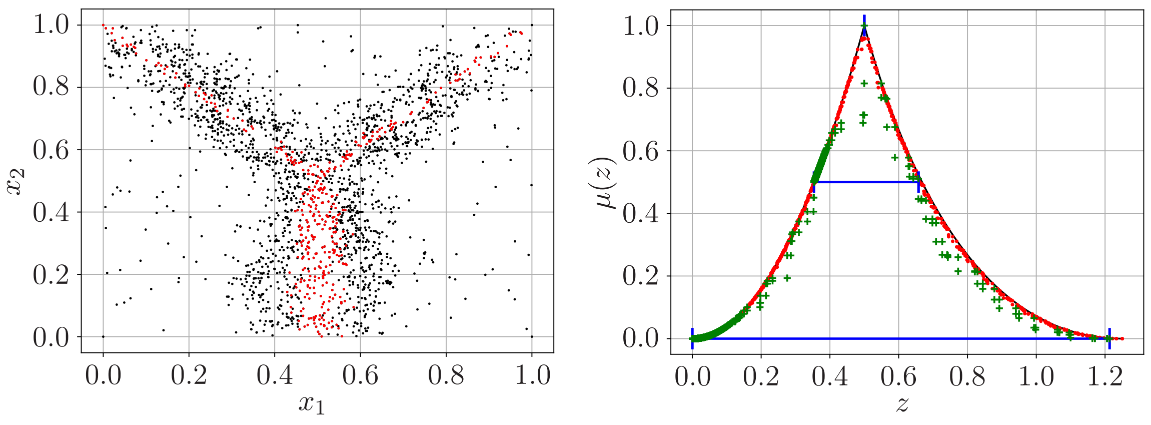
Three approaches are taken. *SWS*, as described in Section 2.5.3, with slice sampling, see Section 2.5.1, is used in all approaches. Slice sampling uses naive shrinking, but no expansion with the initial hyperrectangle size of $\boldsymbol{w} = (1.0, 1.0)$. As weight function the one given in Equation (2.32) is used with falloff factor $\delta = 300$. The results are compared based on the two α-levels 0.0 and 0.5. Since the core consists of a single data point, comparison on the core is not useful.

The first approach is flat α-level-optimization, each data point is assigned the nominal membership value of the α-level. On α-level 0.0 `SWS` with in 200 initial points and 2 000 generations with a single point is used. For α-level 0.5, data is imported and another 2 000 generations are carried out. For both α-level, in total 4 200 data points are evaluated. Objectives for the optimization are maximization and minimization of the fundamental solution. The α-levels are calculated in ascending order. After α-level 0.0, already calculated data points are imported to the optimizer object for the α-level 0.5.

The second approach is non-flat α-level-optimization. As in the first approach, it is optimized for both maximization and minimization objectives with the same settings. For better comparability, the exact same data points, as evaluated by the first approach are used. Therefore, the results based on the α-levels are identical and the difference is in-between the calculated α-levels. An actual run is emulated by reevaluating each data points' membership individually afterwards again.

In the third approach, α-level-free fuzzy analysis method is used. `SWSForFuzzy`, is initiated with 200 initial points, that are uniformly distributed over the union rectangle $[0,1]^2$. In each of the 100 generations carried out, 20 points are drawn. In total, the fundamental solution is evaluated 2 200 times. It is optimized for maximization of $\mu(z)$ and both maximization and minimization for z.

The results of the α-level-free method are shown in Figure 4.1. In Figure 4.1(a), the analytical solution is drawn as a black solid line. Intervals yielded by the flat α-level-optimization are drawn in blue, for comparison the result of non-flat α-level-optimization, are plotted as green pluses. The result of the α-level-free approach is shown as red dots in both Figure 4.1(b) and Figure 4.1(a). In the design spcace, a vertical branch is visible which is related to the left side slope of the result membership. The two diagonal branches in the design space are accountable for the right membership slope.



(a) Input space as explored by `SWS`.
Black: evaluated points
Red: points, which are part of the α-level-free solution's result.

(b) Fuzzy result quantity.
Black: analytical solution
Red: α-level-free
Blue: flat α-level-optimization
Green: non-flat α-level-optimization

Figure 4.1: Data points in the design space and objective space, calculated by the different approaches: α-level-free, flat α-level-optimization and non-flat α-level-optimization.

In Table 4.1 the calculated values for the α-levels 0.0 and 0.5 are listed and compared to the analytical solution. Due to the more samples, α-level-optimization yields the better results in the α-level 0.5.

`SWSForFuzzy` includes all hyperrectangle corners of support and core of the input quantity. Since maximum z is yielded by two data points $\phi(0.0, 1) = 1.25$ and $\phi(1, 1) = 1.25$ located in the corners, it is found by `SWSForFuzzy`. With about 400 contributing data points, the result is densely resolved. Thus, the proportion of evaluations contributing to the final result in comparison to the total evaluation count as defined in Equation (2.29), is roughly $p_{\text{eff}} \approx 18\,\%$. For flat α-level-optimization, the result consists of four data points, for which $p_{\text{eff}} \approx 0.9\,\%$. The result of non-flat α-level-optimization consists of roughly 1 250 data points, $p_{\text{eff}} \approx 30\,\%$.

Table 4.1: α-levels of the results.

Method	n_{tot}	z_{l}		z_{u}	
α-level		0.0	0.5	0.5	0.0
analytical		0.0	$0.353\,553$	$0.667\,893$	1.25
α-level-optimization	4 200	$0.000\,174$	$0.353\,573$	$0.657\,337$	$1.213\,280$
α-level-free	2 200	$0.000\,593$	$0.356\,696$	$0.661\,409$	1.25

4.2 Single Span Girder with Fuzzy Load Positions

A crane is used to drop off two crates of equipment on an already build beam. These crates cannot be stacked, so they are put down individually somewhere on the beam. The first is placed "somewhere in the left half", the other "to its right, but in the right half". A single span girder with two loads is given. Despite the deterministic weights, the exact position of both those loads is not known. The fuzzy maximum momentum load of the girder due to the loading is wanted. The system with applied loads and resulting partial moments is shown in Figure 4.3.

The girders length is set to

$$l = 5\,\mathrm{m}. \tag{4.3}$$

The deterministic loads are assumed as point loads with the respective values of

$$\begin{aligned} F_1 &= 5\,\mathrm{kN} \\ F_2 &= 3\,\mathrm{kN}. \end{aligned} \tag{4.4}$$

Their x-coordinate measured from the left bearing on the girder is given with the following fuzzy triangular quantities:

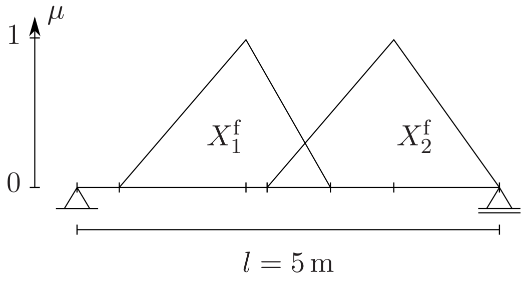
$$X_1^{\mathrm{f}} = \langle 0.1, 0.4, 0.6 \rangle \cdot l, \tag{4.5a}$$

$$X_2^{\mathrm{f}} = \langle 0.45, 0.75, 1 \rangle \cdot l \tag{4.5b}$$

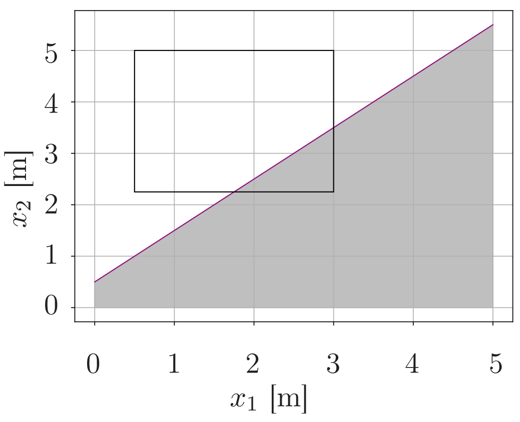
with the joint possibility according to the extension principle, see Section 2.2.2,

$$\mu(x_1, x_2) = \min(\mu(x_1), \mu(x_2)). \tag{4.5c}$$

In Figure 4.2(a) membership functions of X_1^{f} and X_2^{f} are depicted. Due to the crates'



(a) Membership functions of loads.



(b) Design space of the problem.

Figure 4.2: Design space of the girder problem.

spacial dimensions, they cannot be closer than $0.5\,\mathrm{m}$ to each other and $x_2 > x_1$, yielding the constraint

$$x_2 \geq x_1 + 0.5\,\mathrm{m}. \tag{4.6}$$

In Figure 4.2(b) the design space is plotted. The impermissible area is grayed out, while the rectangle shows the bounding box of the compound fuzzy input quantities support, that is object of study. The grayed out area is not part of the support. For a single moving load F on a single span girder the location of maximum bending momentum M is identical to the loads location x, see [2, p. 4.2]. The maximum bending moment is $M = \alpha\beta Fl$ with $\alpha = \frac{x}{l}$ and $\beta = \frac{l-x}{l}$. The maximum bending moments due to two loads is obtained by super-position of the bending moments caused by the individual loads and is located at one of the loads position. Moments are depicted in Figure 4.3.

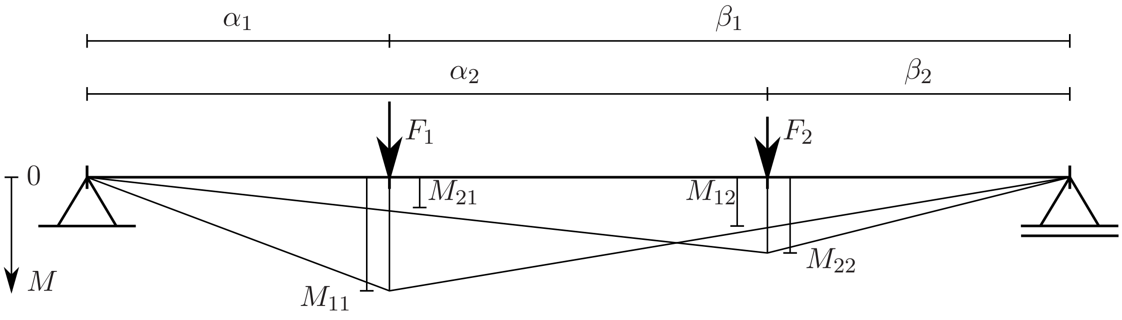


Figure 4.3: Moments on the girder.

The partial moments M_{ij}, where i denotes the reason and j the location of the partial moment, are

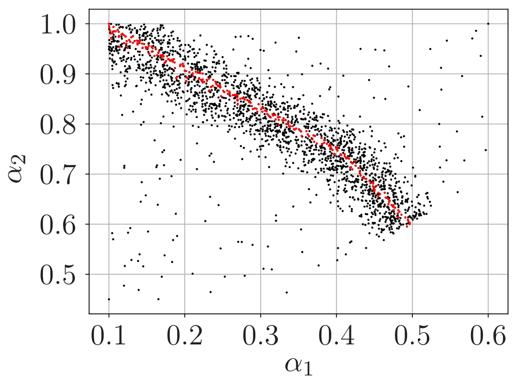
$$\begin{aligned} M_{11} &= \alpha_1\beta_1F_1l \\ M_{22} &= \alpha_2\beta_2F_2l \\ M_{12} &= \alpha_1\beta_2F_1l \\ M_{21} &= \alpha_1\beta_2F_2l \end{aligned} \tag{4.7}$$

and the fundamental solution becomes

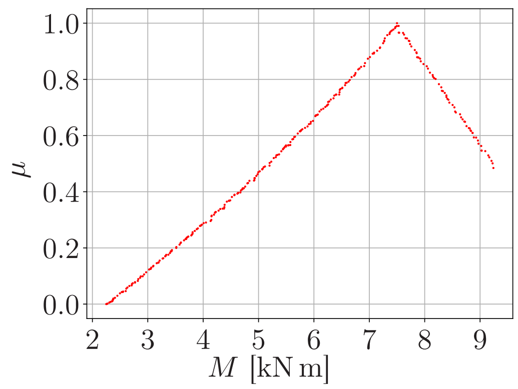
$$M = \max(M_{11} + M_{21}, M_{22} + M_{12}). \tag{4.8}$$

Since all moments are proportional to the girders length, α is used as the basis of computation. This avoids scaling back and forth between relative and absolute positions, thus saving a few operations. As uncertainty analysis *SWS* with Slice Sampling analogously to Section 4.1 is used.

Results of the analysis are shown in Figure 4.4. As visible in Figure 4.4(a), two linear parts are evident, with a knuckle at $(0.4, 0.75)$, which is the modal value of the input quantities. The left-hand side slope of the result quantity is caused by data point part of the branch left to the knuckle, as both loads move towards the middle of the girder. The set belonging to the right-hand side slope of the result quantity spans perpendicular from the modal value to the clearly evident constraint. When both loads are located as close as possible to each other and to the middle of the girder, the maximum momentum is yielded. The constraint's impact on the result is evident in the abrupt cut on the right hand side of the results membership. Interpretation of the result is, that under the postulated assumptions, maximum bending moments between $2.25\,\mathrm{kN\,m}$ and $9.25\,\mathrm{kN\,m}$ are to be anticipated. The upper bound is more plausible that the lower bound, with the most plausible value of $7.5\,\mathrm{kN\,m}$.



(a) Design space, points in red are contributing to the final result seen on the right.



(b) Result consisting of contributing samples.

Figure 4.4: Design space and result for the girder example.

4.3 Safety Assessment of a Wide Flange Steel Column under Consideration of Polymorphic Uncertain Parameters

In [68] different approaches are shown by five research groups, how an engineering problem, that exhibits polymorphic uncertainty can be tackled. First, the problem is described, afterwards the used assumptions, that are derived from the information given in [68] are presented. Finally, the result is discussed, with regards to an approach in [68].

4.3.1 Basic Problem

The engineering problem consists of a wide flange steel column. It is loaded with the permanent load P_p and the environmental load P_e, which models the snow load. The limit state function is given by

$$g(x) = 1 - \left(\frac{P}{f_\mathrm{y} A_\mathrm{s}} + \frac{P\delta_0}{f_\mathrm{y} W_\mathrm{s}} \cdot \frac{P_\mathrm{b}}{P_\mathrm{b} - P} \right) \tag{4.9}$$

with the EULER buckling load

$$P_\mathrm{b} = \frac{\pi^2 E I_\mathrm{s}}{L^2} \tag{4.10a}$$

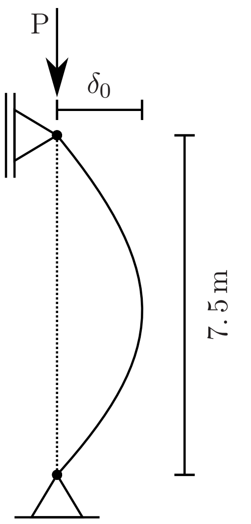
and cross section characteristic values area, section modulus and moment of inertia around the weak axis

$$A_\mathrm{s} = 2bt_\mathrm{b} + ht_\mathrm{h} \tag{4.10b}$$

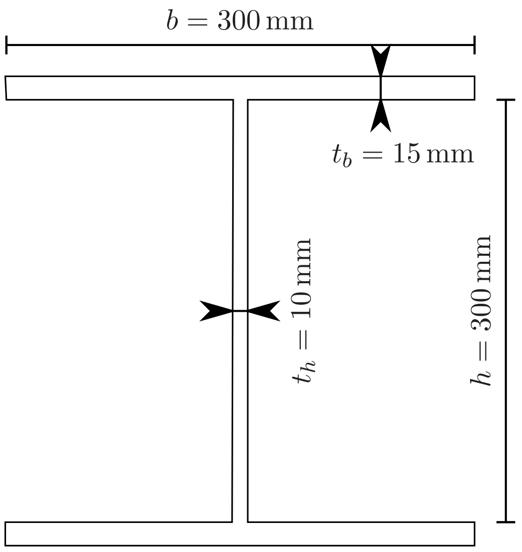
$$W_\mathrm{s} = \frac{ht_\mathrm{h}^2}{6b} + \frac{t_\mathrm{b}b^2}{3} \tag{4.10c}$$

$$I_\mathrm{s} = \frac{ht_\mathrm{h}^2}{12} + \frac{t_\mathrm{b}b^3}{6}. \tag{4.10d}$$

The material's YOUNGs modulus is denoted as E and the strength as f_y.



(a) Static loading of the column.



(b) Cross-section of the column.

Figure 4.5: Dimensions of the column.

4.3.2 Assumptions

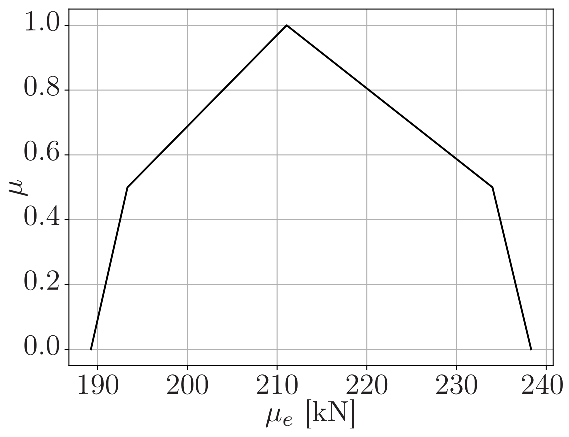
The assumptions made in [68, pp. 14 sqq.] are reflected in the following. P_p is modeled as a fuzzy triangular number $P_\mathrm{p}^\mathrm{f} = \langle 100\,\mathrm{kN}, 150\,\mathrm{kN}, 200\,\mathrm{kN}\rangle$. δ_0^f is modeled as a fuzzy triangular number $\delta_0^\mathrm{f} = \langle 0\,\mathrm{mm}, 0\,\mathrm{mm}, 60\,\mathrm{mm}\rangle$. The steels strength f_y^s is modeled as a lognormal distribution with a mean value of 400 MPa and a standard deviation of 32 MPa. YOUNGs modulus E^s is modeled as a lognormal distribution with a mean value of 210 000 MPa and a standard deviation of 8 400 MPa. $P_\mathrm{e}^\text{fp-r}$ is considered a *fp-r* quantity. Its GUMBEL-distribution has the fuzzy parameters $\mu_{P_\mathrm{e}}^\mathrm{f}$ and $\beta_{P_\mathrm{e}}^\mathrm{f}$. The fuzzy *cdf* of $P_\mathrm{e}^\text{fp-r}$ is plotted in Figure 4.7. Grid points for the piece-wise linear membership functions for both fuzzy parameters are listed in Table 4.2. Plots of the membership functions are shown in Figure 4.6(b), and Figure 4.6(a). The values are read off the given plot in [68, p. 15]. The uncertainty from inaccurate reading off is assumed to be small and is neglected. Geometric design parameters of the column are considered deterministic are listed in Table 4.3.

Table 4.2: Data points for $\mu_{P_\mathrm{e}}^\mathrm{f}$ and $\beta_{P_\mathrm{e}}^\mathrm{f}$.

Membership	0	0.5	1	0.5	0
μ_{P_e} in [kN]	189.245	193.320	211.066	234.010	238.320
β_{P_e} in [kN]	32.902	35.093	55.581	70.603	75.052

Table 4.3: Geometric design parameters of the column.

Dimension	b	t_b	h	t_h	L
Value	300 mm	15 mm	300 mm	10 mm	7.5 m



(a) Membership of $\mu_{P_e}^{f}$.

(b) Membership of $\beta_{P_e}^{f}$.

Figure 4.6: Membership functions for $\mu_{P_e}^{f}$ and $\beta_{P_e}^{f}$.

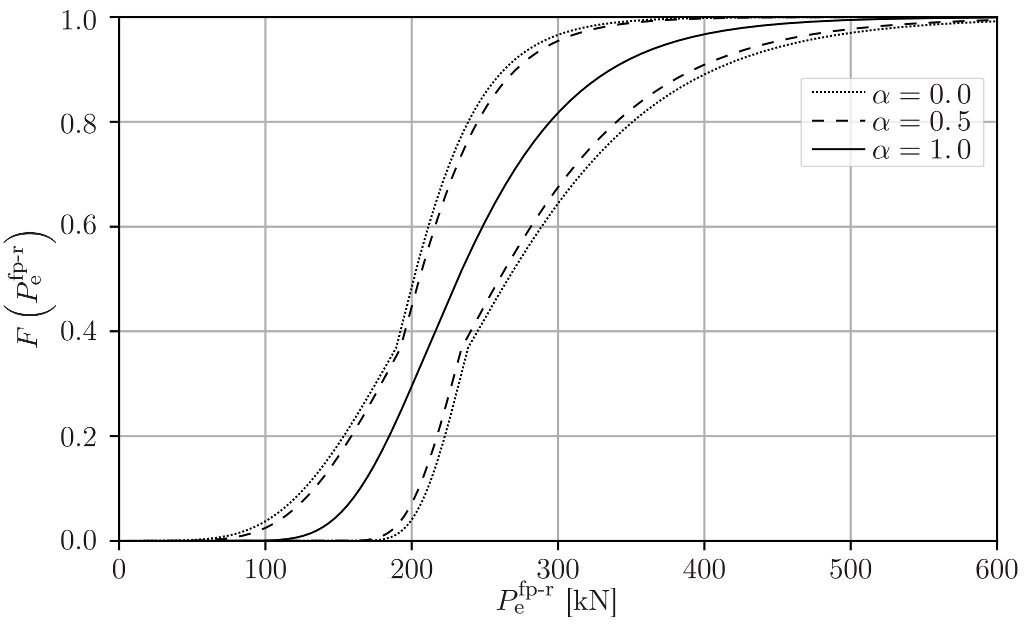


Figure 4.7: α-levels of the *cdf* for P_e.

4.3.3 Approach Shown by the Research Group

In [68, pp. 14 sqq.] a *fp-fr*-analysis is chosen, despite the fact, that no real *fp-fr*-quantity is present. The distribution uncertainty from scarce data, which cannot be reduced easily, is modeled by the fuzzy parameters $\mu_{P_e}^{f}$ and $\beta_{P_e}^{f}$. The uncertainty stemming from expert knowledge, which could be rather easily reduced by measuring is modeled with the fuzzy quantities δ_0^{f} and P_p^{f}. The influences of different uncertainty sources can be isolated from each others by splitting the input space, which will result in a split up result space too. Thereby it is possible to assess the uncertainty influence separately. To be able to do so, two distinct fuzzy analyses are employed. A structure of the analysis is shown in Figure 4.8

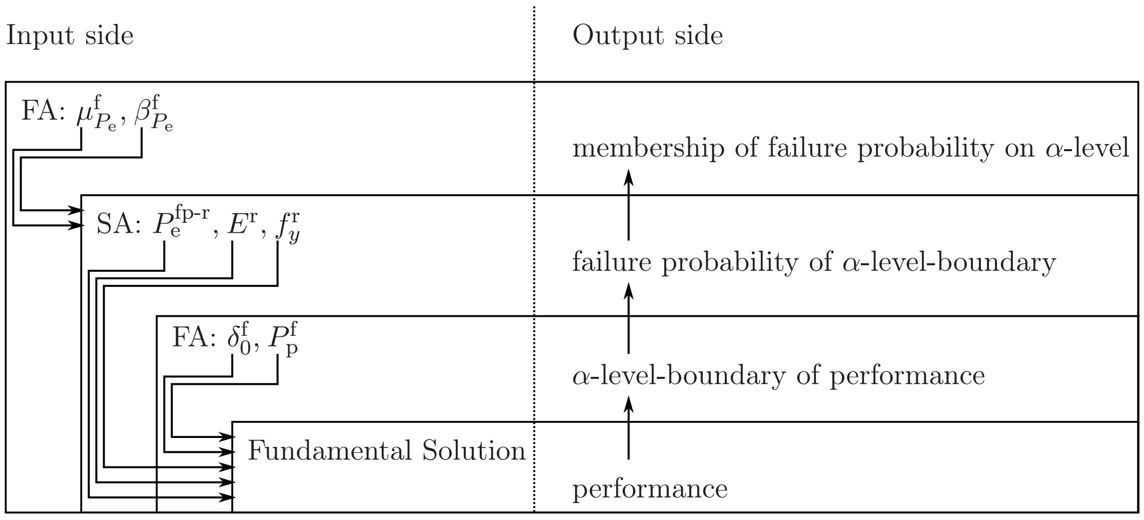

Input side
Output side
FA: $\mu^{\mathrm{f}}_{P_\mathrm{e}}$, $\beta^{\mathrm{f}}_{P_\mathrm{e}}$
membership of failure probability on α-level
SA: $P_\mathrm{e}^{\text{fp-r}}$, E^r, f_y^r
failure probability of α-level-boundary
FA: δ_0^f, P_p^f
α-level-boundary of performance
Fundamental Solution
performance

Figure 4.8: Structure of the analysis according to [68, pp. 14 sqq.].

4.3.4 Approach – Modifications and Algorithmic Parameters

To cut the computational effort, the *fp-fr* analysis is reduced to a *fp-r* one by omitting the inner fuzzy analysis. For this, P_p and δ_0^f are assumed to be deterministic. Therefore the respective values are chosen to be the worst possible ones according to the α-level, thus leading to the highest failure probabilities. It can be interpreted as to only evaluate the most critical point on the border of the support of the compound quantity is evaluated. This is done for two runs. In the first run, the upper borders of $C_0(P_\mathrm{p})$ and $C_0(\delta_0^\mathrm{f})$ are chosen. In the second run, the upper borders of $C_{0.5}(P_\mathrm{p})$ and $C_{0.5}(\delta_0^\mathrm{f})$ are chosen. The values are shown in Table 4.4. All other assumptions are adopted as declared in Section 4.3.2.

Table 4.4: Deterministic values chosen for the two runs.

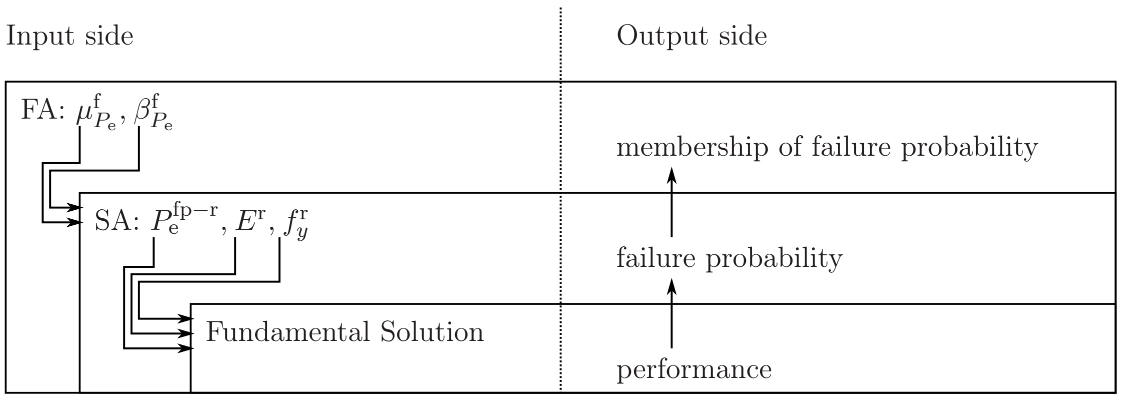
Run	α-level	δ_0^f	P_p
1	0.0	$0.06\,\mathrm{m}$	$200\,\mathrm{kN\,m}$
2	0.5	$0.03\,\mathrm{m}$	$175\,\mathrm{kN\,m}$

As the initialization of `Layer` objects in PUQPY, the analysis stages are described from innermost to the outermost. The inner analysis is the stochastic one. According to [69], crude *MC*-analysis requires an amount of $n = 10^{k+2}$ samples in total to approximate a failure probability of $P_\mathrm{f} = 10^{-k}$ with a statistical accuracy of 10 %. In [68], the permissible failure probability is set to 1.36×10^{-6}. This would require roughly 1×10^8 fundamental solution evaluations. Since this is far too expensive as the inner analysis loop, Subset Simulation is used for the intermediate stochastic analysis. For each subset level 5 000 samples are drawn, the conditional probability of the next level is set to 10 %. At most, 15 levels are carried out. The samples in the initial level are drawn by crude *MC*, in all further levels a *MCMC* sampling is used. The used algorithm is described in [3; 81].

The outer fuzzy analysis on $\mu^{\mathrm{f}}_{P_\mathrm{e}}$ and $\beta^{\mathrm{f}}_{P_\mathrm{e}}$ is done by *SWS* as α-level-free method. Settings are as shown in Table 4.5. Result is the membership function for a single α-level-boundary for the failure probability. The analysis' structure is shown in Figure 4.9, Layers are omitted in the graphic.

Table 4.5: Optimization parameters for the outer fuzzy analysis.

Parameter	n_{init}	$n_{\text{generations}}$	n_{iter}	δ
Value	200	100	20	300



Input side
Output side
FA: $\mu^{\text{f}}_{P_{\text{e}}}$, $\beta^{\text{f}}_{P_{\text{e}}}$
membership of failure probability
SA: $P^{\text{fp-r}}_{\text{e}}$, E^{r}, f^{r}_y
failure probability
Fundamental Solution
performance

Figure 4.9: Procedure structure for the *fp-r*-procedure, Layers are not shown.

4.3.5 Results

In the following, the results are shown and discussed. The resulting data points of the two runs are plotted in Figure 4.10, the respective α-level for $C_0(P_f)$, $C_{0.5}(P_f)$ and $C_1(P_f)$ are listed in Table 4.6. As expected, the failure probabilities are lower in the second run, due to the smaller initial deflection and permanent load.

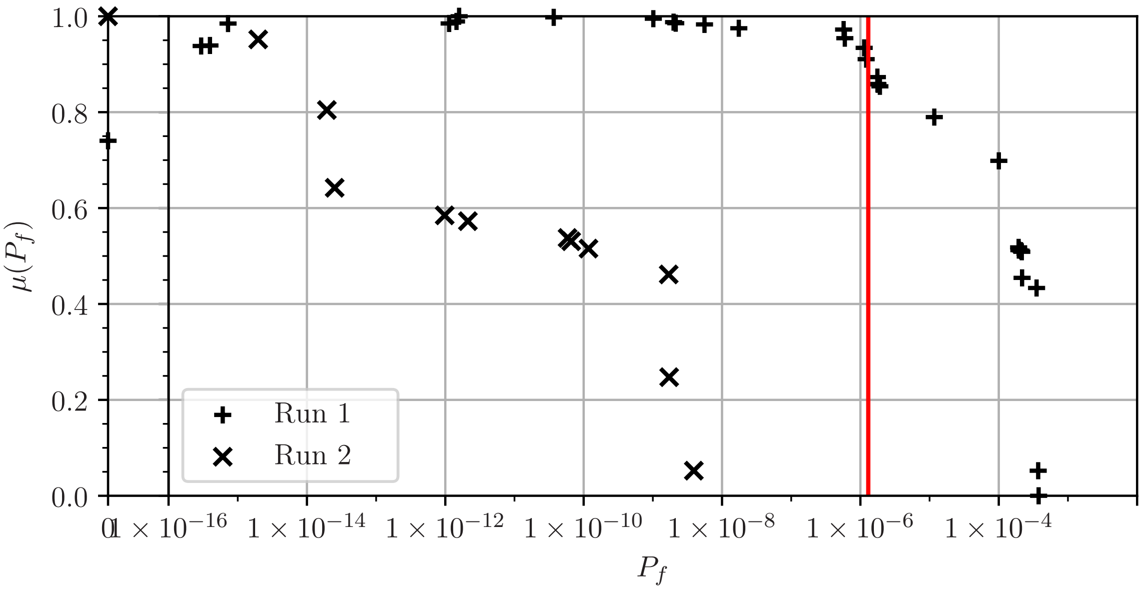


Figure 4.10: Result of the two runs. Red: permissible failure probability.

The structure can be considered safe for Run 2, as 3.914×10^{-9} is found as the highest failure probability on this level, which is lower than the set permissible failure probability 1.36×10^{-6}. Run 1, however, must be considered at least partially unsafe. The maximum failure probability found in this runs is 3.770×10^{-4}, which is significantly larger than the permissible failure probability. The core of the fuzzy result quantity consists of a single point at 1.584×10^{-12}. The permissible failure probabilities membership to the fuzzy result

Table 4.6: α-levels of the result for both runs.

	Run 1		Run 2	
$C_i(P_f)$	x_l	x_u	x_l	x_u
1.0	1.584×10^{-12}	1.584×10^{-12}	0.0	0.0
0.5	0.0	2.152×10^{-4}	0.0	1.174×10^{-10}
0.0	0.0	3.770×10^{-4}	0.0	3.914×10^{-9}

quantity is 0.9, using linear interpolation. Thus, it is highly plausible, that the failure probability exceeds the permissible one.

In comparison to the other numerical examples, the result quantity consists of considerably less data points and shows stepped result data points. This is a sign of bad optimization results, caused by *SWS* having difficulties to find the PARETO-set. It can be suspected, that result points, that are really close to each other, have nearly identical values for $\mu^{\mathrm{f}}_{P_{\mathrm{e}}}$ and $\beta^{\mathrm{f}}_{P_{\mathrm{e}}}$, since *SWS* does not have a restriction, how far points need to be from each other. To confirm the guess, that inconsistent return values may be the reason, subset simulation is carried out 100 times on two points. The core of the fuzzy quantities $\mu^{\mathrm{f}}_{P_{\mathrm{e}}}$ and $\beta^{\mathrm{f}}_{P_{\mathrm{e}}}$ are chosen as the first point. As the second point, the upper support boundaries of the same quantities are chosen. The mean $E(P_f)$, the standard deviation $\sqrt{\mathrm{Var}(P_f)}$ and coefficient of variation $\nu(P_f) = \frac{\sqrt{\mathrm{Var}(P_f)}}{\mathrm{E}(P_f)}$ of the results are shown in Table 4.7. With decreasing target

Table 4.7: α-levels of the result for both runs.

δ_0 [m]	P_{p} [kN]	$\mu^{\mathrm{f}}_{P_{\mathrm{e}}}$ [kN]	$\beta^{\mathrm{f}}_{P_{\mathrm{e}}}$ [kN]	$E(P_f)$	$\sqrt{\mathrm{Var}(P_f)}$	$\nu(P_f)$
0.06	200.0	211.066	55.581	1.036×10^{-8}	6.188×10^{-8}	5.974
0.06	200.0	238.320	75.052	3.834×10^{-4}	3.895×10^{-5}	0.102

failure probability the variation coefficient of the reported results increases. These findings could explain the wide spread of high membership from 1×10^{-15} to 1×10^{-6}. The 5 000 generated samples per level in a trivariate input space may be too few. A parameter study to postulate recommendations for parameters depending on the target failure probability and input dimensionality is proposed. However, this is not in the scope of this thesis.

Notable differences to the results presented in [68, p. 16] are present. These are:

- Run 1
  - With 3.770×10^{-4}, the found maximum failure probability is twice as high as the respective value in [68].
  - The modal value is 5 magnitudes lower than in [68].
  - Lowest failure probability is 0.0 with a membership of 0.74, in [68] the support is bounded at roughly 1×10^{-11} on the lower side.
- Run 2
  - With 3.914×10^{-9} the found maximum failure probability is three magnitudes lower as the respective value in [68].
  - As modal value of 0.0 is found. Thus no left slope is found as in [68].

5 Discussion

5.1 Results

In this thesis, uncertainty modeling and analysis methods are discussed. Monomorphic uncertainty is differentiated in aleatoric and epistemic uncertainty. Modeling of polymorphic uncertainty by combination of monomorphic uncertainty is explained. For epistemic uncertainty, fuzzy quantities, interactions of fuzzy quantities, and fuzzy analysis are shown. Fuzzy-based fuzziness (*ff*) is proposed as an advanced epistemic uncertainty model. α-level-based and α-level-free methods are shown in contrast and the hybrid non-flat α-level-optimization is presented. The order of α-level for α-level-based fuzzy analysis is discussed. Implementation approaches to updating algorithms for PARETO-fronts are benchmarked.

In cooperation with [81], PUQPY, a framework for generic modeling and analysis of aleatoric, epistemic and polymorphic uncertainty is conceptualized and implemented. Flexibility and extensibility are achieved by the modular architecture and standardized interfaces of analyses and quantities. Monomorphic uncertain quantities can be constructed from predefined classes or modified in a sub-class. Analysis methods for aleatoric and epistemic uncertainty are provided. Polymorphic uncertain quantities are modeled by staging several analyses, using `Layer` objects to connect and manage the analyses. With the use of `Layer` objects, analysis are arbitrarily nestable. *SWS* with slice sampling is implemented as optimization algorithm and fuzzy analysis algorithm. In PUQPY, parallelization by using DASK and pre-compilation with CYTHON are approached as performance enhancements.

In the numerical examples, the usability is demonstrated for monomorphic and polymorphic analyses. Flat α-level-optimization is compared to non-flat α-level-optimization and the α-level-free method in a benchmark example.

5.2 Limitation and Perspective

The implementation of PUQPY has considerable improvement potential, both in performance and features. The integration with DASK, see [15; 72], is to be extended for a broad support of distributed computing. Measures need to be taken, to limit memory usage, especially in staged analyses in combination with brute *MC*-analysis. Automatic dividing of data into manageable chunks is yet to be implemented. The distributed character is to be adopted from the parallel invocation of the subordinated analysis to parts of the body on analyses. Scalability is to be improved by better work load distribution. For small tasks, the scheduling and communication overhead becomes the bottleneck, see [15], but for long running task, the cluster may wait on a single, slow worker to finish. Further speedup may be gained, by transitioning more parts from plain PYTHON to CYTHON. While pre-compilation alone will not yield noticeable improvements, breaking down algorithms, may speed up the computation by orders of magnitude. The combination of both parallelization and pre-compilation is not accomplished yet.

At the moment, *SWS* is the only implemented optimization algorithm. It is planned to extend the module of optimization algorithms. Therefore, a wrapper to integrate PYMOO is planned, since it offers a wide range of proven powerful optimization routines, such as NSGA2 based algorithms, see [10; 17; 18; 79]. When analyzing a computationally cheap fundamental solution, by far the most run-time is spent in the weight function of SWS. The performance hit, most observable with many contributing data points, could be remedied by employing parallelization and pre-compilation. As implemented in the context of this thesis, SWS does not find points located directly on the border of the design space. This is a significant drawback for the optimization accuracy of fundamental solutions, whose optimium is located on the design space border. It is proposed to investigate the possible advantage of multiple MARKOV-chains for sample generation. The use of low discrepancy sampling methods, such as the SOBOL sequence, see [85], or latin hypercube sampling, see [52] in the initial generation of SWS is yet to be implemented. In-depth parameter studies on SWS are not done in this thesis. It is proposed to investigate the parameters' influence to improve default parameters and to develop recommendations for the user documentation The development of an adaptive termination criterion for SWS is suggested.

As implemented in the scope of this thesis, slice sampling supports sampling from multivariate quantities, but expansion of the hyperrectangle is not implemented. Approaches to multivariate Slice Sampling steps are proposed in [48; 90; 91; 92]. Additional improvement is possible, since procedures for random walk suppression and adaptive shrinkage using intelligent sensitivity analysis are proposed in [64; 65].

The algorithms available in PUQPY for stochastic analysis have shown high inaccuracies for small failure probabilities. Further development work is necessary, to rule out implementation errors and improve the algorithms themselves. Parameter studies are proposed to develop recommendation for default parameters in accordance to the dimensionality of stochastic input quantities and target probability.

The order of α-levels are discussed, but no in-depth study is made in this thesis. It is proposed to investigate the benefits of either calculation order by extensive benchmarking, in which cases the one or the other proves more effective.

Field quantities are not implemented in the scope of this work. Implementation of an unified class in PUQPY for interval, fuzzy, stochastic and polymorphic uncertain field quantities is proposed as another research task.

The concept of fuzzy valued fuzzy quantities is not yet dealt with in depth-in research. A step towards this goal is taken in [68]. Yet, the taken approach is still dependent on information reduction measures in-between the analyses. Therefore, further investigation of the isolation of input and output spaces is suggested. Direct analysis approaches to ff, that are independent from information reduction measures are still to be developed.

Acronyms and Glossary

Acronyms

cdf	cumulated density function	30
icdf	inverse cumulated density funtion	27
iid	independent, identically distributed	4
FEM	Finite-Element-Method	39
ff	fuzzy-based fuzziness	42
fr	fuzzy randomness	5
fp-r	fuzzy probability based randomness	5
fp-fr	fuzzy probability based fuzzy randomness	5
MC	Monte-Carlo	30
MCMC	Markov-Chain Monte-Carlo	27
MCS	Monte-Carlo-Sampling	29
pdf	probability density function	27
SWS	Sequential Weighted Sampling	IX

Glossary

$\uparrow$	maximize the following expression
$\downarrow$	minimize the following expression
$\square^{\mathrm{f}}$	fuzzy quantity
$\square^{\mathrm{s}}$	stochastic quantity
$\boldsymbol{v} = (a, b)$	vector
$\mathbf{0}$	zero vector
$\mathbf{1}$	ones vector
$\mathcal{U}(a, b)$	uniformly distributed quantity between a and b
$\mathcal{U}(\boldsymbol{a}, \boldsymbol{b})$	uniformly distributed quantity element wise between $\boldsymbol{a}$ and $\boldsymbol{b}$
$\mathbb{N}$	set of natural numbers
$\mathbb{N}^{+}$	set of positive natural numbers ($\mathbb{N} \setminus \{0\}$)
$\mathbb{R}$	set of real numbers

References

[1] Akther, T.; Ahmad, S. U.: A computantional method for fuzzy arithmetic operations. *Daffodil international University Journal of science and technology* 4 (1 2009). doi: 10.3329/diujst.v4i1.4350.

[2] Albert, A., (ed.): *Bautabellen für Ingenieure. mit Berechnungshinweisen und Beispielen.* 22nd ed. Bundesanzeiger Verlag, Köln, 2016.

[3] Au, S.-K.; Beck, J. L.: Estimation of small failure probabilities in high dimensions by subset simulation. *Probabilistic Engineering Mechanics* 16 (2001) 263–277. doi: 10.1016/S0266-8920(01)00019-4.

[4] Audet, C.; Dang, C.-K.; Orban, D.: Efficient use of parallelism in algorithmic parameter optimization applications. *Optimization Letters* 7 (2011) 421–433. doi: 10.1007/s11590-011-0428-6.

[5] Audet, C.; Dang, K.-C.; Orban, D.: Optimization of algorithms with OPAL. *Mathematical Programming Computation* 6 (2014) 233–254. doi: 10.1007/s12532-014-0067-x.

[6] Baldick, R.: *Applied Optimization. Formulation and Algorithms for Engineering Systems.* Cambridge University Press, Cambridge, 2006. doi: 10.1017/cbo9780511610868.

[7] Baudin, M.; Dutfoy, A.; Iooss, B.; Popelin, A.-L.: Open TURNS: An industrial software for uncertainty quantification in simulation (2015). doi: 10.1007/978-3-319-11259-6_64-1.

[8] Bayes, T.; Price, R.: LII. An essay towards solving a problem in the doctrine of chances. By the late Rev. Mr. Bayes, F. R. S. communicated by Mr. Price, in a letter to John Canton, A. M. F. R. S. *Philosophical Transactions of the Royal Society of London* 53 (1763) 370–418. doi: 10.1098/rstl.1763.0053.

[9] Behnel, S.; Bradshaw, R.; Citro, C.; Dalcin, L.; Seljebotn, D. S.; Smith, K.: Cython: The Best of Both Worlds. *Computing in Science Engineering* 13 (2011) 31–39. doi: 10.1109/MCSE.2010.118.

[10] Blank, J.; Deb, K.: pymoo: Multi-objective Optimization in Python. *IEEE Access* (2020). doi: 10.1109/access.2020.2990567.

[11] Bobach, T.; Umlauf, G.: *Natural Neighbor Interpolation and Order of Continuity.* Research rep. Computer Science Department, University of Kaiserslautern, 2006.

[12] Böttcher, M.; Leichsenring, F.; Fuchs, A.; Graf, W.; Kaliske, M.: Efficient Utilization of Surrogate Models for Uncertainty Quantification. *Proceedings in Applied Mathematics and Mechanics* (2020). doi: 10.1002/pamm.202000210.

[13] Brooks, S.; Gelman, A.; Jones, G.; Meng, X.-L., (eds.): *Handbook of Markov Chain Monte Carlo.* Chapman and Hall/CRC, 2011. doi: 10.1201/b10905.

[14] Burke, E. K.; Kendall, G., (eds.): *Search Methodologies.* 2nd ed. Springer, New York, 2014. doi: 10.1007/978-1-4614-6940-7.

[15] Dask Development Team: Dask: Library for dynamic task scheduling. 2016. url: https://dask.org.

[16] Deb, K.: In: *Multi-objective Optimization: Search Methodologies*. Ed. by Burke, E. K.; Kendall, G. 2nd ed. Springer, New York, 2014. 15, 403–449. doi: 10.1007/978-1-4614-6940-7_15.

[17] Deb, K.; Tiwari, S.: Omni-optimizer: A generic evolutionary algorithm for single and multi-objective optimization. *European Journal of Operational Research* 185 (2008) 1062–1087. doi: 10.1016/j.ejor.2006.06.042.

[18] Deb, K.; Tiwari, S.: Omni-optimizer: A Procedure for Single and Multi-objective Optimization. In: *Evolutionary Multi-Criterion Optimization* 3410, Springer, Berlin, Heidelberg, (2005). doi: 10.1007/978-3-540-31880-4_4.

[19] Drieschner, M.; Petryna, Y.; Gruhlke, R.; Eigel, M.; Hömberg, D.: Comparison of various uncertainty models with experimental investigations regarding the failure of plates with holes. *Reliability Engineering & System Safety* 203 (2020). doi: 10.1016/j.ress.2020.107106.

[20] Du, L.; Choi, K.; Youn, B. D.: A New Fuzzy Analysis Method for Possibility-Based Design Optimization. *Collection of Technical Papers – 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference* 5 (2004). doi: 10.2514/6.2004-4585.

[21] Dubois, D.; Prade, H.: Operations on fuzzy numbers. *International Journal of Systems Science* 9 (1978) 613–626. doi: 10.1080/00207727808941724.

[22] Dubois, D.; Prade, H.: The mean value of a fuzzy number. *Fuzzy Sets and Systems* 24 (1987) 279–300. doi: 10.1016/0165-0114(87)90028-5.

[23] Fahrmeir, L.; Heumann, C.; Künstler, R.; Pigeot, I.; Tutz, G.: *Statistik. Der Weg zur Datenanalyse*. 2016. doi: 10.1007/978-3-662-50372-0.

[24] Fan, Y.; Sisson, S. A.: In: *Reversible Jump MCMC: Handbook of Markov Chain Monte Carlo*. Ed. by Brooks, S.; Gelman, A.; Jones, G.; Meng, X.-L. Chapman and Hall/CRC, 2011. 3, 67–91. doi: 10.1201/b10905-5.

[25] Fernandes, C.; Pontes, A.; Viana, J.; Gaspar-Cunha, A.: In: *Using Multi-Objective Evolutionary Algorithms in the Optimization of Polymer Injection Molding: Applications of Soft Computing*. Vol. 58, Springer, Berlin, Heidelberg, 2009, 357–365. doi: 10.1007/978-3-540-89619-7_35.

[26] Filev, D.; Yager, R.: A Generalized Defuzzification Method via Bad Distributions. *International Journal of Intelligent Systems* 6 (1991) 687–697. doi: 10.1002/int.4550060702.

[27] Gagolewski, M.: *A Guide to the FuzzyNumbers Package for R (FuzzyNumbers version 0.4-6)*. Comp. software. Systems Research Institute, Polish Academy of Sciences, Warsaw, Poland; Institute of Geoinformatics, VŠB – Technical University of Ostrava, 2019.

[28] Gelman, A.; Shirley, K.: In: *Inference from Simulations and Monitoring Convergence: Handbook of Markov Chain Monte Carlo*. Ed. by Brooks, S.; Gelman, A.; Jones, G.; Meng, X.-L. Chapman and Hall/CRC, 2011. 6, 163–174. doi: 10.1201/b10905-8.

[29] Geyer, C.: In: *Introduction to Markov Chain Monte Carlo: Handbook of Markov Chain Monte Carlo*. Ed. by Brooks, S.; Gelman, A.; Jones, G.; Meng, X.-L. Chapman and Hall/CRC, 2011. 1, 3–48. doi: 10.1201/b10905-3.

[30] Götz, M.: *Numerische Entwurfsmethoden unter Berücksichtigung polymorpher Unschärfe.* Dissertationsschrift, Institut für Statik und Dynamik der Tragwerke, Fakultät Bauingenieurwesen der Technischen Universität Dresden, 2017.

[31] Götz, M.; Leichsenring, F.; Graf, W.; Kaliske, M.: Four Types of Dependencies for the Fuzzy Analysis. *6th European Conference on Computational Mechanics (ECCM 6), 7th European Conference on Computational Fluid Dynamics (ECFD 7)* (2018).

[32] Graf, W.; Götz, M.; Kaliske, M.: In: *Structural design with polymorphic uncertainty model*: *Proceedings of the 6th International Workshop on Reliable Engineering Computing. Reliability and Computations of Infrastructures.* Ed. by Modares, M. Illinois Institute of Technology, Chicago, 2014, 64–76.

[33] Graf, W.; Götz, M.; Kaliske, M.: Structural design with polymorphic uncertainty model. *International Journal of Reliability and Safety (IJRS)* 9 (2015) 112–131. doi: 10.1504.ijrs.2015.072715.

[34] Graf, W.; Götz, M.; Kaliske, M.: Analysis of dynamical processes under consideration of polymorphic uncertainty. *Structural Safety* 52, Part B (2015) 194–201. doi: 10.1016/j.strusafe.2014.09.003.

[35] Gruhlke, R.; Drieschner, M.; Eigel, M.; Hömberg, D.; Petryna, Y.: Artificial Neural Network forecasting for monomorphic and polymorphic uncertainty models and comparison with experimental investigations. *PAMM* 19 (2019) e201900359. doi: https://doi.org/10.1002/pamm.201900359.

[36] Hanss, M.; Mäck, M.: Certainly uncertain – the charm of fuzzy predictions. *Procedia Engineering* (2017). doi: 10.1016/j.proeng.2017.09.149.

[37] Hanss, M.: *Applied Fuzzy Arithmetic. An Introduction with Engineering Applications.* Springer, Stuttgart, 2010. doi: 10.1007/b138914.

[38] Harris, C. R.; Millman, K. J.; Walt, S. J. van der, et al.: Array programming with NumPy. *Nature* 585 (2020) 357–362. doi: 10.1038/s41586-020-2649-2.

[39] Hose, D.; Mäck, M.; Hanss, M.: On probability-possibility consistency in high-dimensional propagation problems. In: *3rd ECCOMAS Thematic Conference on Uncertainty Quantification in Computational Sciences and Engineering*, Crete, Greece, 2019. doi: 10.7712/120219.6330.18439.

[40] Jarre, F.; Stoer, J.: *Optimierung.* Einführung in mathematische Theorie und Methoden. 2nd ed. Berlin, 2019. doi: 10.1007/978-3-662-58855-0.

[41] Kaufmann, M. A.: *Inductive Fuzzy Classification in Marketing Analytics.* PhD thesis, Faculty of Science of the University of Fribourg, Switzerland, 2012. doi: 10.1007/978-3-319-05861-0.

[42] Kotulak, K.; Froń, A.; Krankowski, A.; Pulido, G. O.; Henrandez-Pajares, M.: Sibsonian and non-Sibsonian natural neighbour interpolation of the total electron content value. *Acta Geophysica* (2017) 1895–7455. doi: 10.1007/s11600-017-0003-3.

[43] Kroese, D.; Taimre, T.; Botev, Z.: *Handbook of Monte Carlo Methods.* Wiley, 2011. doi: 10.1002/9781118014967.

[44] Laguna, M.; Molina, J.; Pérez, F.; Caballero, R.; Hernández-Díaz, A.: The Challenge of Optimizing Expensive Black Boxes: A Scatter Search/Rough Set Theory Approach. *Journal of the Operational Research Society* 61 (2010) 53–67. doi: 10.1057/jors.2009.124.

[45] Ledoux, H.; Gold, C.: In: *An Efficient Natural Neighbour Interpolation Algorithm for Geoscientific Modelling*: *Developments in Spatial Data Handling.* Springer, Berlin, Heidelberg, 2005. doi: 10.1007/3-540-26772-7_8.

[46] Leekwijck, W. V.; Kerre, E. E.: Defuzzification: criteria and classification. *Fuzzy Sets and Systems* 108 (1999) 159–178. doi: 10.1016/S0165-0114(97)00337-0.

[47] Leichsenring, F.; Jenkel, C.; Graf, W.; Kaliske, M.: Numerical simulation of wooden structures with polymorphic uncertainty in material properties. *International Journal of Reliability and Safety* 12 (2018) 24–45. doi: 10.1504/ijrs.2018.092499.

[48] Liechty, M. W.; Lu, J.: Multivariate Normal Slice Sampling. *Journal of Computational and Graphical Statistics* 19 (2010) 281–294. doi: 10.1198/jcgs.2009.07138.

[49] Mäck, M.; Hanss, M.: An advanced sampling technique for possibilistic uncertainty propagation. *Mechanical Systems and Signal Processing* 147 (2021). doi: 10.1016/j.ymssp.2020.107064.

[50] Mäck, M.; Hose, D.; Hanss, M.: On Using Fuzzy Arithmetic in Optimization Problems with Uncertain Constraints. *Special Issue:88th Annual Meeting of the International Association of Applied Mathematics and Mechanics (GAMM)* 17 (1 2017). doi: 10.1002/pamm.201710017.

[51] Marano, G. C.; Morrone, E.; Quaranta, G.; Trentadue, F.: Fuzzy Structural Analysis of a Tuned Mass Damper Subject to Random Vibration. *Advances in Acoustics and Vibration* 2008 (2008). doi: 10.1155/2008/207254.

[52] McKay, M. D.; Beckman, R. J.; Conover, W. J.: A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics* 21 (1979) 239–245.

[53] Mogharreban, N.; Dilalla, L.: Comparison of Defuzzification Techniques for Analysis of Non-interval Data. In: 2006. doi: 10.1109/NAFIPS.2006.365418.

[54] Möller, B.; Beer, M.: *Safety Assessment using Fuzzy Theory.* Research rep. Dresden University of Technology, Department of Civil Engineering, 1998.

[55] Möller, B.; Beer, M.; Graf, W.; Sickert, J.-U.: Fuzzy finite element Method and its application. *Trends in computational structural mechanics* (2001).

[56] Möller, B.; Beer, M.; Liebscher, M.: Fuzzy analysis as alternative to stochastic methods – theoretical aspects. *4th German LS-DYNA Forum, Bamberg* (2005).

[57] Möller, B.; Graf, W.; Beer, M.: Fuzzy structural analysis using α-level optimization. *Computational Mechanics* 26 (2000) 547–565. doi: 10.1007/s004660000204.

[58] Möller, B.; Graf, W.; Beer, M.; Sickert, J.-U.: Fuzzy probabilistic method and its application for the safety assessment of structures. *European Conference on Computational Mechanics, Cracow* (2001).

[59] Möller, B.; Graf, W.; Beer, M.; Sickert, J.-U.: Fuzzy Randomness - Towards a new Modeling of Uncertainty. *Fifth World Congress on Computational Mechanics* (2002).

[60] Möller, B.; Hoffmann, A.; Liebscher, M.: Modeling of blasting processes in view of fuzzy randomness. *9th ASCE Specialty Conference on Probabilistic Mechanics and Structural Reliability (PMC2004)* (2004).

[61] Murray, I.; Adams, R. P.; MacKay, D. J.: Elliptical slice sampling. *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)* (2010).

[62] Navara, M.; Žabokrtský, Z.: How to make constrained fuzzy arithmetic efficient. *Soft Computing* 5 (2001) 412–417. doi: 10.1007/s005000100089.

[63] Neal, R.: In: *MCMC Using Hamiltonian Dynamics*: *Handbook of Markov Chain Monte Carlo*. Ed. by Brooks, S.; Gelman, A.; Jones, G.; Meng, X.-L. Chapman and Hall/CRC, 2011. 5, 113–162. doi: 10.1201/b10905-7.

[64] Neal, R. M.: *Slice sampling*. Tech. rep. Department of Statistics and Department oc Vomputer Science, University of Toronto, Ontario, Canada, 2000.

[65] Neal, R. M.: Slice sampling. *The Annals of Statistics* 31 (2003) 705–767. doi: 10.1214/aos/1056562461.

[66] Nehi, H.; Malek, H.: *Intuitionistic Fuzzy Numbers and It's Applications in Fuzzy Optimization Problem*. Research rep. Department of Mathematics, Sistan and Baluchestan University, Zahedan, IRAN; College of Basic Sciences, Shiraz University of Technology, Shiraz, IRAN, 2005.

[67] Papadopoulos, V.; Giovanis, D. G.; Lagaros, N.; Papadrakakis, M.: Accelerated subset simulation with neural networks for reliability analysis. *Computer Methods in Applied Mechanics and Engineering* 223-224 (2012) 70–80. doi: 10.1016/j.cma.2012.02.013.

[68] Papaioannou, I.; Daub, M.; Drieschner, M., et al.: Assessment and design of an engineering structure with polymorphic uncertainty quantification. *GAMM-Mitteilungen* 42 (2019). doi: 10.1002/gamm.201900009.

[69] Papaioannou, I.; Papadimitriou, C.; Straub, D.: Sequential importance sampling for structural reliability analysis. *Structural Safety* 62 (2016) 66–75. doi: 10.1016/j.strusafe.2016.06.002.

[70] Pardalos, P. M.; Žilinskas, A.; Žilinskas, J.: *Non-Convex Multi-Objective Optimization*. Ed. by Pardalos, P. M. Springer, Gainesville (Florida), Vilnius, 2017. doi: 10.1007/978-3-319-61007-8.

[71] Patelli, E.: In: *COSSAN: A Multidisciplinary Software Suite for Uncertainty Quantification and Risk Management*: *Handbook of Uncertainty Quantification*. Ed. by Ghanem, R.; Higdon, D.; Owhadi, H. Springer International Publishing, 2016, 1–69. doi: 10.1007/978-3-319-11259-6_59-1.

[72] Rocklin, M.: Dask: Parallel Computation with Blocked algorithms and Task Scheduling. In: *Proceedings of the 14th Python in Science Conference* 2015.

[73] Rommelfanger, H.: *Fuzzy Decision Support-Systeme. Entscheiden bei Unschärfe*. 2nd ed. Springer-Verlag Berlin Heidelberg, 1994. doi: 10.1007/978-3-642-57929-5.

[74] Saletic, D. Z.; Velasevic, D. M.; Mastorakis, N. E.: Analysis of Basic Defuzzification Techniques. In: *6th WSEAS International Multiconference on Circuits, Systems, Communications and Computers (CSCC 2002)* 2002.

[75] Sastry, K.; Goldberg, D.; Kendall, G.: In: *Genetic Algorithms*: *Search Methodologies*. Ed. by Burke, E. K.; Kendall, G. 2nd ed. Springer, New York, 2014. 4, 93–117. doi: 10.1007/978-1-4614-6940-7_4.

[76] Saxena, B.; Pal, S.: Some new concepts in fuzzy arithmetic. *Journal of Discrete Mathematical Sciences and Cryptography* 13 (2010) 257–270. doi: 10.1080/09720529.2010.10698291.

[77] Schietzold, F. N.: *Numerische Analyse von Holzstrukturen unter Beachtung von Unschärfe*. Diploma thesis, Institut für Statik und Dynamik der Tragwerke, Fakultät Bauingenieurwesen der Technischen Universität Dresden, 2017.

[78] Schietzold, F. N.; Graf, W.; Kaliske, M.: Polymorphic uncertainty modeling for optimization of timber structures. *Computing with Confidence* (2018).

[79] Seada, H.; Deb, K.: *U-NSGA-III: A Unified Evolutionary Algorithm for Single, Multiple, and Many-Objective Optimization.* Research rep. Computational Optimization, Innovation Laboratory (COIN), Department of Computer Science, and Engineering, Michigan State University, 2014.

[80] Segura, C.; Coello, C. A. C.; Miranda, G.; León, C.: Using multi-objective evolutionary algorithms for single-objective optimization. *4OR* 11 (2013) 201–228. doi: 10.1007/s10288-013-0248-x.

[81] Seidowski, M.: *Stochastische Analyse als Teil generischer polymorpher Unschärfe-Analyse.* Diploma thesis, Institut für Statik und Dynamik der Tragwerke, Fakultät Bauingenieurwesen der Technischen Universität Dresden, 2022.

[82] Shenify, M.; Mazarbhuiya, F. A.: The Expected Value of a Fuzzy Number. *International Journal of Intelligence Science* 5 (2015) 1–5. doi: 10.4236/ijis.2015.51001.

[83] Shields, M. D.; Giovanis, D.: UQPy – Uncertainty Quantification with Python. Version 2.0.0. 2018.

[84] Sickert, J.-U.; Beer, M.; Graf, W.; Möller, B.: In: *Fuzzy probabilistic structural analysis considering fuzzy random functions.* der Kiureghian, A.: *Applications of Statistics and Probability in Civil Engineering.* 2003.

[85] Sobol, I. M.: On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics* 7 (1967) 86–112. doi: 10.1016/0041-5553(67)90144-9.

[86] Spaethe, G.: *Die Sicherheit tragender Baukonstruktionen.* Springer Vienna, 1992.

[87] Steinigen, F.; Sickert, J.-U.; Graf, W.; Kaliske, M.: In: *Fuzzy and Fuzzy Stochastic Methods for the Numerical Analysis of Reinforced Concrete Structures Under Dynamical Loading*: *Computational Methods in Stochastic Dynamics: Volume 2.* Ed. by Papadrakakis, M.; Stefanou, G.; Papadopoulos, V. Springer Netherlands, Dordrecht, 2013, 113–130. doi: 10.1007/978-94-007-5134-7_7.

[88] Štěpnička, M.; Baets, B. D.; Nosková, L.: Arithmetic Fuzzy Models. *IEEE TRANSACTIONS ON FUZZY SYSTEMS* 18 (2010). doi: 10.1109/tfuzz.2010.2062522.

[89] Thiele, M.; Liebscher, M.; Graf, W.: Fuzzy analysis as alternative to stochastic methods – a comparison by means of a crash analysis. *4th German LS-DYNA Forum, Bamberg* (2005).

[90] Thompson, M.; Neal, R. M.: *Covariance-Adaptive Slice Sampling.* Tech. rep. Department of Statistics, University of Toronto, 2010.

[91] Thompson, M.: *Slice Sampling with Multivariate Steps.* PhD thesis, Graduate Department of Statistics, University of Toronto, 2011.

[92] Tibbits, M. M.; Haran, M.; Liechty, J. C.: Parallel multivariate slice sampling. *Statistics and Computing* 21 (3 2011) 415–430. doi: 10.1007/s11222-010-9178-z.

[93] Vahidi, J.; S.Rezvani: Arithmetic Operations on Trapezoidal Fuzzy Numbers. *Journal Nonlinear Analysis and Application* 2013 (2013) 1–8. doi: 10.5899/2013/jnaa-00111.

[94] van Heesch, D.: doxygen. Manual for version 1.9.1. 2021. url: `www.doxygen.nl/download.html` (visited on 02/08/2021).

[95] van Rossum, G.; Drake, Jr., F., (eds.): Python Reference Manual. Release 2.1.1. Python Software Foundation, 2001.

[96] van Rossum, G.; Drake, Jr., F., (eds.): Python Reference Manual. Release 3.2.3. Python Software Foundation, 2012.

[97] van Rossum, G.: *Python tutorial.* Tech. rep. Centrum voor Wiskunde en Informatica, Computer Science/Department of Algorithmics and Architecture, 1995.

[98] Viertl, R.: *Statistical Methods for Fuzzy Data.* John Wiley & Sons, Vienna University of Technology, Austria, 2011. doi: `10.1002/9780470974414`.

[99] Virtanen, P.; Gommers, R.; Oliphant, T. E., et al.: SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020) 261–272. doi: `10.1038/s41592-019-0686-2`.

[100] Zadeh, L. A.: Fuzzy Sets. *Information and control* 8 (1965). doi: `10.1016/S0019-9958(65)90241-X`.

Appendix

In the following, inheritance graphs of the classes implemented in PUQPY are shown. Each of the graphs is based on the output of DOXYGEN and is present in the documentation. Abstract classes are grayed out, that is classes not meant to be used to actually instantiate an object of.

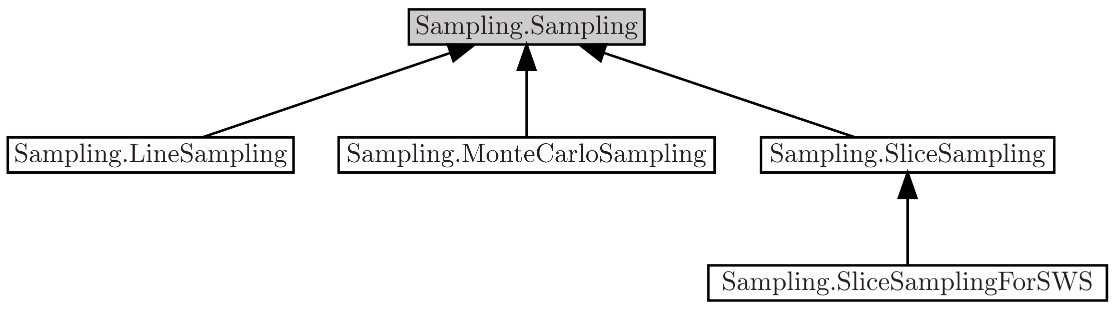


Figure 1: Hierarchy of sampling algorithms

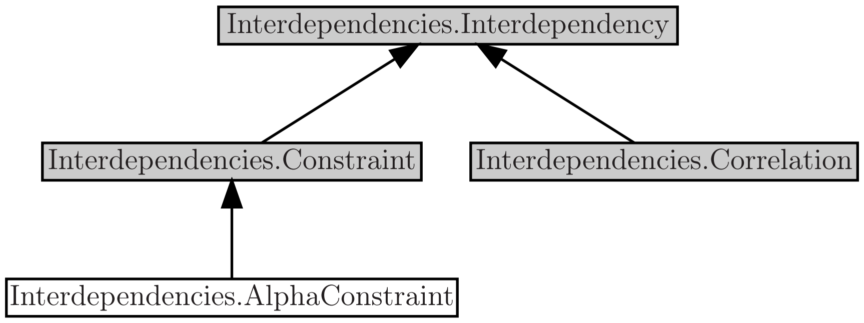

Interdependencies.Interdependency
Interdependencies.Constraint
Interdependencies.Correlation
Interdependencies.AlphaConstraint

Figure 2: Hierarchy of Interdependencies

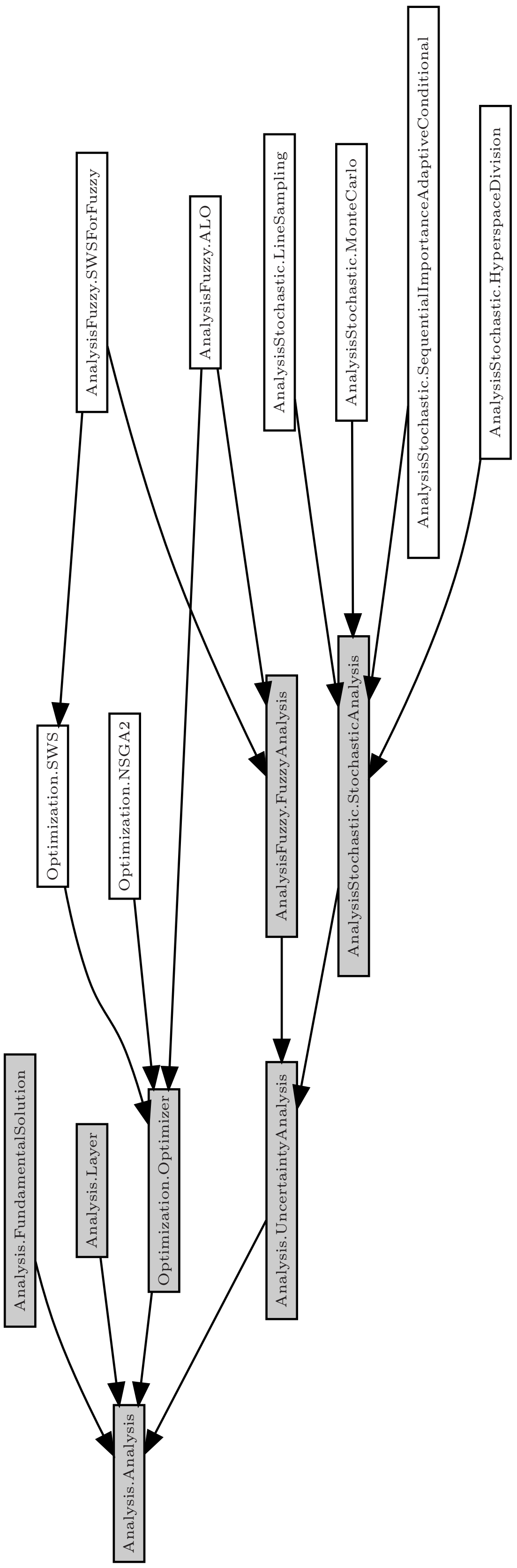

AnalysisStochastic.SequentialImportanceAdaptiveConditional
AnalysisStochastic.HyperspaceDivision
AnalysisStochastic.MonteCarlo
AnalysisStochastic.LineSampling
AnalysisFuzzy.ALO
AnalysisFuzzy.SWSForFuzzy
AnalysisStochastic.StochasticAnalysis
AnalysisFuzzy.FuzzyAnalysis
Optimization.NSGA2
Optimization.SWS
Analysis.UncertaintyAnalysis
Optimization.Optimizer
Analysis.Layer
Analysis.FundamentalSolution
Analysis.Analysis

Figure 3: Hierarchy of analyses

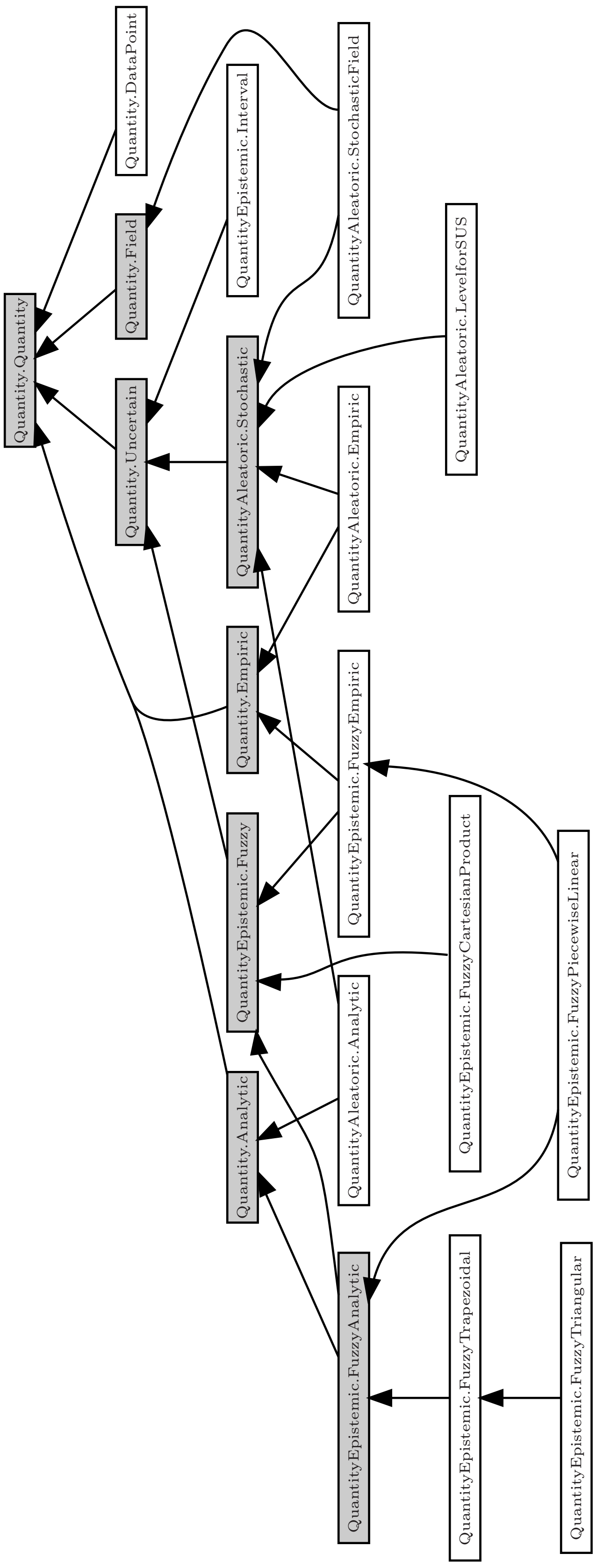

Quantity.Quantity
Quantity.DataPoint
Quantity.Field
Quantity.Uncertain
Quantity.Empiric
Quantity.Analytic
QuantityEpistemic.Interval
QuantityAleatoric.Stochastic
QuantityEpistemic.Fuzzy
QuantityAleatoric.StochasticField
QuantityAleatoric.Empiric
QuantityEpistemic.FuzzyEmpiric
QuantityAleatoric.Analytic
QuantityEpistemic.FuzzyAnalytic
QuantityAleatoric.LevelforSUS
QuantityEpistemic.FuzzyCartesianProduct
QuantityEpistemic.FuzzyTrapezoidal
QuantityEpistemic.FuzzyPiecewiseLinear
QuantityEpistemic.FuzzyTriangular

Figure 4: Hierarchy of quantities