Uncertainty Modelling for Engineers

Feb 16, 2021

CONTENTS

How to use this book

This book is intended to serve as a practical introduction to uncertainty quantification for engineers. It is intended to be particularly useful for those beginning a graduate course of study. Basic concepts are explained in detail, and relevant literature is summarised for more advanced concepts. After reading the book and studying the worked examples you should be able to implement a basic uncertainty quantification workflow.

The text of the book is largely adapted from [128].

I have included interactive code examples in python. Where possible the code examples use primitive python datatypes and commonly used packages for clarity.

A pdf of the book can be downloaded from here.

How to make a contribution

This book is an open source project; all readers are heartily invited to contribute with corrections and improvements.

To make a contribution you should first familiarise yourself with JupyterBook. If you are making a major change you may first wish to discuss the change by opening a GitHub issue.

You should fork the repository and make your desired changes to the files locally. Once this is done you can preview your changes locally by running `jupyter-book build uncertainty-book/` in the project directory. Do not commit the output files from the build process. You may have to install the project requirements, if you have not already done so. Once you are happy with your changes you should commit and push your changes, then make a pull request which we can review.

How to cite

You can reference this book through the project's Zenodo archive using DOI: 10.5281/zenodo.4483793.

The citation will look something like:

> Jonathan Sadeghi. (2021, January 31). Uncertainty Modelling for Engineers: Version 0.1.1 (Version v0.1.1). Zenodo. http://doi.org/10.5281/zenodo.4484026

Part I

Introduction

CHAPTER

ONE

MOTIVATION

1.1 Uncertainty in engineering simulation

Since the advent of the computer, engineering simulation has become an invaluable tool for the design and analysis of systems. Simulation allows engineers to bridge the gap between theoretical models of systems and empirical evidence, whilst making predictions about yet to be constructed systems [127]. Some important applications of this include structural engineering [86], aeronautical structures [86], and petroleum reservoir engineering [13].

The engineer's theoretical model of the system's physics can be defined by a mathematical function or a more complex simulation, and this theoretical model will depend upon associated parameters which determine the specific properties of the system under consideration. The physical model to be used may be motivated by the engineer's expert judgement, or prescribed by a relevant design standard document. Provided the model's parameters are known, the model can be used to make predictions about the system. In some cases, e.g. well known material properties, the parameters to be used will also be prescribed by the design standard document. If this is not the case, the engineer must identify these parameters from data or expert judgement. Hence, in many realistic situations these parameters will not be known exactly, and therefore will be associated with some uncertainty. If the system is not well understood, then the theoretical model of the system's physics may itself be uncertain. However, in many cases this situation can be dealt with by adding more uncertain parameters to the model, thereby increasing the model's degrees of freedom. This uncertainty will be reflected in the predictions made by the model, hence the ability of the system to meet some specified objective, e.g. safe operation, now becomes uncertain. In essence, this motivates the well known structural reliability analysis problem, where we wish to calculate the probability that the system under uncertainty doesn't meet a specified objective, which is referred to as the failure probability of the system [106].

1.2 Reliability engineering

Researchers in the discipline of reliability engineering have proposed many techniques to solve the reliability analysis problem. Most generally, Monte Carlo simulation can solve any reliability analysis problem with arbitrary accuracy, given sufficient samples of the uncertain system parameters and evaluations of the system model [106]. If the failure probability of the system is small, which is typical in most realistic engineering problems, then the number of model evaluations required increases significantly, and creates a bottleneck to the calculation. Therefore, in practice, more efficient methods are required to solve the reliability analysis problem for expensive computational models with many inputs. These include approximate methods, e.g. the First Order Reliability Method (FORM) and advanced simulation techniques, e.g. line sampling, which require fewer samples of the system model [86] [26].

Alternatively, since the cost of the analysis depends strongly upon the cost of evaluating the system model, one may attempt to replace the expensive system model with a cheaper surrogate, known as a metamodel. This metamodel is usually obtained by using machine learning technologies to learn a function which is a sufficiently accurate representation of the true model. Well known metamodels applied in reliability engineering include neural networks, response surfaces (polynomial regression), polynomial chaos, and Kriging (Gaussian process emulators) [86] [85] [129]. If the metamodel is inaccurate then this can introduce additional uncertainty into the calculation, and typically this must be traded off

against time required to create the metamodel. In any case, the uncertainty in the metamodel should be quantified and its influence on the failure probability of the system stated. As such, the problem is challenging and does not yet have an entirely satisfactory solution, although significant progress has been made in recent years.

1.3 Models of uncertainty

Many techniques exist for modelling uncertainty, and therefore the chosen uncertainty model is also, to an extent, an engineering judgement. This judgement is usually based on the type of uncertainty being modelled, and usually two types of uncertainty are considered; epistemic uncertainty and aleatory uncertainty [87] [26]. Broadly speaking, epistemic uncertainty represents uncertainty which originates from a lack of knowledge, and aleatory uncertainty represents uncertainty which originates from natural variability, i.e. stochasticity.[1] Again, some guidelines are available in design standards or regulations, for example the United States Nuclear Regulatory Commission often suggests using Bayesian probability theory [130] [131].

Bayesian probability theory is a logically consistent method of reasoning under uncertainty, though it has been shown to lack empirical justification in some circumstances, e.g. [6]. Efficient computational methods exist to identify many probabilistic models for uncertain variables in the Bayesian paradigm, in addition to convenient analytic techniques [1]. In recent decades, several extensions to the traditional probabilistic models for uncertainty have been proposed, e.g. Dempster-Shafter Theory, probability boxes, and random sets [26] [19] (often referred to as specific manifestations of *imprecise probabilities*). These methods enable reasoning with imprecise data and a severe lack of prior information.

Imprecise data consists of data where each measurement is not specified by a real number (sometimes referred to as crisp measurements), but instead the data falls within certain bounds which can be characterised. Scarce data refers to the case where insufficient data is available to accurately identify an unknown model parameter. Therefore our knowledge of the parameter places undue weight on our prior belief about the parameter. In such cases an engineer may wish to check the sensitivity of their model's predictions to the chosen prior, and it is therefore essential that the engineer can accurately represent uncertainty in their prior belief about a parameter [132] [133]. Crucially, imprecise probabilities offer a method of reasoning with uncertainty which is more flexible and hence requires fewer assumptions than traditional probabilistic methods.

Despite these advantages, working with imprecise probabilities in practice introduces some new difficulties. Since the uncertainty model is more complex, the computational techniques required to perform computations are also more complex. Typically, this computation involves some optimisation in addition to the already expensive Monte Carlo simulation [26]. Hence, in recent years, researchers have described techniques to solve the reliability analysis problem with imprecise probability models which apply similar approximations to those used in the straightforward probabilistic case.

[1] Note that in some cases this distinction is unclear. For example when a very simple model is used for the behaviour of a coin, the outcome of the coin flip may appear to be random. However, one could imagine a situation where the kinematics of the coin can be simulated exactly using Newton's laws of mechanics, and the only uncertainty in the outcome of the coin flip is caused by lack of knowledge in the coin's initial position and velocity. Chaotic systems (e.g. the Lorenz attractor), where the future evolution of the system depends strongly on the initial conditions, may appear to be random, for example when the model considered for the system is insufficiently detailed and the initial conditions are not known with sufficient accuracy.

CHAPTER

TWO

STRUCTURE OF THIS BOOK

The topics discussed in the previous section will be addressed in the following three chapters.

Chapter *Models of Uncertainty* reviews uncertainty models, i.e. models describing the uncertainty in a variable, or set of variables. We review techniques to construct these models from data, and expert opinion, and describe techniques for converting between common models. This chapter describes probability boxes in detail, and shows how traditional probability distributions emerge as a particular case of a probability box.

Chapter *Machine Learning of Regression Models* reviews machine learning techniques for creating regression models, which are used as metamodels in engineering. Regression models differ from the uncertainty models described in Chapter *Models of Uncertainty*, as they model the behaviour of a uncertain variable which depends on the behaviour of another variable. In this chapter, the theory behind interval predictor models is described in detail, and compared to traditional techniques in statistical learning.

Chapter *Reliability Analysis* describes the well known reliability analysis problem in engineering, where one wishes to calculate the probability that the performance of a system under the influence of uncertainty meets a particular design condition. State-of-the-art techniques for efficiently solving this problem with random variables and probability box variables are reviewed.

Part II

Models of Uncertainty

CHAPTER

THREE

OVERVIEW OF UNCERTAINTY MODELS

The purpose of uncertainty models is to allow analysts to move beyond computation with point values of variables. By using an uncertainty model one can describe the variability or lack of knowledge in a parameter, and consider how this affects the result of a calculation. In this chapter, we describe the process of quantifying the uncertainty in a set of variables independently of the values of other variables, which is known as *generative modelling* (the alternative case, *regression modelling* is described in the subsequent chapter).

As a concrete example, consider predicting the weather at a particular location, on a particular day. A model to predict the weather on Tuesday independent of other information would be considered a generative model. If the model predicted the weather on Tuesday, given the weather conditions on Monday, which were already known, this would be termed a regression model.

So far we have avoided discussing any type of uncertainty model in particular, since many different models of uncertainty exist. The particular model chosen for an application may be selected based on a number of factors, including its ability to be created from the type of data available, ease of computation, and the desired properties of predictions made by the model.

Models which can be created from imprecise data, i.e. interval data, often result in less informative predictions than models which cannot. However, uninformative predictions are not necessarily undesirable, if the predictions are a true representation of the state of knowledge of the analyst or engineer. Hence the chosen uncertainty model should truthfully represent the level of uncertainty of the analyst, given the available information.

It should also be noted that, contrary to common opinion, particular classes of uncertainty model do not necessarily restrict the type of uncertainty which can be represented. In some cases, a probabilistic model may allow epistemic uncertainty to be communicated [1], and in principle one could use an interval model to represent aleatory uncertainty — though usually probabilistic models are more useful representations of aleatory uncertainty.

The ease of calculation should also be considered when the class of uncertainty model to be used is chosen. If the speed of calculation is very important, because a decision based on the predictions of the model must be made with severe time limitations, then it could actually be *more unsafe* to choose an uncertainty model which is *more accurate* but reduces the speed with which predictions can be made. For example, this could be the case in an online safety system in a power plant [2]. However, in most situations the speed with which predictions must be made is more flexible, and therefore it is usually preferable to choose an uncertainty model which is accurate, and then use approximations or efficient computational methods to compute the desired prediction.

With this in mind, in this section several uncertainty models will be presented, including probabilistic models, non-probabilistic models, and imprecise probabilistic models. The type of information which can be used to create the models will be discussed, and their amenabilities to calculation will be compared.

3.1 Probabilistic models of uncertainty

3.1.1 Overview

Probabilistic models of uncertainty require the definition of a probability space, which is given by $(\Omega, \mathcal{F}, P)$, where Ω represents the space of all possible outcomes (the sample space), $\mathcal{F}$ is a σ algebra representing the set of events, where each event is a set containing outcomes, and $P \in [0, 1]$ represents probabilities which are assigned to the events in $\mathcal{F}$ [1]. Note that the probability assigned to the space of all outcomes is 1, $P(\Omega) = 1$, and the probability of the empty set of outcomes is zero, $P(\emptyset) = 0$. By definition, for an outcome $\omega \in \Omega$ and its complementary outcome ω^C, their intersection is the empty set, $\omega \cap \omega^C = \emptyset$, and their union is equal to the entire sample space, $\omega \cup \omega^C = \Omega$. It follows that $P(\omega) + P(\omega^C) = 1$.

The aim of defining the probability space is usually to obtain the random variable $X : \Omega \to E$, which is a function from the set of outcomes Ω to the measurable space E. The probability that X takes a value falling inside in the set S is given by

$$\Pr(X \in S) = P(\{\omega \in \Omega | X(\omega) \in S\}).$$

When the measurable space is the real line, $E = \mathbb{R}$, the cumulative distribution function (CDF) can be obtained by calculating the probability of the event that that X is less than a particular value x, $\{X < x\}$. Therefore the cumulative distribution function is given by

$$F_X(x) = P(X \leq x).$$

The well known probability distribution function (PDF) of the variable, $p_X(x)$, is the gradient of the cumulative distribution function, $p_X(x) = \frac{dF_X(x)}{dx}$. The probability density represents the relative likelihood of a random variable taking a particular value. Note that, as follows from our definition, the cumulative distribution must increase monotonically with x, and as such the probability density function is always positive. In addition, the integral of the probability density function over the whole real line will always be equal to 1, since $\lim_{x\to\infty} F_X(x) = 1$ and $\lim_{x\to-\infty} F_X(x) = 0$ [21].

Note that important properties of the random variable are summarised by the mean of the random variable,

$$\mu_X = \mathbb{E}[X] = \int_{\mathbb{R}} x p_X(x) dx,$$

where $\mathbb{E}$ is the expectation operator, and the variance

$$\mathrm{Var}(X) = \mathbb{E}[(X - \mu_X)^2] = \int_{\mathbb{R}} (x - \mu_X)^2 p_X(x) dx,$$

which is sometimes quoted in terms of the standard deviation, $\sigma_X = \sqrt{\mathrm{Var}(x)}$ [21]. The standard deviation is sometimes expressed as the coefficient of variation of a varible (CoV), which is defined as $\mathrm{CoV} = \frac{\mu_X}{\sigma_X}$.

In practical calculations, one is not usually concerned with the probability space $(\Omega, \mathcal{F}, P)$, because defining either the probability distribution function or cumulative distribution function by assigning probability density to the possible values for X is usually sufficient for calculations to proceed. The measure theoretic framework for probability theory which we have summarised here was introduced by [31]. An equivalent theory was derived by extending Boolean logic to assign quantitative values of truthfulness to statements, termed plausibilities, in the seminal work of [1].

The probability of an event may be considered in terms of probabilities of sub-events, for example if $A = A_1 \cap A_2$, where $\cap$ represents the logical and operation, then $P(A) = P(A_1 \cap A_2)$. This can be evaluated as $P(A_1 \cap A_2) = P(A_1|A_2)P(A_2)$, by definition. $P(A_1|A_2)$ is the probability that A_1 is true, given that A_2 is true. The dependence between A_1 and A_2 is encoded in $P(A_1|A_2)$; $P(A_1|A_2) = P(A_1)$ if A_1 and A_2 are independent events, so that $P(A) = P(A_1)P(A_2)$. If the dependence is not known then bounds for $P(A)$ can be established, and this is discussed in [32]. Similar bounds are available for the logical or operation, $\cup$.

One may model dependencies between variables by considering the joint distribution over more than one variable, e.g. $p(x_1, x_2) = P(X_1 = x_1 \cap X_2 = x_2)$, where X_1 and X_2 are two random variables. Traditionally, a generative model is

defined as a joint probability distribution. One may summarise the properties of a joint distribution using the co-variance between two variables

$$\text{cov}(X_1, X_2) = \mathbb{E}[(X_1 - \mu_{X_1})(X_2 - \mu_{X_2})] = \int_{\mathbb{R}} \int_{\mathbb{R}} (x_1 - \mu_{X_1})(x_2 - \mu_{X_2}) p(x_1, x_2) dx_1 dx_2.$$

Taking the co-variance of a variable with itself yields the variable's variance, $\text{cov}(X_1, X_1) = \text{Var}(X_1)$ [21].

3.1.2 Worked example : elementary probability theory

The amount of rainfall in mm on a particular day is given by the normal random variable $p(x) = \mathcal{N}(\mu = 10, \sigma = 1)$

What is the probability of less than 8mm of rainfall?

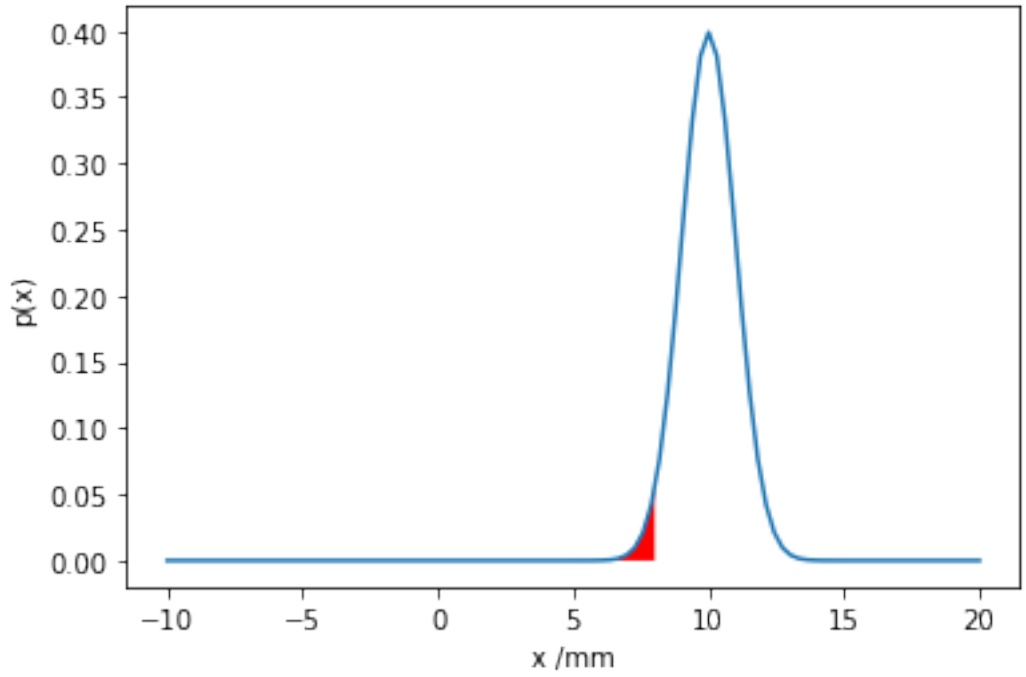
The probability of less than a certain amount of rainfall is obtained by integrating the probability density function up to that amount. This is equivalent to the cumulative density function at that value.

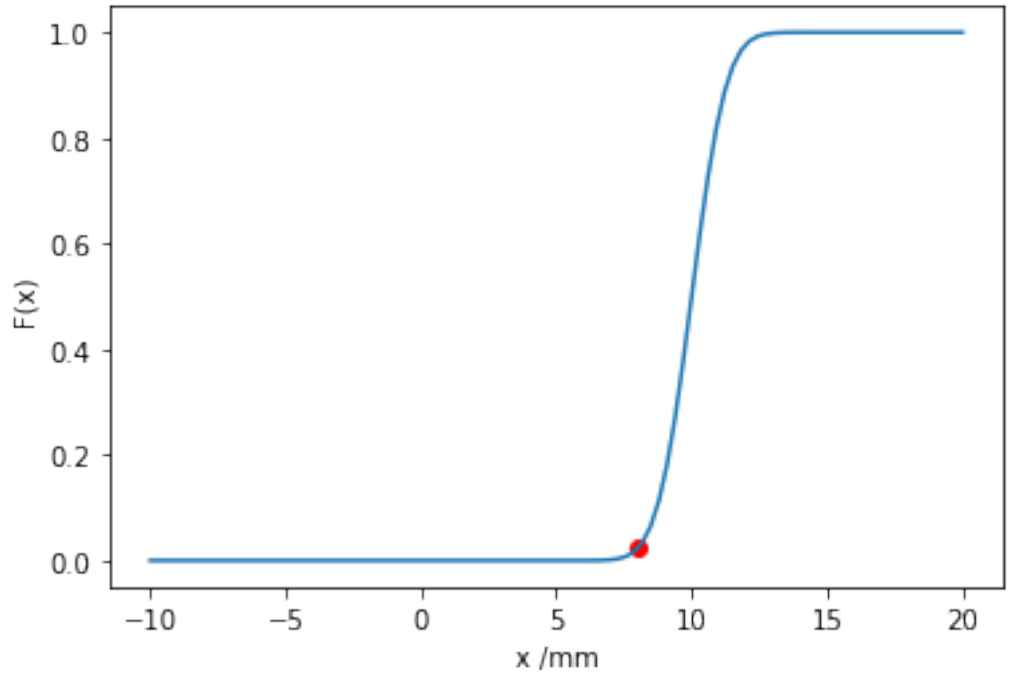
$P(\text{rainfall} < 8mm) = \int_{-\infty}^{8mm} p(x)dx = F(x = 8mm)$

In the following code snippet we evaluate the CDF of the random variable using `scipy.stats`. Then we plot the PDF and CDF of the random variable using `matplotlib` and show the relevant integrated area of the random variable in red.

```
import matplotlib.pyplot as plt
import scipy.stats
import numpy as np

print(scipy.stats.norm.cdf(8, loc=10, scale=1))


plot_x_range = np.linspace(-10, 20, 100)
plot_x_range_integrate = np.linspace(-10, 8, 100)
plt.plot(plot_x_range, scipy.stats.norm.pdf(plot_x_range, loc=10, scale=1))
plt.fill_between(
    plot_x_range_integrate,
    0,
    scipy.stats.norm.pdf(plot_x_range_integrate, loc=10, scale=1),
    facecolor="red"
)
plt.xlabel("x /mm")
plt.ylabel("p(x)")
plt.show()

plot_x_range = np.linspace(-10, 20, 100)
plt.plot(plot_x_range, scipy.stats.norm.cdf(plot_x_range, loc=10, scale=1))
plt.scatter(8, scipy.stats.norm.cdf(8, loc=10, scale=1), color='red')
plt.xlabel("x /mm")
plt.ylabel("F(x)")
plt.show()
```

```
0.022750131948179195
```





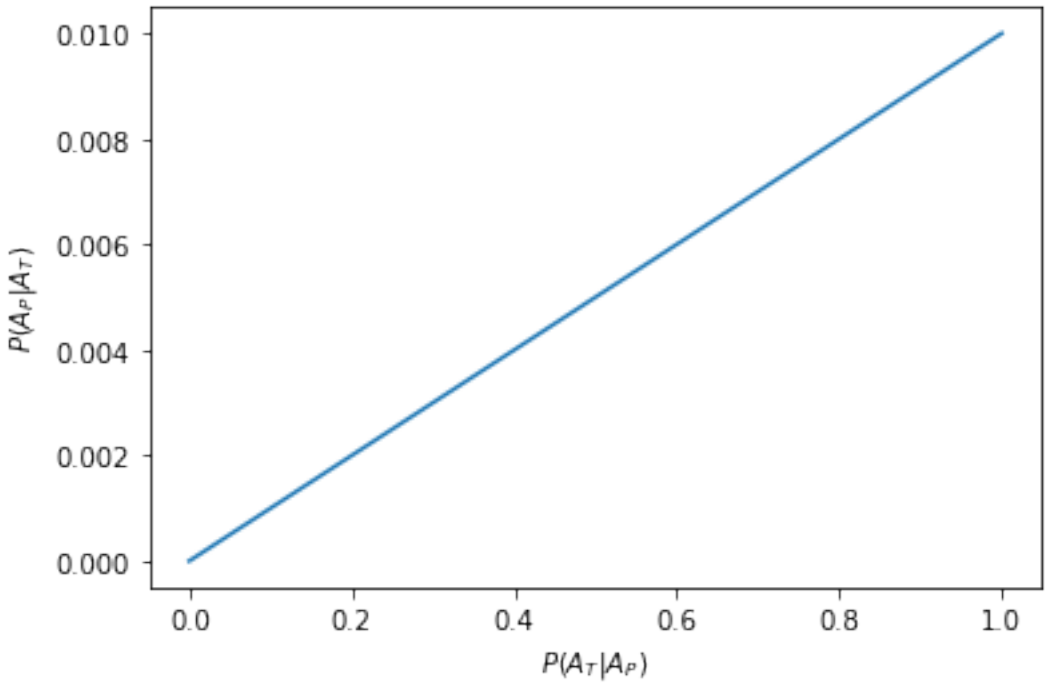
Do you notice anything strange about this model? The probability of a negative amount of rainfall is small, but non-zero. Can you think of a way to change the support of the probability distribution and thereby improve the model?

3.1.3 Worked example : elementary probability theory

Consider two events: a patient having a disease A_P, and the patient testing positive for the disease A_T. The two events are associated with the probabilities $P(A_P) = 0.001$ for the patient having the disease, $P(A_T) = 0.1$ for a random patient testing positive and $P(A_T|A_P) = x$ for the patient testing positive if they have the disease. Plot the probability that a patient has the disease if they test posititive, $P(A_P|A_T)$, as a function of x:

We know $P(A_P|A_T) = \frac{P(A_P \cap A_T)}{P(A_T)} = \frac{P(A_T|A_P)P(A_P)}{P(A_T)}$

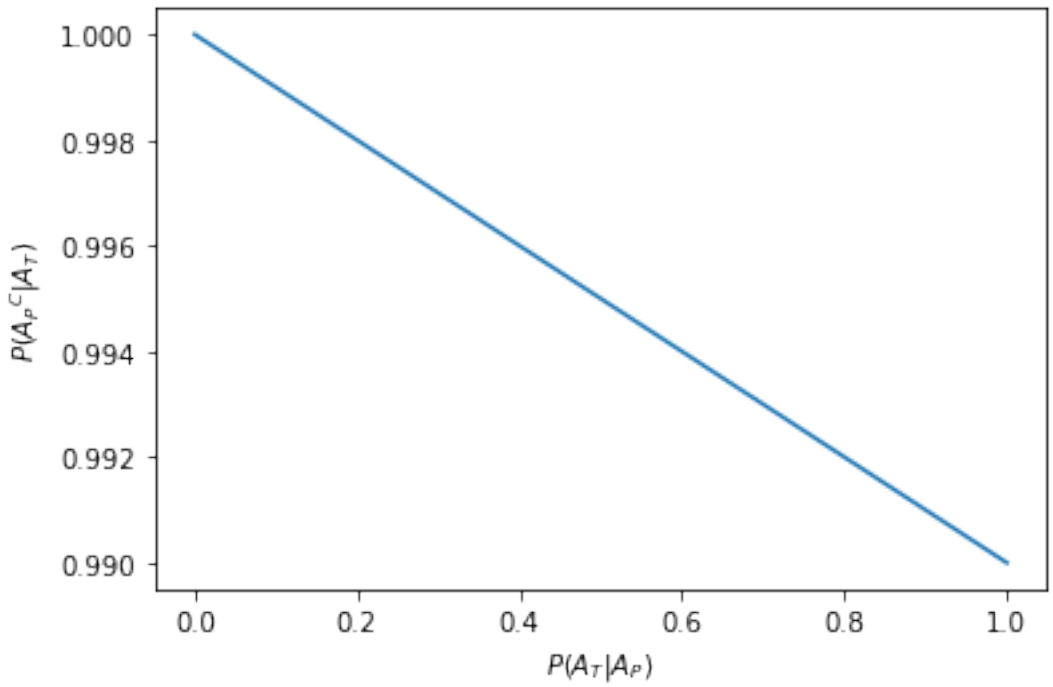
```
prob_infected = 0.001
prob_positive_test = 0.1


def prob_true_positive(x): return x * prob_infected / prob_positive_test


plot_x_range = np.linspace(0, 1, 10)
plt.plot(plot_x_range, prob_true_positive(plot_x_range))
plt.xlabel("$P(A_T|A_P)$")
plt.ylabel("$P(A_P|A_T)$")
plt.show()
```



So even with a perfect test, $P(A_T|A_P) = 1$, the probability of having the disease after testing positive is only 0.01. This makes sense because very few people have the disease but the probability of the test being positive is much higher, i.e. 0.1 >> 0.001

This is clear when we plot the number of false positives, $P({A_P}^C|A_T) = 1 - P(A_P|A_T)$

```
plt.plot(plot_x_range, 1 - prob_true_positive(plot_x_range))
plt.xlabel("$P(A_T|A_P)$")
plt.ylabel("$P({A_P}^C|A_T)$")
plt.show()
```



3.1.4 Computation with Probabilistic Models

The expectation (mean value) of a general function $g(x)$ with respect to an uncertain variable, which is modelled with a probability distribution $p(x)$, can be evaluated as

$$\mathbb{E}[g(X)] = \int_{\mathbb{R}^N} g(x)p(x)dx, \tag{1}$$

where we now allow x to have multiple components.

In the most general case, one can draw realisations, known as samples, from the random variable by selecting values for the variable in the ratio of their assigned probability densities. Given M samples drawn from $p(x)$, $\{x_{(1)}, \ldots, x_{(M)}\}$, $\mathbb{E}[g(X)]$ can be approximated by the Monte Carlo estimator

$$\mathbb{E}[g(X)] \approx I = \frac{1}{M}\sum_{i=1}^{M} g(x_{(i)}). \tag{2}$$

The error of the Monte Carlo estimator is given by

$$\frac{\sigma_g}{\sqrt{M}} = \frac{\sqrt{\sum_{i=1}^{M}(g(x_{(i)}) - I)^2}}{\sqrt{M(M-1)}},$$

where σ_g^2 is the unbiased sample estimator of the variance of g [33]. Clearly the error in the estimator decreases with $\frac{1}{\sqrt{M}}$, so collecting more samples will slowly decrease the error in the estimator [34]. However in many circumstances collecting more samples of M is not feasible, and σ_g may be large.

Therefore, in practice, more efficient methods are used to compute the expectation. In *Reliability Analysis* efficient methods for solving a specific form of this integral from engineering reliability analysis will be explained. However, it is useful to note there are some more general efficient approaches for computing an approximation of the expectation in (1).

For example, provided that $g(x)$ can be differentiated, one can attempt to approximate the expectation in (1) with a Taylor expansion

$$\mathbb{E}[g(X)] = \mathbb{E}[g(\mu_X + (X - \mu_X))] \approx \mathbb{E}[g(\mu_X) + g'(\mu_X)(X - \mu_X) + \frac{1}{2}g''(\mu_X)(X - \mu_X)^2 \ldots],$$

where μ_X is the mean of random variable X. When $g(x)$ is linear, the the expectation in (1) can therefore be calculated trivially, since $X - \mu_X = 0$ and $g''(x) = 0$. If $g(x)$ is non-linear then using the approximation $\mathbb{E}[g(X)] \approx \mathbb{E}[g(\mu_X)]$ is correct only to first order [35]. At the expense of generality, the expectation is evaluated using only two evaluations of $g(x)$, which is a clear advantage when $g(x)$ is expensive to evaluate, or the computation of $\mathbb{E}[g(X)]$ is required quickly. This idea is applied to engineering in the Kalman Filter and Extended Kalman Filter [35].

[36] proposed a more accurate approximate method for approximating the expectation in (1) which requires few evaluations of $g(x)$, and does not require $g(x)$ to be differentiated. The Unscented Transformation, sometimes referred to as deterministic sampling, requires the computation of so called sigma points which are specially chosen represent the covariance of the input distributions. It is then only required to compute $g(x)$ for these sigma points, after which a weighted average is used to obtain the approximation of the expectation in (1). [35] demonstrate that the Unscented Transformation is accurate to at least second order, but potentially to third or forth order in some cases. The number of sigma points required, and hence the cost of the computation, depends linearly on the dimensionality of x. Although this approach is more generally applicable and accurate than the Taylor Series method, the Unscented Transformation is limited to use cases where x has reasonably low dimensionality, because otherwise too many evaluations of $g(x)$ are required. The method has been applied to engineering in the Unscented filter [35] and uncertainty quantification in the nuclear industry by [37] and [38].

In order to avoid these approximations, techniques can be used to obtain more accurate approximations of $g(x)$ in specific cases (metamodels), or alternatively efficient sampling strategies can be used. This is discussed in *Reliability Analysis* for application to reliability analysis.

3.1.5 Worked example : Computation with a probabilistic model

Again, the amount of rainfall in mm on a particular day is given by the normal random variable $p(x) = \mathcal{N}(\mu = 10, \sigma = 1)$. It is known that the number of umbrellas sold in London on a particular day is given by $g(x) = 1000x^2$.

Estimate the number of umbrellas sold in London given our uncertain model of rainfall.

$$\text{Umbrellas sold} = \int_{-\infty}^{\infty} p(x)1000x^2 dx$$

Analytically evaluating this integral in a popular computer algebra system yields Umbrellas sold $= 101000$.

```
# First try Monte Carlo integration
n_samples = 100
samples_x = scipy.stats.norm(loc=10, scale=1).rvs(n_samples)


def g(x): return 1000 * x ** 2


umbrellas_sold = np.mean(g(samples_x))

print("We expect that {} umbrellas were sold".format(umbrellas_sold))

error = np.std(g(samples_x)) / np.sqrt(n_samples-1)

print("The error of this Monte Carlo estimator is {}".format(error))
```

```
We expect that 101287.30984936524 umbrellas were sold
The error of this Monte Carlo estimator is 2063.720321793903
```

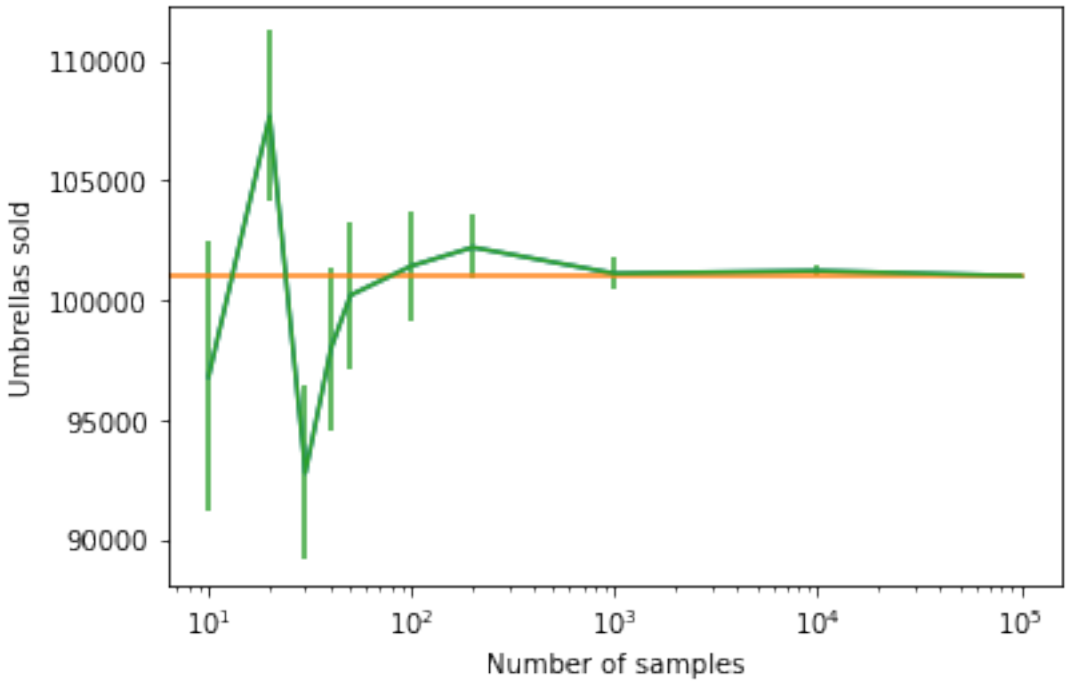
Using `{}` in a string with `.format()` is a compact and readable method of inserting a quantity into a print statement. The scipy stats library is used for the random variables, and the expectation is taken using `np.mean`.

Note that we divide the standard deviation from numpy by $M - 1$, since the output of `np.std` is the standard deviation and *not* the unbiased estimator for sample standard deviation.

Most importantly the Monte Carlo simulation agrees with the analytic result!

We can plot how the result changes with an increasing number of samples.

```
def monte_carlo_simulation(n_samples):
    """
    Monte Carlo Simulation for number of umbrellas sold in London on a
      particular day
        Args:
            n_samples (int): Number of samples to be made in Monte Carlo
              simulation
        Returns:
            Number of umbrellas sold (float): The expected number of umbrellas
              sold
            error (float): The error in the estimated number of umbrellas sold
    """
    samples_x = scipy.stats.norm(loc=10, scale=1).rvs(n_samples)
    def g(x): return 1000 * x ** 2

    umbrellas_sold = np.mean(g(samples_x))

    error = np.std(g(samples_x)) / np.sqrt(n_samples-1)
    return umbrellas_sold, error


number_of_samples = [10, 20, 30, 40, 50, 100, 200, 1000, 10000, 100000]
simulation_results = [monte_carlo_simulation(n) for n in number_of_samples]

plt.plot(number_of_samples, list(zip(*simulation_results))[0])
plt.plot([0, 100000], [101000]*2)
plt.errorbar(number_of_samples, *zip(*simulation_results))
plt.xlabel("Number of samples")
plt.ylabel("Umbrellas sold")
plt.xscale("log")
plt.show()
```



Notice that all simulation results are within 3 standard deviations of the true value.

The true value from the analytic calculation is shown with an orange line. The simulation results and error bars are shown in green.

Note that we used several useful features of python in the above code.

We use a docstring below the function definition to explain what the purpose of the function is and the expected inputs and outputs.

Note that the defined function returns two outputs - these are treated like a `tuple` object when inserted into the list. `zip(*simulation_results)` converts the list of tuples into a tuple of lists.

As in the previous example, the sampling is vectorised (i.e. no loop is used in generating the samples for each simulation) to reduce the time required for computation.

A *list comprehension* is used to tidily obtain the results from the Monte Carlo simulation and put these into a list.

To reduce the number of samples required we now evaluate the integral using a Taylor Series.

```
def g(x): return 1000 * x ** 2


def g_dash_dash(x): return 1000 * 2


loc = 10
scale = 1

zero_order_taylor_series = g(loc)
# first order contribution is zero!
second_order_term = g_dash_dash(loc) * 0.5 * scale
taylor_series_umbrellas = zero_order_taylor_series + second_order_term

print("Taylor series - umbrellas sold: {}".format(taylor_series_umbrellas))
```

```
Taylor series - umbrellas sold: 101000.0
```

The correct result is obtained exactly! This is because the third and higher derivatives of $g(x)$ are zero!

3.2 Set-based models of uncertainty

The two most common set-based models of uncertainty are convex sets and interval models. The popularity of these models can be attributed to the computational simplifications they enable, relative to non-convex set models of uncertainty. Set-based models are a method of characterising uncertainty, without describing the level of belief for each distinct value, as in probabilistic models. Alternatively, they can be seen as an initial way of prescribing the support for an as-yet undetermined probabilistic model.[1]

3.2.1 Interval models of uncertainty

An interval model of uncertainty is a set of numbers where any number falling between the upper and lower bounds of the interval is included in the set. In the interval notation, an interval uncertainty model is given by $X = [\underline{x}, \overline{x}] = \{x \in \mathbb{R} \mid \underline{x} \leq x \leq \overline{x}\}$ [39]. Note that interval models can also be defined for the case where the endpoints are excluded from the intervals, and in this case regular brackets are used instead of square brackets. Interval models can be re-parameterised in terms of their centre and interval radius, $c = \frac{\overline{x}+\underline{x}}{2}$ and $r = \frac{\overline{x}-\underline{x}}{2}$. Interval models usually don't express dependency between multiple variables.

Note that when using an interval model of uncertainty, the relative likelihoods of different values of the variable are unknown and therefore it is impossible to draw precise samples from the model in the same sense that one draws samples from a probability distribution. Put simply, an interval model represents complete lack of knowledge, apart from the bounds. For this reason, they are often preferred for modelling severe epistemic uncertainty, sometimes known as incertitude, rather than aleatory uncertainty, where the ability to model the variability of a quantity is the main focus [40].

Basic computation with interval models is typically computationally inexpensive since one can rely upon interval arithmetic for elementary operations. Complex functions and computer codes can be 'converted' to accept interval inputs by applying the so-called natural extension, where arithmetic operations are converted to interval arithmetic and real valued inputs are replaced with interval values [39]. However, applying the natural interval extension can result in gradual widening of the prediction interval during propagation through the code, due to repeated appearance of the same variables [39].

In many cases, one can create a Taylor expansion for $g(x)$ up to terms of a specific order in x, and then use an interval bound for the rounding error to rigorously and accurately bound $g(x)$ [41] [42] [43]. Intermediate linear Taylor models may be combined together to yield relatively tight bounds on the output with reduced computational expense, since computing the maximum of a multivariate linear function over a set of intervals is trivial. Engineers could choose to neglect the modelling of the remainder in cases where a rigorous bounding of the range of $g(x)$ over X is not required, or the Taylor model is sufficiently accurate. Alternatively, if $g(x)$ is represented a Bernstein polynomial, bounds on the range of $g(x)$ over X can be obtained analytically [44].

For complex functions or computer codes it may not be possible to rewrite the code in terms of interval arithmetic, as the code may be 'black-box' and therefore inaccessible. In these cases it may be necessary to rely upon numerical optimisation to compute approximate bounds on the range of $g(x)$. For example, if one wishes to make predictions about the output of a model $g(x)$ for $x \in [\underline{x}, \overline{x}]$ then one must evaluate

$$[\min_{x \in [\underline{x}, \overline{x}]} g(x), \max_{x \in [\underline{x}, \overline{x}]} g(x)]. \tag{3}$$

For general $g(x)$, a non-linear optimiser with the ability to respect bounds must be used to solve (3), for example Bayesian optimisation or a genetic algorithm. A brute force search could also be used to solve (3). If $g(x)$ is convex, then one may apply convex optimisation routines [45]. Due to the curse of dimensionality this optimisation becomes more difficult

[1] The support of a distribution is the set of values for which the probability measure is non-zero.

when there are many interval variables to be propagated, since the complexity for many convex optimisation algorithms increases with the dimensionality of the problem [46].

The method of Cauchy deviates offers an efficient alternative to computing finite difference gradients of $g(x)$ in order to apply a Taylor expansion for black-box functions [47]. The Cauchy deviates method evaluates $g(x)$ for a set of transformed samples drawn from Cauchy distributions and then uses the property that the linear combination of Cauchy distributed random variables is also Cauchy distributed with a known distribution parameter to bound $g(x)$.

3.2.2 Worked example : Interval Models

This time, the model of rainfall will be given by an interval. Imagine the rainfall in London on a particular day is known to be bounded in the interval $x \in [9, 11]$.

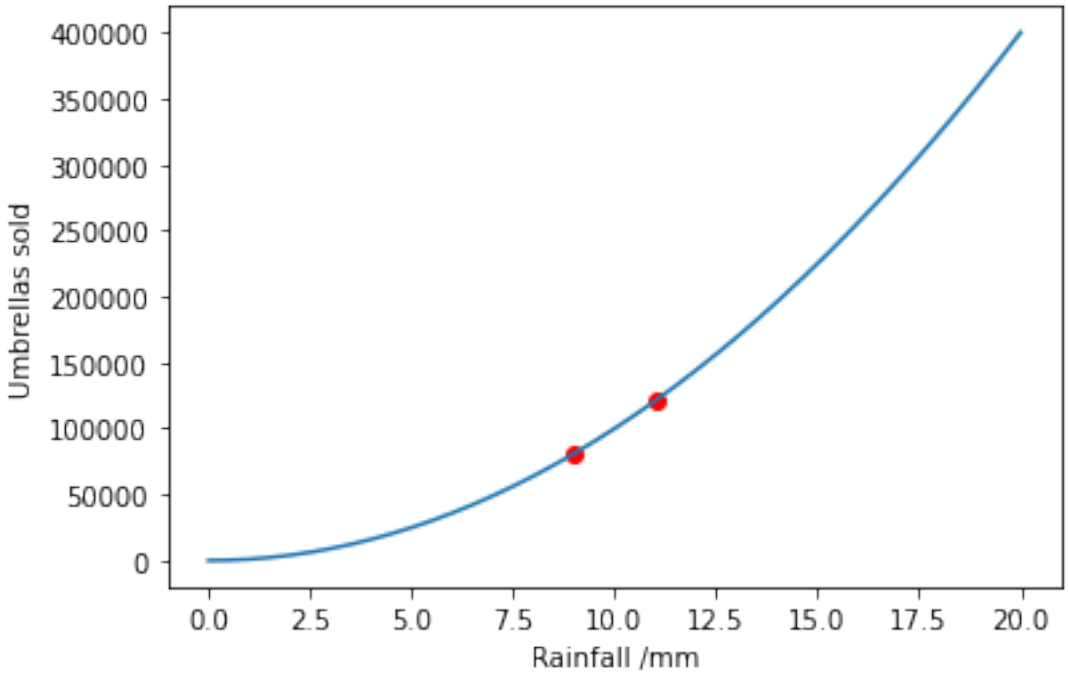
Again the number of umbrellas sold is given by $1000x^2$.

To bound the number of umbrellas sold we must evaluate $[\min_{x\in[9,11]} 1000x^2, \max_{x\in[9,11]} 1000x^2]$

This can be evaluated analytically since x^2 is monotonic in x for positive x:

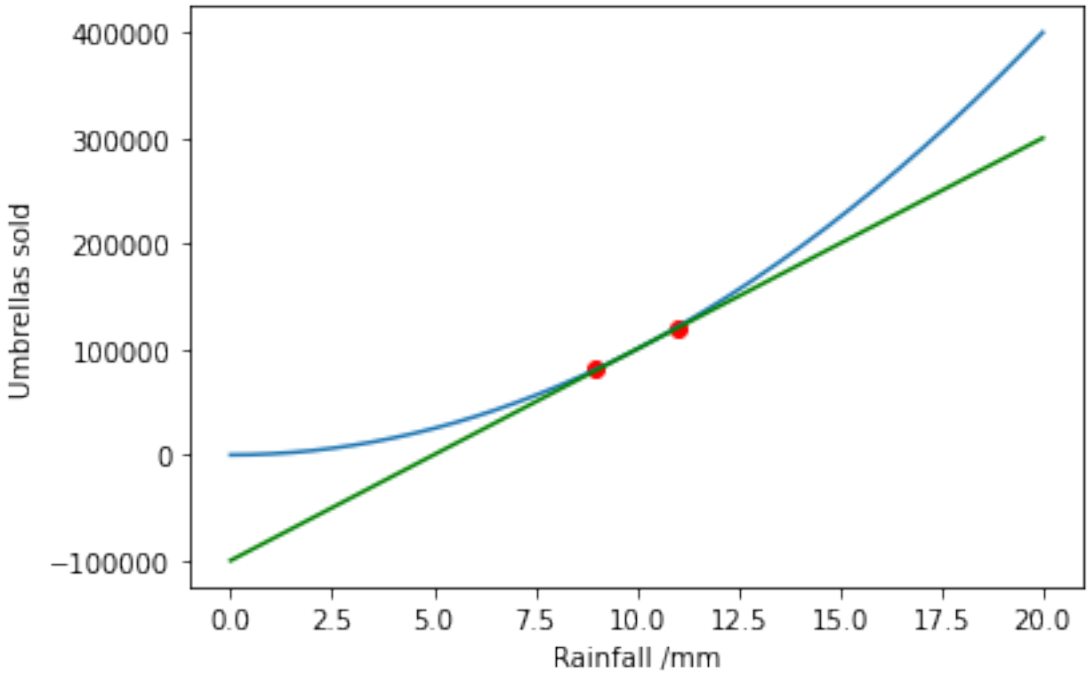
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize


def g(x): return 1000 * x ** 2


min_x = 9
max_x = 11

umbrellas_sold = [g(min_x), g(max_x)]
print("The number of umbrellas sold falls in the interval: {}".format(
    umbrellas_sold))

x_grid = np.linspace(0, 20, num=200)
plt.plot(x_grid, g(x_grid))
plt.scatter([min_x, max_x], g(np.array([min_x, max_x])), color="red")
plt.xlabel("Rainfall /mm")
plt.ylabel("Umbrellas sold")
plt.show()
```

```
The number of umbrellas sold falls in the interval: [81000, 121000]
```



This can also be evaluated approximately using a taylor series (shown in green):

```
def g_dash(x): return 1000 * 2 * x


x_mid = (max_x + min_x) / 2


def taylor_expansion(x): return g(x_mid) + (x - x_mid) * g_dash(x_mid)


x_grid = np.linspace(0, 20, num=200)
plt.plot(x_grid, g(x_grid))
plt.plot(x_grid, taylor_expansion(x_grid), color="green")
plt.scatter([min_x, max_x], g(np.array([min_x, max_x])), color="red")
plt.xlabel("Rainfall /mm")
plt.ylabel("Umbrellas sold")
plt.show()

umbrellas_sold = [taylor_expansion(min_x), taylor_expansion(max_x)]
print("The number of umbrellas sold falls in the interval: {}".format(
    umbrellas_sold))
```




```
The number of umbrellas sold falls in the interval: [80000.0, 120000.0]
```

Finally we use black box optimisation in `scipy.optimize` to evaluate the interval model:

```
min_umbrellas = scipy.optimize.minimize(g, x0=x_mid, bounds=[(min_x, max_x)])
neg_max_umbrellas = scipy.optimize.minimize(lambda x: -g(x),
                                            x0=x_mid,
                                            bounds=[(min_x, max_x)])
print("The number of umbrellas sold falls in the interval: {}".format(
    [min_umbrellas.fun[0], -neg_max_umbrellas.fun[0]]))
```

```
The number of umbrellas sold falls in the interval: [81000.0, 121000.0]
```

3.2.3 Convex set models of uncertainty

A convex set is defined as a set where, for any two points in the set, all points along the connecting line between the two points are also included in the set. Convex sets are useful as they are in many ways similar to interval models, but allow dependencies to be modelled between variables. [48] provide examples of how convex set models may be used in engineering practice. There is a deep connection between interval models and convex set models. An interval model with multiple variables would be represented as the specific case of a hyper-rectangular convex set. In addition, affine transformations of hyper-rectangular convex sets result in a class of models known as zonotopes [49].

The smallest convex set containing a particular set of data points is termed the convex hull of the dataset. Computing the convex hull of a dataset has complexity $\mathcal{O}(n \log n + n^{\lfloor \frac{d}{2} \rfloor})$ [50], where $\lfloor \cdot \rfloor$ is the floor operator which rounds a real number down to the nearest integer, and therefore in practice one often learns a simplified representation of the convex set with desirable computational properties.

ℓ_p ellipsoids (for $p > 1$) are a particularly useful case of a convex set, because they can be easily manipulated in calculations [51]. An ℓ_p ellipsoid is given by the set $X = \{x \mid \|x - c\|_{p,w} \leq r\}$, where the weighted ℓ_p norm is $\|x\|_{p,w} = (\sum_{i=1}^{N} |w_i x_i|^p)^{\frac{1}{p}}$, x_i are variables in a set of dimensionality N, $c \in \mathbb{R}^N$ represents the centre point of the set, and w_i are weights controlling the relative uncertainty in each variable [52]. p can be adjusted to control the correlation between

the variables. The case ℓ_∞ corresponds to the hyper-rectangle (interval) model, where there is no correlation between variables. When p is decreased the variables become more correlated.

Computation with convex set models is performed in the same way as with interval models, except the bounds on x are replaced with convex constraints:

$$[\min_x\{g(x) : x \in X\}, \max_x\{g(x) : x \in X\}], \tag{4}$$

where X is a convex set. If $g(x)$ is linear with known gradient, and X is a hyper-sphere or hyper-rectangle, then this optimisation program in (4) admits an analytic solution [53]. If the convex set is small then the analytic solution may be a reasonable solution when used with a Taylor expansion for $g(x)$, as with interval uncertainty models.

3.3 Imprecise probabilistic models of uncertainty

3.3.1 Probability Boxes

Probability boxes generalise probability distributions and intervals; they model a set of cumulative distribution functions. Probability boxes are used to communicate epistemic uncertainty in the precise form of a probability distribution [19]. In the particular limiting cases of no epistemic uncertainty and no aleatory uncertainty, traditional CDFs and intervals can be recovered from the probability box, respectively.

Broadly speaking, probability boxes can be split into two types. Distribution-free probability boxes consist of an envelope defined by two CDFs. Any CDF contained within the envelope is permitted, i.e. the probability box contains all cumulative distribution functions $F(x)$ which satisfy the envelope condition $\underline{F}(x) \leq F(x) \leq \overline{F}(x) \forall x$.

Distributional probability boxes consist of a conventional probability distribution where at least one parameter of the distribution is given as an interval rather than a crisp value, i.e. the probability box is given by the probability distribution $p_\theta(x)$ with parameters $\theta \in \Theta$, where Θ is a hyper-rectangular convex set. It is possible to perform a conversion from distributional probability boxes to distribution-free probability boxes, by finding a distribution-free probability box which encloses the distributional probability box. This conversion results in the loss of information about the distribution types enclosed, and hence the conversion can not be easily reversed. The envelope of a distributional probability box, which will be a distribution-free probability box, can be obtained by evaluating

$$[\underline{F}(x), \overline{F}(x)] = [\min_\theta F_\theta(x), \max_\theta F_\theta(x)],$$

where $F_\theta(x)$ is the CDF corresponding to the probability distribution $p_\theta(x)$.

3.3.2 Computation with Probability Boxes

Probability boxes are a specific case of a random set [26][27], and therefore when they are propagated through a calculation their propagation can be decomposed into two distinct parts; the propagation of a set of epistemic uncertain variables which fall within an uncertain hyper-rectangle $\theta \in \Theta$, and the propagation of an aleatory set of variables α, which are associated with a probability distribution. The result of propagation through the model is a probability box, rather than a single CDF. The propagation is non-trivial since the epistemic variables are intervals and have no probability distribution, which means that conventional Monte Carlo simulation cannot be applied. Two methods are commonly used to propagate probability boxes: Double Loop Monte Carlo (sometimes referred to as search or optimisation of the epistemic space), and integration of the aleatory variables [28].

For a distributional probability box the upper and lower expectation are defined by

$$\overline{\mathbb{E}}[g(X)] = \max_{\theta \in \Theta} \int_{\mathbb{R}^N} g(x) p_\theta(x) dx \tag{5}$$

and

$$\underline{\mathbb{E}}[g(X)] = \min_{\theta \in \Theta} \int_{\mathbb{R}^N} g(x) p_\theta(x) dx.$$

If $g(x)$ is linear or easily approximated by a Taylor series then similar approximation techniques to those discussed for probability distributions earlier in the chapter can be applied.

In the general case, to accurately approximate the upper expectation in (5) we can use the Monte Carlo estimator from (2) inside an optimisation routine:

$$\overline{\mathbb{E}}[g(X)] = \max_{\theta \in \Theta} \frac{1}{M} \sum_{i=1}^{M} g(x_{(i)}), \tag{6}$$

where $x_{(1)}, \ldots, x_{(M)}$ are drawn from $p_\theta(x)$. This is known as Double Loop Monte Carlo simulation, and can be applied only in the case of distributional probability boxes. This estimator can be shown to have a positive bias [29]. As is the case with computation with interval models, it is necessary to use a non-linear optimisation routine to evaluate the outer loop in the general case.

The Double Loop Monte Carlo method can be modified to propagate distribution-free probability boxes by drawing the samples required for (6) from a monotonically increasing staircase function, which is parameterised by θ and satisfies the envelope condition for the probability box under consideration. However, this approach will be computationally inefficient in general since a high dimensionality θ would be required to approximate the probability box with sufficient accuracy, and therefore another approach is required.

For a distribution-free probability box note that one can sample intervals from the probability box by generating random numbers between 0 and 1 and applying inverse transform sampling to $\overline{F}$ and $\underline{F}$, i.e. generate sampled intervals $[\underline{x}, \overline{x}] = [\underline{F}^{-1}(\alpha), \overline{F}^{-1}(\alpha)]$ by sampling $\alpha \sim \mathcal{U}(0, 1)$, where $\underline{F}^{-1}$ and $\overline{F}^{-1}$ are the inverse CDFs of the probability box envelopes and $\mathcal{U}$ is the uniform distribution. The process of generating a single sample is known as taking the α cut, and the sampled intervals constitute focal elements [26]. This process is shown in Fig. 1. One can then propagate these intervals through $g(x)$ using the techniques described for intervals in the previous section, e.g. using the natural interval extension of $g(x)$, or using black-box optimisation. Therefore one can calculate the upper expectation as

$$\overline{\mathbb{E}}[g(X)] = \frac{1}{M} \sum_{i=1}^{M} \max_{x \in [\underline{x}_{(i)}, \overline{x}_{(i)}]} g(x),$$

where $[\underline{x}_{(i)}, \overline{x}_{(i)}]$ are the intervals generated from inverse transform sampling. In this thesis we dedicate most of our attention to the case of Double Loop Monte Carlo simulation for distributional probability boxes.

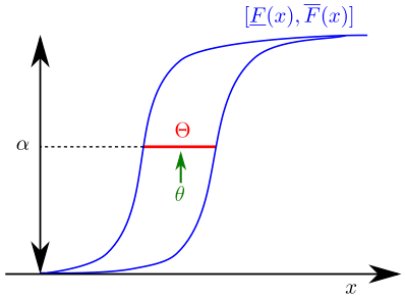


Fig. 1: Obtaining samples from a probability box.

3.3.3 Worked example : imprecise probability theory

The amount of rainfall in mm on a particular day is given by the normal probability box $p_\mu(x) = \mathcal{N}(\mu, \sigma = 1), \mu \in [9, 11]$.

What is the probability of less than 8mm of rainfall?

The probability of less than a certain amount of rainfall is obtained by integrating the probability density function up to that amount. However, a probability box is composed of many probability density functions. Therefore we must find the maximum and minimum such value. This is equivalent to evaluating the maximum and minimum cumulative density function at that value.

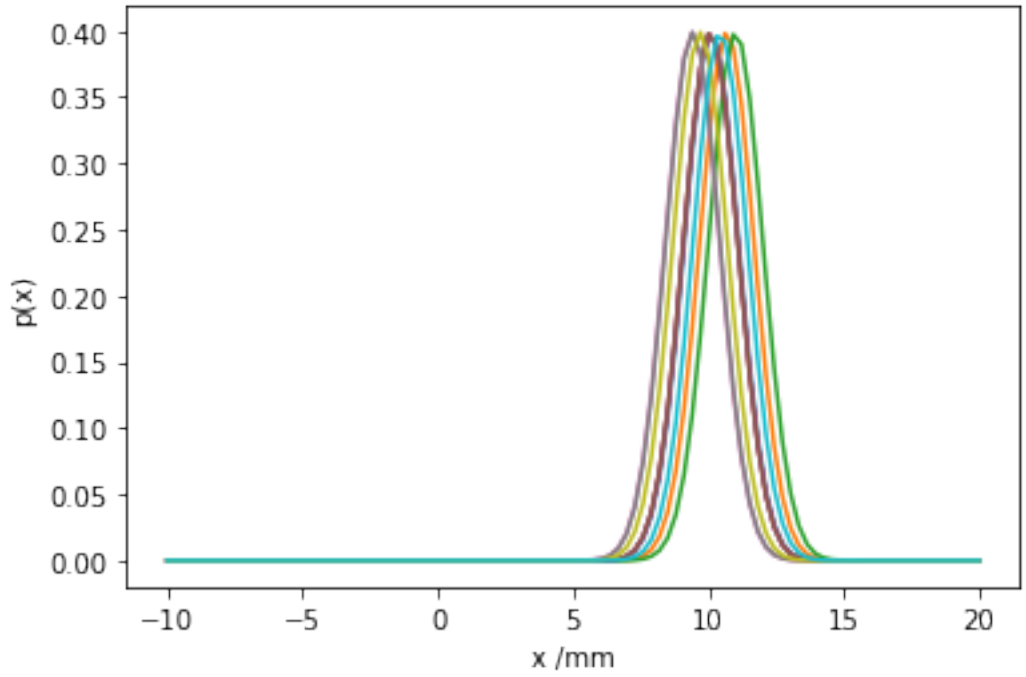
$P(\text{rainfall} < 8mm) < \max_{\mu \in [9,11]} \int_{-\infty}^{8mm} p_\mu(x)dx = \overline{F}(x = 8mm)$

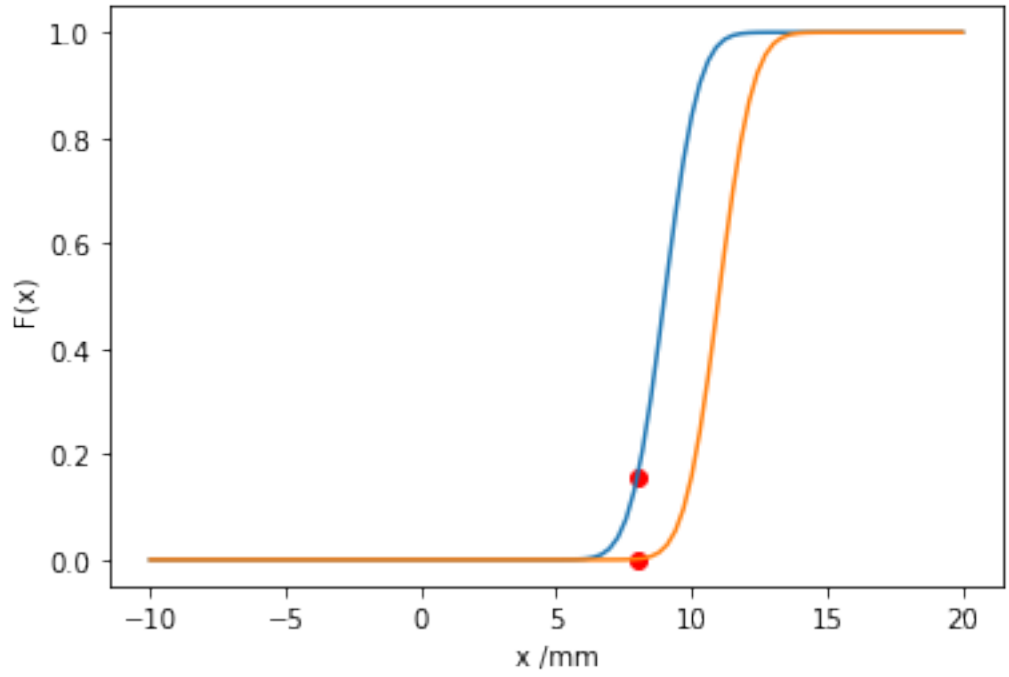
$P(\text{rainfall} < 8mm) > \min_{\mu \in [9,11]} \int_{-\infty}^{8mm} p_\mu(x)dx = \underline{F}(x = 8mm)$

Below, we show various probability density functions contained in the probability box. Then the upper and lower bounding CDFs are evaluated at the desired value. This works because the CDF increases monotonically in the mean parameter, and only one parameter is being varied. Such a normally distributed probability box, where only the mean is varied is sometimes known as a 'combed' probability box.

```
import matplotlib.pyplot as plt
import scipy.stats
import numpy as np
import scipy.optimize
import random

print("Upper probability: {}".format(scipy.stats.norm.cdf(8, loc=9, scale=1)))
print("Lower probability: {}".format(scipy.stats.norm.cdf(8, loc=11, scale=1)))

plot_x_range = np.linspace(-10, 20, 100)
plot_x_range_integrate = np.linspace(-10, 8, 100)
for _ in range(10):
    plt.plot(
        plot_x_range,
        scipy.stats.norm.pdf(plot_x_range, loc=9+2*random.random(), scale=1)
    )
plt.xlabel("x /mm")
plt.ylabel("p(x)")
plt.show()

plot_x_range = np.linspace(-10, 20, 100)
plt.plot(plot_x_range, scipy.stats.norm.cdf(plot_x_range, loc=9, scale=1))
plt.plot(plot_x_range, scipy.stats.norm.cdf(plot_x_range, loc=11, scale=1))
plt.scatter(8, scipy.stats.norm.cdf(8, loc=11, scale=1), color='red')
plt.scatter(8, scipy.stats.norm.cdf(8, loc=9, scale=1), color='red')
plt.xlabel("x /mm")
plt.ylabel("F(x)")
plt.show()
```

```
Upper probability: 0.15865525393145707
Lower probability: 0.0013499880316300933
```


0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
p(x)
−10
−5
0
5
10
15
20
x /mm


1.0
0.8
0.6
0.4
0.2
0.0
F(x)
−10
−5
0
5
10
15
20
x /mm

3.3.4 Worked example : Computation with an imprecise probabilistic model

Again, the amount of rainfall in mm on a particular day is given by the normal random variable $p_\mu(x) = \mathcal{N}(\mu, \sigma = 1), \mu \in [9, 11]$. It is known that the number of umbrellas sold in London on a particular day is given by $g(x) = 1000x^2$.

Estimate the number of umbrellas sold in London given our uncertain model of rainfall.

$$\text{Umbrellas sold} \in [\min_{\mu\in[9,11]} \int_{-\infty}^{\infty} p(x)1000x^2dx, \max_{\mu\in[9,11]} \int_{-\infty}^{\infty} p(x)1000x^2dx]$$

Analytically evaluating this integral in a popular computer algebra system yields Umbrellas sold $\in [82000, 122000]$.

```
# First try double loop Monte Carlo integration
n_samples = 1000


def mc_sim(mu: float):
    """
    Performs a Monte Carlo simulation for number of umbrellas sold.
    Args:
        mu: Mean of random variable
    """
    samples_x = scipy.stats.norm(loc=mu, scale=1).rvs(n_samples)

    def g(x): return 1000 * x ** 2

    return np.mean(g(samples_x)), np.std(g(samples_x)) / np.sqrt(n_samples-1)


min_mu = 9
max_mu = 11
mid_mu = 0.5 * (min_mu + max_mu)

min_mean_umbrellas = scipy.optimize.brute(
    lambda x: mc_sim(x)[0],
    ranges=((min_mu, max_mu),),
    finish=None,
    full_output=True
)

neg_max_mean_umbrellas = scipy.optimize.brute(
    lambda x: -mc_sim(x)[0],
    ranges=((min_mu, max_mu),),
    finish=None,
    full_output=True
)

print("The expected number of umbrellas sold falls in the interval: {}".format(
    [min_mean_umbrellas[1], -neg_max_mean_umbrellas[1]]))

error = mc_sim(mid_mu)[1]

print("The error of this Monte Carlo estimator is {}".format(error))
```

```
The expected number of umbrellas sold falls in the interval: [82026.72539211591,␣
↪122393.81237902433]
The error of this Monte Carlo estimator is 630.5853205364439
```

Good agreement is obtained with the analytic calculation. Note that the probabilistic calculation from *Worked example*

: Computation with a probabilistic model falls inside the stated bounds. When performing double loop Monte Carlo simulation as above one must take care to choose an outer loop optimiser which is robust to a stochastic objective function.

Notice in the above code example we use a `lambda x:` to define an *anonymous function* to access only the first output of the Monte Carlo simulation function for convenience.

3.3.5 Dempster-Shafer structures

A Dempster-Shafer Structure is another form of imprecise probability model which has been widely applied in engineering [30]. In this thesis we do not use Dempster-Shafer structures, however we briefly outline the model and the relationship to probability boxes here for context. A Dempster-Shafer (DS) structure represents the assignment of probability mass to intervals rather than point values, as is the case with probability density functions.

Consider the case where probability mass has been assigned to intervals on the real line, $\{([\underline{x}_{(1)}, \overline{x}_{(1)}], p_{(1)}), \ldots, ([\underline{x}_{(n)}, \overline{x}_{(n)}], p_{(n)})\}$, where $\sum_{i=1}^{n} p_{(i)} = 1$. We can define two measures called belief and plausibility which bound the probability mass contained in a particular interval. The belief measure is defined by

$$\text{bel}(A) = \sum_{\substack{i \\ [\underline{x}_{(i)}, \overline{x}_{(i)}] \subseteq A}} p_{(i)},$$

and the plausibility measure is defined by

$$\text{pls}(A) = \sum_{\substack{i \\ [\underline{x}_{(i)}, \overline{x}_{(i)}] \cap A \neq \emptyset}} p_{(i)},$$

where $\text{bel}(A) \leq P(A) \leq \text{pls}(A)$ and $\text{pls}(A) = 1 - \text{bel}(\overline{A})$, and $\subseteq$ represents a subset. For a particular DS structure, [19] define an associated probability box using

$$\overline{F}(x) = \sum_{\underline{x}_{(n)} \leq x} p_{(i)}$$

and

$$\underline{F}(x) = \sum_{\overline{x}_{(n)} < x} p_{(i)}.$$

3.4 Creating models in practice

3.4.1 Choosing a model

As discussed previously, it is essential to exercise engineering judgement when choosing an uncertainty model, both in terms of speed of computations which can be performed with the model and the appropriateness of the model's representation of uncertainty. Engineers should also note that it is essential to exercise their judgement even after the theoretical uncertainty framework is chosen, since the set of hypotheses included in the uncertainty model has a strong affect on the conclusions drawn from analysis.

For example, consider the following problem proposed by [3] two doctors examine a patient, but differ in their diagnoses. Doctor A believes the patient has a 99% chance of meningitis and 1% chance of concussion. Doctor B believes the patient has a 99% chance of tumor and 1% chance of concussion. Since the doctors' diagnoses strongly conflict with each other, a naïve application of Bayesian probability concludes that the patient most likely has concussion. Zadeh proposes that this problem can be solved with fuzzy logic. However, [4] shows that Bayesian probabilities can, in fact, be used to solve the problem, by allowing the model to consider that the doctors may have made a mistake in their estimations of probabilities.

Another interesting example is demonstrated by [5] and [6]; it is shown that using probability distributions to represent epistemic uncertainty in satellite conjunction analysis does not provide a useful description of the likelihood of collision between satellites, since the likelihood of collision appears to decrease when data with more incertitude is collected.

3.4.2 Training models from data

Here we provide a non-exhaustive review of methods to calibrate probabilistic and non-probabilistic generative uncertainty models, in order to set the context for the remainder of the thesis.

Creating parametric Bayesian probabilistic models

Consider the probability distribution $p_\theta(x) = p(x^{(i)}|\theta)$ with vector of parameters θ, which we wish to identify based on a set of n training samples, $\mathcal{X}_{\text{train}} = \{x^{(1)}, \dots, x^{(n)}\}$, drawn from the random variable specified by $p_\theta(x)$. A distribution over the parameters θ, given the data $\mathcal{X}_{\text{train}}$ can be obtained by applying Bayes' law:

$$P(\theta|\mathcal{X}_{\text{train}}) = \frac{P(\mathcal{X}_{\text{train}}|\theta)p(\theta)}{P(\mathcal{X}_{\text{train}})}, \tag{7}$$

where $p(\theta)$ represents a prior distribution on θ, $P(\mathcal{X}_{\text{train}}) = \int P(\mathcal{X}_{\text{train}}|\theta)d\theta$ acts as a normalising constant, and the data likelihood can be written as $P(\mathcal{X}_{\text{train}}|\theta) = \prod_i p(x^{(i)}|\theta)$ by assuming independence of training samples. This approach, known as Bayesian Hierarchical Modelling [7], has desirable properties. For example, the epistemic uncertainty on θ will decrease as more data becomes available which will be observed as a 'concentration' of the posterior distribution for θ around one point.

Although simple analytical distributions are often used for the likelihood $P(\mathcal{X}_{\text{train}}|\theta)$ (e.g. a Gaussian distribution with mean θ_1 and scale θ_2). One can also extend the framework to consider more complex likelihood functions. For example, often the likelihood is $P(\mathcal{X}_{\text{train}}|\theta) = \int P(\mathcal{X}_{\text{train}}|y)\delta(f(\theta) - y)dy$, where $f(\theta)$ is an arbitrarily expensive and complex function, for which we may not know the gradient, and $p(x^{(i)}|\theta)$ is a simple probability density, for example a normal distribution, and $\delta(x)$ is the Dirac delta function. This setting is referred to as an 'inverse problem' [8].

The probability distributions over θ represent epistemic uncertainty in θ, whilst the data likelihood, $p(x^{(i)}|\theta)$, represents the natural stochasticity (aleatory uncertainty) of the data generating mechanism. Note that the prior distribution, $p(\theta)$, should be chosen to represent our prior knowledge of the parameter θ, and in the case of no knowledge, should be set to an appropriate uninformative distribution. The distribution used for the uninformative prior should be chosen based on physical considerations regarding the parameter of interest, but is often somewhat arbitrarily assumed to be uniform [1].

The prior distribution is not the only place where prior knowledge enters into the probabilistic model; the model specification, i.e. the data likelihood, represents another form of prior knowledge which must be carefully considered with this approach [1]. It is particularly important to decide which parameters in the likelihood function should be modelled as uncertain, e.g. if the likelihood is assumed to be a Gaussian density, will a value be assumed for the standard deviation of the distribution, or will this be an element of θ, and hence an uncertain parameter?

We can derive point estimates for θ from the Bayesian approach [9]. The maximum a posteriori estimator for θ is obtained by evaluating $\theta_{\text{MAP}} = \max_\theta P(\theta|\mathcal{X}_{\text{train}})$, where $P(\theta|\mathcal{X}_{\text{train}}) \propto P(\mathcal{X}_{\text{train}}|\theta)P(\theta)$. The maximum likelihood estimator for θ is obtained by evaluating $\theta_{\text{ML}} = \max_\theta P(\mathcal{X}_{\text{train}}|\theta)$. Note that the maximum likelihood estimator is equivalent to the maximum a posteriori estimator when a uniform prior distribution is used. These estimators can be evaluated using any optimisation method. Stochastic Gradient Descent, a widely used optimisation method, will be discussed in *Machine Learning of Regression Models* since it is most often applied to regression models.

We do not necessarily have to disregard uncertainty in θ when using the maximum a posteriori approach, since the covariance of the distribution can be estimated by inverting the Hessian (matrix of second derivatives with respect to parameters θ) of $P(\theta|\mathcal{X}_{\text{train}})$. This is known as the Laplace approximation. This estimate is exact in some well known cases, e.g. the case of a Gaussian likelihood and prior, where the optimisation loss (the logarithm of the posterior) becomes the mean squared error [8].

In many cases the full posterior for θ can be calculated analytically, for example where a conjugate prior is used so that the posterior distribution has the same functional form as the prior distribution. If this is not the case, and one wishes to compute the full posterior distribution, then a Markov Chain Monte Carlo (MCMC) method is often used to obtain samples from the posterior distribution, or other approximate numerical techniques are used.

An MCMC algorithm constructs a Markov chain with the desired distribution as its equilibrium distribution, so that if the Markov chain is simulated for a sufficient time then the samples drawn are from the posterior distribution [7]. MCMC methods typically do not require the gradient of the posterior to be known, and are hence applicable to a wide class of problems. Unfortunately, MCMC simulation can be computationally infeasible when θ has high dimensionality, or when the training dataset is large. Recently, efficient sampling based algorithms have been proposed to combat this problem [10].

As an alternative to MCMC based methods, Variational Inference can be used to find the closest match between an approximating parametric 'proposal' distribution and the true posterior distribution. This method typically requires the gradient of the likelihood function to be known, but scales very well to high dimensionality problems [11].

Approximate Bayesian Computation is an efficient computational method which can be used to sample from an approximation of the posterior distribution in the case that the likelihood is too expensive to compute [12]. [13] demonstrate a similar method, where the true likelihood probability density function is replaced by an interval with associated probability, and show that bounds on the likelihood function can still be obtained in this case.

Worked example : elementary probability theory

We wish to measure the length of bolts in a box. It is known that all of the bolts in the box have the same length. Your colleague tells you that he thinks the bolts are about 1 metre long, and he is almost certain that the bolts are between 0.9 and 1.1 metres. Your ruler measures the bolts to the nearest 5 centimetres. You draw 10 bolts with lengths measuring $L = \{0.95m, 1m, 1m, 1m, 0.9m, 0.9m, 0.95m, 0.95m, 0.9m, 1m\}$.

This problem can be solved most easily by modelling your colleague's prior belief with a normal prior distribution: $P(l) = \mathcal{N}(\mu = 1, \sigma = 0.033)$, giving a 3σ (99%) confidence interval to fit the colleague's stated belief.

The likelihood function describes the precision of the ruler $P(L|l) = \prod_i P(l_i|l) = \prod_i \mathcal{N}(\mu, \sigma = 0.05)$, where the first equality holds because the measurements are made independently.

Then the posterior is obtained easily since the Gaussian prior is conjugate to the Gaussian likelihood: $P(l|L) \propto P(L|l)P(l)$

The equations to analytically calculate the posterior hyperparameters are given here.

Using the sample mean and standard deviation:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{1}{10}(0.95 + 1 + 1 + 1 + 0.9 + 0.9 + 0.95 + 0.95 + 0.9 + 1) = 0.955m$$

$$\sigma = 0.0437798m$$

The posterior hyperparameters are:

$$\sigma_0^{2\prime} = \frac{1}{\frac{n}{\sigma^2} + \frac{1}{\sigma_0^2}} = 0.0128^2 m^2$$

$$\mu_0' = \frac{\frac{n\bar{x}}{\sigma^2} + \frac{\mu_0}{\sigma_0^2}}{\frac{n}{\sigma^2} + \frac{1}{\sigma_0^2}} = 0.962m$$

So you can be slightly more confident than your colleague about the length of the bolts in the box. Also, you believe the bolts are slightly shorter.

Helpfully, this type of computation allows you to simulate how much data is required to obtain a certain level of confidence in the posterior distribution, by drawing more data from the posterior predictive distribution and performing inference with this simulated data. This can also be achieved by studying analytic formulae to predict what is known as the posterior concentration rate.

Choosing a prior and likelihood function with support on the whole real line is probably a poor choice for a variable like length, which must be positive. How could the model be improved?

Frequentist confidence intervals

In this thesis, traditional frequentist statistics are not used, except for in the validation of some Interval Predictor Models, but for the interested reader we briefly describe here how a frequentist confidence interval can be obtained for θ.

In frequentist statistics, one aims to identify a region of parameter space which would contain the true value of the parameter with a specified frequency if the experiment was repeated, i.e. we aim to find the confidence interval $\Theta = [\underline{\theta}, \overline{\theta}]$, where $P(\theta \in \Theta) = 1 - \alpha$, and α is an arbitrarily small probability.

Non-parametric prediction intervals

The maximum and minimum of a dataset (i.e. $[\min_i x_{(i)}, \max_i x_{(i)}]$ when x is one dimensional) can be used to produce a prediction interval with coverage probability $\frac{n-1}{n+1}$, i.e. the probability that $x_{(n+1)}$ will fall inside the prediction interval [14]. A tighter prediction interval, with a lower coverage probability of $\frac{n+1-2j}{n+1}$ can be obtained by using the j-th smallest and largest values in the dataset.

Creating parametric imprecise probability models

The application of Bayes' law in (7) assumes that the data $\mathcal{X}_{\text{train}}$ consists of real, 'crisp', values. However, we can also apply Bayes' law to imprecise, interval data. For example, consider the set of training data $\mathcal{X}_{\text{train}} = \{[\underline{x}^{(1)}, \overline{x}^{(1)}], \ldots, [\underline{x}^{(n)}, \overline{x}^{(n)}]\}$. If an analytic equation is available for the posterior parameters then in many cases it is possible to obtain bounds on the posterior parameters given interval data. For example, if a Gaussian density is used for the likelihood and prior, then one may obtain bounds on the posterior normal distribution parameters analytically [15].

The standard Bayesian paradigm can also be made robust by considering a set of prior distributions. This is known as Robust Bayes [16]. Again, bounds on the posterior parameters are available analytically in many cases, e.g. the Imprecise Dirichlet model.

Probability boxes can also be obtained by creating so-called confidence structures, which are encoded as probability boxes. Confidence boxes encode confidence intervals at all confidence levels. The binomial confidence bounds, which bound the success probability of a binomial random variable, are a particularly useful example which can be found by inverting the CDF of a binomial random variable [17].

Creating non-parametric imprecise probability models

Several methods exist to obtain non-parametric CDFs from data. The CDF can be estimated from n training samples, $\mathcal{X}_{\text{train}} = \{x^{(1)}, \ldots, x^{(n)}\}$, using the empirical cumulative distribution function (eCDF), which is given by

$$S_n(x) = \frac{1}{n} \sum_{i}^{n} \mathbb{I}_{x \geq x^{(i)}}(x),$$

where $\mathbb{I}$ is the indicator function, which is equal to 1 if the subscript statement is satisfied, and is otherwise equal to zero [18]. The eCDF is effectively the random variable which is formed by assigning probability density equally at the point value of each sample, and hence when plotted the eCDF looks like a staircase function. The eCDF can be generalised to the case of imprecise sampled data, by considering a separate eCDF for the upper and lower bounds of the samples. These upper and lower bounds represent the envelope of a probability box, and hence an empirical probability box is obtained [19].

So-called concentration inequalities can be used to obtain bounds on the CDF of a random variable with a certain confidence. A probability box can be obtained for the random variable by choosing a cutoff confidence, such that the CDFs at that confidence will form the envelope of the probability box [19].

The Kolmogorov-Smirnov statistic can be used to measure the confidence that the true CDF of a random variable differs by more than a certain probability from the eCDF obtained from sampled data (i.e. the vertical distance between the CDFs is compared) [20]. The Kolmogorov-Smirnov statistic is given by

$$D = \sup_x |S_n(x) - F(x)|,$$

where $F(x)$ is the true CDF and the values for D can be obtained from [18]. Now a set of CDFs can be found with associated confidence. The Kolmogorov-Smirnov statistic can be applied to eCDF bounds obtained by considering imprecise data [19].

Chebyshev's inequality bounds the probability density of a random variable which can fall more than a certain number of standard deviations from the mean [21]. Therefore knowledge of the mean and standard deviation of a random variable imposes bounds on its CDF. Chebyshev's inequality is given by

$$P((X - \mu) \geq k\sigma) \leq \frac{1}{k^2},$$

where $k > 1$ is a real number, μ is the mean of a random variable, and σ is the standard deviation of the random variable.

3.4.3 Creating uncertainty models without data

When insufficient data is available to create a satisfactory uncertainty model using the techniques described in the previous section, one may resort to creating a model based on the opinions of experts. This process is known as expert elicitation [22]. In this section we briefly outline how various uncertainty models can be obtained from expert opinion, in order to further justify and provide context for the uncertainty models used in this thesis. Expert elicitation is not the main focus of this thesis, and therefore this section may be skipped without consequence.

Probabilistic elicitation

In the Bayesian Hierarchical Modelling paradigm, discussed in the previous section, the posterior distribution concentrates as data is received and gradually the prior has less influence on the posterior. The prior distribution represents the state of knowledge about a parameter before data is available, and if limited data is available then more care should be taken to choose an appropriate prior, i.e. the opinions of experts should be considered and assessed quantitatively.

When eliciting multiple expert opinions one must attempt to aggregate the opinions of experts, regardless of the model chosen. Usually the opinions of experts are fused using quantitative rules [22], and feedback may be given to the experts in order to allow their opinions to be changed. [23] propose the SHELF framework which gives specific rules for how the opinions of experts should be elicited, and proposes that the opinions should be aggregated by a rational unbiased observer during the elicitation process.

Non-probabilistic elicitation

Imprecise probability models also have a role to play in expert elicitation. For example, if the model exhibits severe dependency on a probabilistic prior which can only be elicited approximately, one may wish to conduct a sensitivity analysis to the prior by considering a probability box prior, as discussed in *Creating parametric imprecise probability models*.

Other non-probabilistic models may be considered for a parameter, for example interval bounds on a parameter may be available from physical considerations. Alternatively, experts may prefer to specify their estimates as intervals, or may feel more comfortable specifying bounding CDFs (i.e. probability boxes). [19] discuss several methods for aggregating probability boxes which can be chosen based on the desired properties of the analysis.

3.4.4 Validating a trained model

Once an uncertainty model has been obtained it is essential that the model is validated, to ensure that it suitably represents the analysts uncertainty. Even if one uses a Bayesian framework and trusts the priors and probability calculus, it is still possible that the model has been misspecified. For this reason, one should qualitatively inspect the results of the analysis, and quantitatively check that a probabilistic model is correctly calibrated using numerical techniques.

Validating probabilistic models

Usually one partitions the data available for creating the model into the data set used for training the model, $\mathcal{X}_{\text{train}}$, and the data set used for testing, $\mathcal{X}_{\text{test}}$ (containing N_{test} data points).

If a probabilistic model is correctly calibrated then we expect the stated probabilities to represent the real frequencies with which events occur, for example if a set of events are predicted to occur with 0.9 probability then we expect that they occur 90% of the time in reality. This can be verified by plotting the test data relative to the trained distribution. One method of achieving this is 'binning' the data into a histogram, and visually comparing the histogram to the plotted distribution for the trained model [7].

One may use the test set, $\mathcal{X}_{\text{test}}$, to compute various statistics of the model. For example, classical statistical tests can be used, such as the χ^2 summary statistic for the sum of squared errors which represents goodness of fit [7]. The test set can be used to compute the negative logarithmic predictive density for the model, i.e. $-\log P(\mathcal{X}_{\text{test}}|\text{Model}) = -\log \mathbb{E}_{P(\theta|\mathcal{X}_{\text{train}})} P(\mathcal{X}_{\text{test}}|\theta)$, which can be used as a figure of merit for comparing models.

If one wishes to compare two probabilistic models Model_1 and Model_2 then one may compare the evidence for the models by computing the Bayes Factor:

$$\frac{P(\mathcal{X}_{\text{train}}|\text{Model}_1)}{P(\mathcal{X}_{\text{train}}|\text{Model}_2)} = \frac{P(\text{Model}_1|\mathcal{X}_{\text{train}})P(\text{Model}_2)}{P(\text{Model}_2|\mathcal{X}_{\text{train}})P(\text{Model}_1)}, \tag{8}$$

where the model evidence (or marginal likelihood) $P(\mathcal{X}_{\text{train}}|\text{Model})$ is computed by evaluating the expectation of the data likelihood for the model over the posterior obtained in training ($P(\mathcal{X}_{\text{train}}|\text{Model}) = \mathbb{E}_{P(\theta|\mathcal{X}_{\text{train}})} P(\mathcal{X}_{\text{train}}|\theta) = \int P(\mathcal{X}_{\text{train}}|\theta)P(\theta|\mathcal{X}_{\text{train}})d\theta$). If the Bayes factor is greater than one then Model_1 is preferred, otherwise one should choose Model_2 [24]. When $P(\text{Model}_2) = P(\text{Model}_1)$ the prior belief in each model is equal and the Bayes factor becomes equal to the likelihood ratio. The likelihood ratio can also be computed for the test data set. One can also generate new data by sampling from the distribution $\mathbb{E}_{P(\theta|\mathcal{X}_{\text{train}})} p(x|\theta)$ and comparing this to the training and test data. This is known as a posterior predictive check [7].

Validating non-probabilistic models

If a convex-set or interval based model is compared to crisp data then one can check that all elements of the test set $\mathcal{X}_{\text{test}}$ fall within the model. [25] proposes that interval models are validated against interval data using the metric for comparison of two sets A and B:

$$\Delta(A, B) = \inf_{a \in A, b \in B} |a - b|,$$

where A would represent the trained convex model, and B would represent an element of $\mathcal{X}_{\text{test}}$. The mean of $\Delta(A, B)$ over every element $B \in \mathcal{X}_{\text{test}}$ could be used to validate against the entire test set, i.e. $\frac{1}{N_{\text{test}}} \sum_{B \in \mathcal{X}_{\text{test}}} \Delta(A, B)$.

Following this, [25] proposes a generalisation of the Wasserstein distance to measure the distance between an eCDF and a probability box, as a probability box validation metric. The proposed metric, termed mean absolute difference of deviates, is given by

$$\mathbb{E}_x \Delta([\underline{F}(x), \overline{F}(x)], [\underline{S_n}(x), \overline{S_n}(x)]),$$

where $[\underline{F}(x), \overline{F}(x)]$ are the bounds of the probability box to be validated, and $[\underline{S_n}(x), \overline{S_n}(x)]$ are the bounds of the empirical probability box created from the training data, which becomes a single CDF in the case of crisp data ($[S_n(x), S_n(x)]$).

The metric reduces to zero when there is overlap of the probability boxes at every x. We compare models by computing the metric for each model, and then choosing the model with the lowest value for the metric.

3.5 Chapter summary

This chapter presents a review of uncertainty models which can be used to describe unknown parameters in a computational model, in addition to describing how the models can be created in practice, from data or otherwise, and how computation can be performed with these models. In particular, we discussed probabilistic models, which are used for conventional uncertainty quantification, and non-probabilistic models, which are used in cases where only limited or imprecise data may be available, and prior knowledge may be difficult to obtain in the form of a probabilistic prior. In this chapter, all considered uncertainty models did not depend upon the behaviour of other variables — their uncertainty was constant or homoscedastic. In many cases, it may be desirable to model how the uncertainty in a variable changes with respect to another variable. This is typical, for example, when we wish to consider how the output of a computer code changes with respect to its inputs. Therefore, the following chapter describes so called regression models, where modelling this dependency is possible.

Part III

Machine Learning of Regression Models

CHAPTER

FOUR

OVERVIEW OF REGRESSION MODELS

Regression models differ from generative uncertainty models in that they model the effect of one variable on another variable. In the language of probability theory this involves modelling a conditional probability distribution, rather than a joint probability distribution. In some fields regression models are known as discriminative models, but this is usually when the dependent variable in the probability distribution is discrete, and the problem to be solved involves classification [54]. This thesis is concentrated only on the case of continuous variables. In this chapter, we review different classes of regression model and describe how they can be trained and validated from data. We describe in which circumstances each type of model should be used.

4.1 Parametric regression models

Typically when learning a regression model, one wishes to obtain a estimate of the relationship between the two variables, in addition to a measure of uncertainty in this relationship. Typically this is achieved by specifying a function relating the two variables, and then using Machine Learning techniques to calibrate parameters of that function based on sampled data. The function can be specified based on expert knowledge, or alternatively a very general function is chosen.

The most simple regression models usually consist of the inner product of a parameter vector and a vector of 'features', such that the model output's dependency on the parameters to be calibrated is linear. The features, usually known as the basis, are a function of the input variables, which may be non-linear. Models of this form are amenable to computation, and in general a wide variety of functions can be expressed in this way. In this formulation the output of the model is given by $f(x, \boldsymbol{p}) = \sum_i p_i \phi_i(x)$, where x and $f(x, \boldsymbol{p})$ represent the input and output to the regression model, respectively, p_i are the parameters of the model to be calibrated which are components of the vector $\boldsymbol{p}$, and $\phi_i(x)$ is the basis [9]. Let x be a vector in $\mathbb{R}^N$, with components x_i.

The basis should be chosen based on prior knowledge and engineering judgement. Basis functions are either global or local. A local basis consists of radial functions, which only depend upon the distance from a certain point, i.e. $\phi(x) = \phi(|x|)$. The Gaussian basis is a common radial basis function, where $\phi_i(x) = \exp\left(-\epsilon_i (x - c_i)^2\right)$. ϵ_i and c_i are parameters which can be set based on knowledge of the physical process being modelled or learnt from data at the same time as p_i, though this is more difficult because $f(x)$ has non-linear dependency on these parameters. The Gaussian basis function tends to zero far away from c_i. On the other hand, a global basis consists of functions which in general are non-zero at all points in the input space. For example, the global polynomial basis of the form $\phi_i(x) = \prod_i x_i^{l_i}$, with indices l_i chosen based on engineering judgement, is nonzero everywhere except for $x = 0$.

Polynomial Chaos Expansions offer a principled way to choose basis functions. Polynomial Chaos Expansions are a class of regression model with a basis consisting of polynomials which are orthogonal to each other, i.e. their inner product is zero with respect to a probability distribution over their inputs ($\int_{\mathbb{R}^N} \phi_i(x)\phi_j(x)p(x)dx = 0 \forall i \neq j$) [85].

In order to learn more complex functions, a more complex function representation is needed, with the ability to model arbitrary non-linearities in the model parameters to be learned. Neural networks are a widely used regression model which fulfil this purpose [9]. Neural networks consist of neurons which apply an inner product between the input and a parameter vector, and then apply an arbitrary non-linearity, before feeding into other neurons, until finally the result is outputted. These computational neurons are organised into layers, which is equivalent to multiplying the input by a

parameter matrix (known as a weight matrix), rather than a vector. The way in which layers are connected can lead to desirable properties, for example spatial invariance of particular layers over the input when the input is an image, i.e. a matrix. This is equivalent to repeating parameters in the weight matrix. The most simple way to connect the layers is to allow each layer to be completely connected to the subsequent layer, which is known as the feedforward architecture. Layer i of a feed-forward neural network is given by

$$f_i(x, W) = \text{act}\left(W_i f_{i-1}(x)\right), \tag{1}$$

where $f_i(x, W)$ is a vector (which is the input vector x when $i = 0$, i.e. $f_0(x) = 0$), act is a non-linear activation function, and W_i is the ith weight matrix. The activation function is typically the hyperbolic tangent function, the soft-max function or the rectified linear function. A diagram of a feed-forward neural network is shown in Fig. 1. [86] demonstrates how neural networks can be trained to replicate the opinions of expert engineers on the probability of failure of particular pipe welds in a power plant.

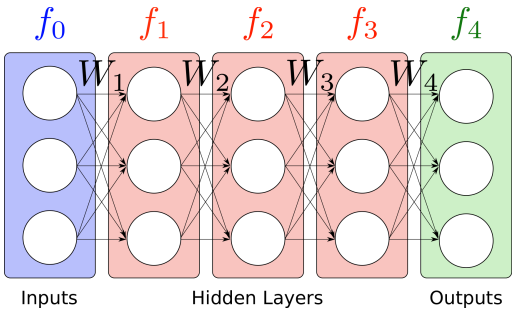


Fig. 1: A diagram of a feed-forward neural network with three hidden layers, each with a width of three neurons. The activation function, which is applied to the weighted sum of the inputs to each neuron, is not shown.

4.1.1 Bayesian parameter learning

Defining a data likelihood

It is common to define a probability distribution based on the output of a regression model, and then use the probability calculus introduced in Chapter *Models of Uncertainty* to learn distributions over the parameters of the regression model. For simple models it is usually assumed that the output $f(x)$ of the model is some meaningful parameter of the distribution, e.g. the mean of a normal distribution: $p(y|x, \boldsymbol{p}) = \mathcal{N}(f(x, \boldsymbol{p}), \sigma^2)$, where σ is the scale parameter of the normal distribution, which should be learned form data. This results in a model where the level of uncertainty in $p(y|x)$ does not depend on the input to the model. This is known as a homoscedastic model of uncertainty.

Sometimes it is desirable to explicitly allow the uncertainty in the predictions to depend on x. This is known as heteroscedastic uncertainty [87]. In this case it can be useful to define a model where other parameters of the distribution depend on x, i.e. we define $p(y|x, \boldsymbol{p}) = \mathcal{N}(f_1(x, \boldsymbol{p}_1), f_2(x, \boldsymbol{p}_2)^2)$, where $f_1(x, \boldsymbol{p}_1)$ and $f_2(x, \boldsymbol{p}_2)$ are different functions with different parameter sets, $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$.

Any valid probability distribution can be used in a similar way, for example the Dirac delta function can be used to define $p(y|x, W, u) = \delta(y - f_i(x, u, W))$, where $f_i(x)$ is the output of a neural network, where in this case the input layer is a function of the true input and a random vector of noise, i.e. $f_0(x, u, W) = \text{concatenate}(x, u)$ where $u \sim \mathcal{U}(0, 1)$. This is a very popular formulation in Machine Learning for Computer Vision [88] [89], because $p(y|x, W, u)$ can now be used to learn a very general probability density in a computationally tractable way, since $p(y|x, W) = \int \delta(y - f_i(x, u, W))\mathcal{U}(0, 1)du$ can be evaluated easily using a Monte Carlo estimator during inference of the posterior distribution.

Performing the Bayesian computation

Using a set of n training samples, $\mathcal{X}_{\text{train}} = \{\{x^{(1)}, y^{(1)}\}, ..., \{x^{(n)}, y^{(n)}\}\}$, one can learn a distribution over $\boldsymbol{p}$ in the same way as in Chapter *Models of Uncertainty* by using

$$P(\boldsymbol{p}|\mathcal{X}_{\text{train}}) = \frac{P(\mathcal{X}_{\text{train}}|\boldsymbol{p})p(\boldsymbol{p})}{P(\mathcal{X}_{\text{train}})} = \frac{\prod_i P(x^{(i)}, y^{(i)}|\boldsymbol{p})p(\boldsymbol{p})}{P(\mathcal{X}_{\text{train}})} = \frac{\prod_i p(y^{(i)}|x^{(i)}, \boldsymbol{p})p(x^{(i)})p(\boldsymbol{p})}{P(\mathcal{X}_{\text{train}})}, \tag{2}$$

where the data likelihood can be written as $P(\mathcal{X}_{\text{train}}|\boldsymbol{p}) = \prod_i p(y^{(i)}, x^{(i)}|\boldsymbol{p})$ by assuming independence of training samples, $p(\boldsymbol{p})$ represents a prior distribution on $\boldsymbol{p}$, $p(x^{(i)}, \boldsymbol{p}) = p(x^{(i)})p(\boldsymbol{p})$ by assuming independence of $\boldsymbol{p}$ and the sampled inputs, and $P(\mathcal{X}_{\text{train}}) = \int \prod_i p(y^{(i)}|x^{(i)}, \boldsymbol{p})p(x^{(i)}, \boldsymbol{p})d\boldsymbol{p}$ acts as a normalising constant. As in Chapter *Models of Uncertainty*, the posterior distribution on $\boldsymbol{p}$ will tend to concentrate around one point as more data is received.

The maximum likelihood and maximum a posteriori estimators described in Chapter *Models of Uncertainty* are equally applicable here. These estimators are evaluated by minimising so-called loss functions (objective functions). The maximum a posteriori estimator for $\boldsymbol{p}$ is obtained by evaluating $\boldsymbol{p}_{\text{MAP}} = \max_{\boldsymbol{p}} P(\boldsymbol{p}|\mathcal{X}_{\text{train}})$, where $P(\boldsymbol{p}|\mathcal{X}_{\text{train}}) \propto P(\mathcal{X}_{\text{train}}|\boldsymbol{p})P(\boldsymbol{p}) = \mathcal{L}_{\text{MAP}}(\boldsymbol{p})$. The maximum likelihood estimator for $\boldsymbol{p}$ is obtained by evaluating $\boldsymbol{p}_{\text{ML}} = \max_{\boldsymbol{p}} \mathcal{L}_{\text{ML}}(\boldsymbol{p}) = \max_{\boldsymbol{p}} P(\mathcal{X}_{\text{train}}|\boldsymbol{p})$. The maximum likelihood estimator is equivalent to the maximum a posteriori estimator when a uniform prior distribution is used. Using a normal distribution for the data likelihood leads to the well known mean squared error or ℓ_2 norm loss function when the maximum likelihood estimator is used. Using a polynomial basis with the mean squared error loss function leads to ordinary least squares regression. If a normal distribution prior is used then this leads to an ℓ_2 weight regularisation (squared penalty) in the maximum a posteriori loss function. In a similar way, most sensible loss functions which aim to estimate point values for parameters have a Bayesian interpretation.

Computational methods

In practice, the most common way to create regression models is to evaluate the estimators $\boldsymbol{p}_{\text{ML}}$ or $\boldsymbol{p}_{\text{MAP}}$ with Stochastic Gradient Ascent, which maximises the logarithm of the relevant probability distribution (or equivalently by using Stochastic Gradient Descent to minimise the negative of the log posterior). This is computationally tractable even for high dimensional $\boldsymbol{p}$, since usually the gradient of $\log P(\boldsymbol{p}|\mathcal{X}_{\text{train}})$ is known analytically. Gradient Descent methods are a class of optimisation methods which adjust the value for a parameter at each step of the algorithm by subtracting a small learning rate constant, η, multiplied by the gradient of the loss, $\mathcal{L}$, with respect to the trainable parameter, i.e. $p_i \leftarrow p_i + \eta \frac{\partial \mathcal{L}}{\partial p_i}$. This is repeated for a set number of iterations until the algorithm has converged. Stochastic Gradient Descent approximates the product of likelihoods in the loss function by evaluating the likelihood for one different sampled data point at each iteration. This is effective since the expectation of the loss used in Stochastic Gradient Descent will still be equal to the true value of the loss function. In this case, the learning rate constant must be reduced to ensure convergence, which means many iterations of the algorithm are required to ensure a good estimate for the parameters is obtained. Mini-batches, where the likelihood is evaluated for a small set of data points at each iteration, can be used to achieve good convergence at higher learning rates, whilst decreasing the required computational time, since a GPU can be used [90]. Various improvements to Stochastic Gradient Descent aim to ensure that the optimiser reaches a true minimum of the loss function, a particularly common improvement being the ADAM optimiser [91].

Using the maximum likelihood and maximum a posteriori estimators can allow some estimate of the uncertainty in the model to be made, but this uncertainty is an underestimate of the true model uncertainty. For very simple regression models, MCMC can be used to obtain the full posterior distribution on $\boldsymbol{p}$, however this is usually intractable for models with large parameter sets. As an alternative, variational inference can be used to minimise the difference between the $P(\boldsymbol{p}|\mathcal{X}_{\text{train}})$ and an approximating posterior distribution, as described in Chapter *Models of Uncertainty*. Note that $P(\boldsymbol{p}|\mathcal{X}_{\text{train}})$ must be differentiable in $\boldsymbol{p}$ for this to be possible. Using Bayes' law to infer posterior distributions over the weights of a neural network is referred to as training a Bayesian neural network, and this is almost always achieved by using variational inference [11].

The technique of dropout sampling has been shown to improve the performance of Stochastic Gradient Descent solvers, by improving the performance on validation tests [92]. Dropout sampling involves randomly setting a fraction, p_{dropout} of the weights to zero during each training iteration. For particular choices of activation function, when an l_2 penalty

on the weights is used in the loss function, it can be shown that dropout sampling is equivalent to variational inference on a Bayesian neural network, where a Bernoulli distribution is used as the approximating posterior distribution [93]. In order for the approximating posterior distribution to be an accurate representation of the true posterior distribution, it is necessary to adjust the dropout probability, p_{dropout}. Rather than repeatedly performing training with different dropout probabilities, it is more efficient to make the dropout probability a parameter which can be optimised during training, by making the loss differentiable in terms of the dropout probability. This can be achieved with concrete dropout, where a continuous approximation of the Bernoulli distribution is used [94].

4.1.2 Validation

As was the case with generative uncertainty models in Chapter *Models of Uncertainty*, it is necessary to validate Regression Models. For probabilistic regression models this involves many of the same techniques which are applied when validating generative models. However, validating conditional probability densities presents additional challenges; although the model's predicted probabilities may be correctly calibrated on average, the model may be overly certain in some areas of the input domain and too uncertain in other areas. For example, using a regression model with homoscedastic uncertainty on a dataset where the uncertainty is heteroscedastic may predict the correct mean squared error on average [95], but the model evidence will be lower than for a more appropriate model.

To briefly recap the content from *Validating a trained model*, before training one should split the data into training and test data sets, and then begin the validation by applying sanity checks. For example, a posterior predictive check could be used, where data is sampled from the trained model and compared to the training data. Alternatively, one could produce a plot of the normalised residuals, where the difference between the model output and the training and test data divided by predicted standard deviation ($\frac{y-y^{(i)}}{\sqrt{\text{Var}_{p(y|x^{(i)})}(y)}}$) is plotted against the model output. Then more formal methods can be used, for example the Bayes factor can be computed as in (8), to compare several models [9]. This is similar to comparing the negative logarithmic predictive density of different models on the test sets, which is equal to the Mahalanobis distance for Gaussian predicted probability densities. It is essential to compare the value of the loss (the negative logarithmic predictive density) between the test and training data sets. If the value of the loss is much higher on the test data set it is likely that the model is over-fitting the data, and will not generalise well to new data. One may also wish to compute the expected variance of the Model's predictions ($\int \text{Var}_{p(y|x)}(y)p(x)dx$), as it is likely that this can be compared to the expected uncertainty of a subject matter expect in order to appraise the performance of the model. If the uncertainty is too high it is likely that the model is under-fitting the data so the model complexity should be increased.

4.1.3 The bias-variance tradeoff

Bayesian techniques rely upon priors to control the complexity of a regression model. Well chosen priors prevent learning too much information from a sample of data, and hence prevent overfitting by restricting the effective learning capacity of the model. The issue of underfitting is usually addressed by giving the model as much complexity as is possible and necessary to reduce the bias of the model, in order to ensure that the model is able to represent the desired function in principle. Then, overfitting is prevented by using a well chosen prior to reduce the variance of the fitted model. Non-Bayesian machine learning techniques often arbitrarily introduce mechanisms to constrain model complexity such as weight penalties; these techniques are unnecessary in the Bayesian paradigm due to the effect of prior distributions, which are in some cases equivalent to weight penalties. Bootstrapping (averaging over maximum likelihood models trained on re-sampled selections of the training data) is often used to reduce the variance of the trained model. [9] describes how bootstrapping can also be seen as a method to compute maximum likelihood estimates of difficult to compute quantities like the standard error in an estimator, and an alternative implementation of maximum a posteriori estimation for the case of an uninformative prior. For certain likelihood functions and priors the bootstrap distribution can be seen as an approximate Bayesian posterior distribution.

The Vapnik-Chervonenkis (VC) dimension, which represents the complexity of a classification model, can be used to derive a bound between the test error and training error of a classification model (i.e. where y is a binary outcome) [96]. Similar bounds exist for regression models. This means that the test error can be established without partitioning the data. We do not use the VC dimension in this thesis because it is difficult to calculate in practice, but we return to the idea

of calculating a bound on the test error of a model without partitioning the data in *Validating models with the scenario approach*.

4.1.4 Worked example : linear regression

We wish to measure the relationship between the rainfall in London on a particular day and the number of umbrellas sold on that day.

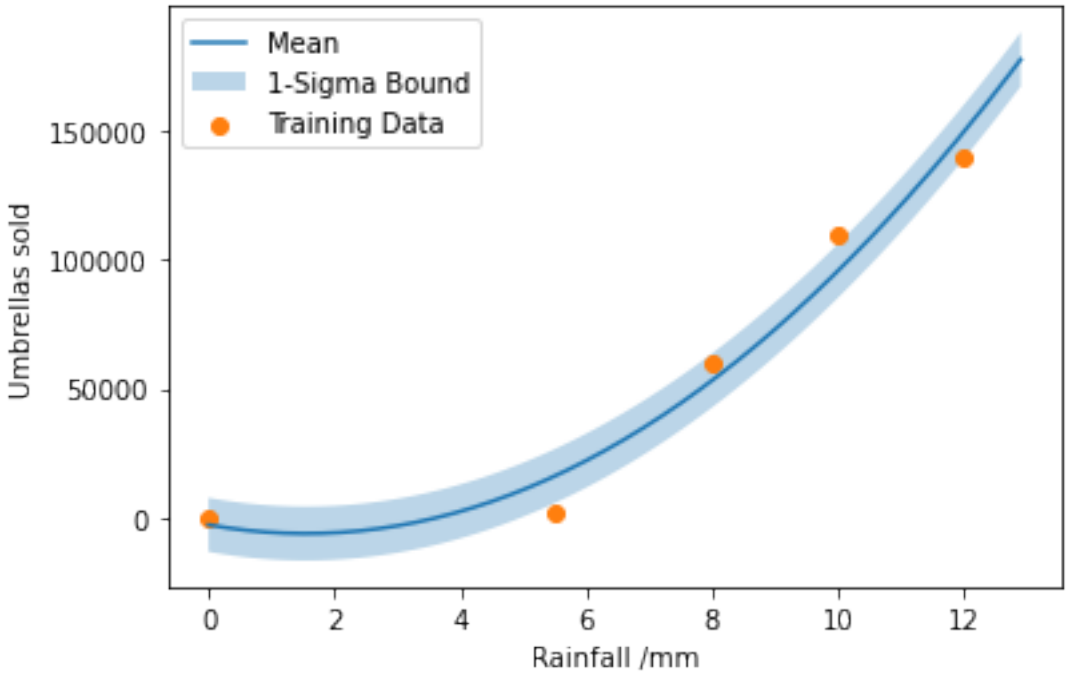
The daya collected from 5 randomly selected days is shown below:

Day	1	2	3	4	5
Rainfall (mm)	0	5.5	10	12	8
Umbrellas sold	50	20000	110000	140000	60000

We choose to fit a quadratic function, because we believe that the relationship between the variables is approximately quadratic for reasonable values of rainfall, and assume that the measurement noise is Gaussian. Using the notation in this Chapter, this corresponds to the model $f(x, \boldsymbol{p}) = \sum_{i=0}^{i=2} p_i x^i$, i.e. $\phi_i(x) = x^i$ and likelihood $p(y|x, \boldsymbol{p}) = \mathcal{N}(f(x, \boldsymbol{p}), \sigma^2)$.

The most simple method of solving this problem involves using least squares estimation in sklearn.

```
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures

import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize

rainfall = np.array([0, 5.5, 10, 12, 8]).reshape(-1, 1)
umbrellas = np.array([0, 2000, 110000, 140000, 60000])

model = make_pipeline(PolynomialFeatures(2), LinearRegression())
model.fit(rainfall, umbrellas)

residuals = model.predict(rainfall) - umbrellas
sigma = np.std(residuals)

x_plot = np.arange(0, 13, 0.1).reshape(-1, 1)

plt.plot(x_plot, model.predict(x_plot), label="Mean")
plt.fill_between(np.arange(0, 13, 0.1), model.predict(x_plot) - sigma,
                 model.predict(x_plot) + sigma,
                 alpha=0.3, label="1-Sigma Bound")
plt.scatter(rainfall, umbrellas, label="Training Data")
plt.xlabel("Rainfall /mm")
plt.ylabel("Umbrellas sold")
plt.legend()
plt.show()
```



The regressed quadratic regression model with Gaussian distributed errors is shown by the shaded region.

We may wish to add some sort of prior distribution for the model parameters, i.e. $p_0 \sim \mathcal{N}(0, \sigma = 0.1)$ and $p_1 \sim \mathcal{N}(0, \sigma = 0.1)$, so the constant and linear terms should make minimal contribution to the model.

To find the maximum a posteriori values for the model parameters we can minimise the loss function: $\mathcal{L} = \frac{(y - f(x, \boldsymbol{p}))^2}{2\sigma^2} - \sigma\sqrt{2\pi} + 0.5p_1^2 + 0.5p_2^2$

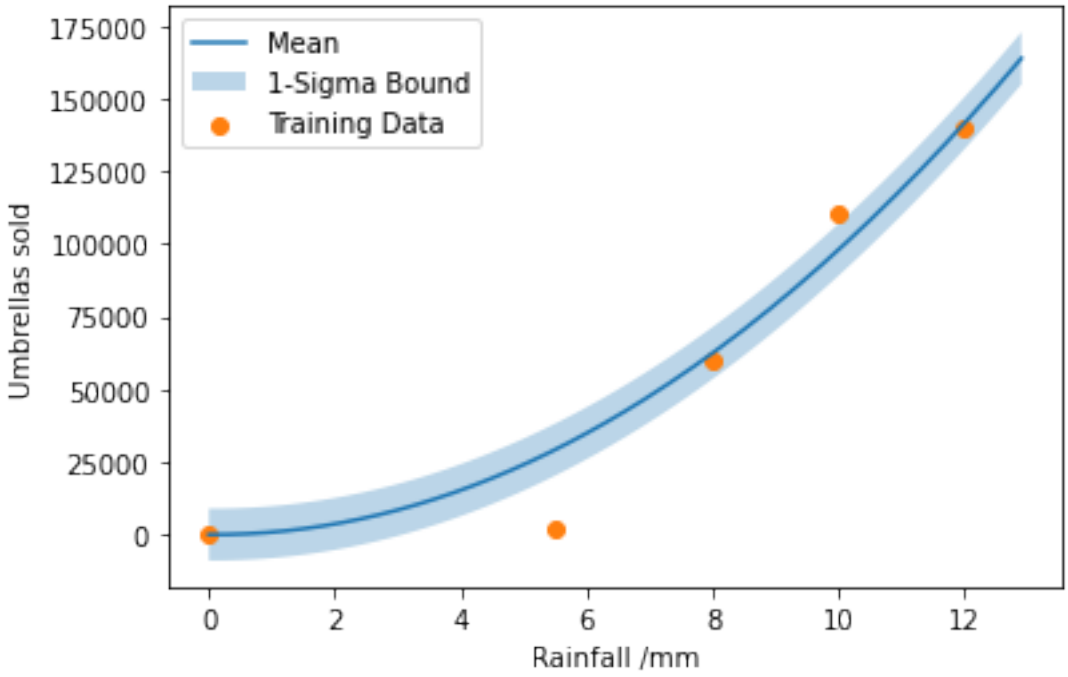
Since the data points are independent we sum the loss for each data point. We rescale the data in order to improve the performance of the optimiser.

```
scale = 10000
umbrellas_rescaled = umbrellas / scale


def model(x: np.ndarray, p0: float, p1: float, p2: float) -> np.ndarray:
    """
    Quadratic Model

    Args:
        x: Input variable
        p0: Parameter
        p1: Parameter
        p2: Parameter
    """
    return p0 + p1 * x + p2 * x ** 2


def prior(p0: float, p1: float, p2: float) -> float:
    """
    Equal up to a constant to the log gaussian prior on p0 and p1
    """
    return 0.5 * p0 ** 2 / 0.01 + 0.5 * p1 ** 2 / 0.01
```

(continues on next page)

(continued from previous page)

```
def loss(x: np.ndarray, y: np.ndarray, p0: float, p1: float, p2: float,
         sigma: float) -> np.ndarray:
    """
    Per datapoint loss
    """
    return ((model(x, p0, p1, p2) - y) ** 2 / (2 * sigma ** 2)
            + sigma * np.sqrt(2 * np.pi) + prior(p0, p1, p2))


def total_loss(p0, p1, p2, sigma):
    """Sum loss for all datapoints"""
    return np.sum([loss(x, y, p0, p1, p2, sigma)
                   for x, y in zip(rainfall, umbrellas_rescaled)])


result = scipy.optimize.minimize(lambda x: total_loss(*x), x0=[0, 0, 0, 1])


def model_predict(x):
    return scale * model(x, result.x[0], result.x[1], result.x[2])


sigma = scale * result.x[3]

x_plot = np.arange(0, 13, 0.1)

plt.plot(x_plot, model_predict(x_plot), label="Mean")
plt.fill_between(np.arange(0, 13, 0.1), model_predict(x_plot) - sigma,
                 model_predict(x_plot) + sigma,
                 alpha=0.3, label="1-Sigma Bound")
plt.scatter(rainfall, umbrellas, label="Training Data")
plt.xlabel("Rainfall /mm")
plt.ylabel("Umbrellas sold")
plt.legend()
plt.show()
```



Of course, fitting such a model with gradient descent in a library such as `pytorch` would be more efficient.

Why might this model be a poor choice? Consider the large rainfall limits and support of the likelihood function. What might be a better model?

4.2 Non-parametric models

Bayesian non-parametric models are models for regression which learn a prior distribution over functions at training time. This is achieved by learning so-called 'Kernel hyper-parameters' from the training data set, which specify a Gaussian Process Prior. Inference is performed at test time by applying Bayes' law to this prior with the available data [79]. The process of performing computation inference at test time is referred to as lazy learning (as opposed to eager learning presented in the previous section, where the inference happens during training, and only a single pass through the network is required to make predictions). Gaussian Process models can be tuned to have desirable properties for many applications, for example one can assume that the training data is noise-free and hence the relevant function can be learned from very few samples. Gaussian Processes are particularly useful for global optimisation of non-linear functions, because the predictive uncertainty can be used to decide where the next sample should be chosen [80]. Gaussian processes are not used in this thesis (except for as a comparison in some of the numerical examples), but we will briefly describe how they relate to the Bayesian parametric models discussed in this Chapter.

It can be shown that a single layer Bayesian neural network, with an infinite number of neurons in the layer, is equivalent to a Gaussian Process, and can therefore be used as a more convenient alternative, since the inference is performed at training time [81]. Variational approximations can be made for Gaussian Processes to make the inference computationally tractable for large data sets and deep architectures [82]. Neural Processes learn a distribution over functions in a similar way to Gaussian processes, but with significantly reduced computational expense since only a forward pass through the neural network is required at test time [83]. Neural Processes are useful for meta-learning (learning to learn). For example, they were used to learn how to predict 2D views of 3D spaces, given limited training data [84].

4.3 Learning bounds on a model

Instead of learning a probability distribution to describe the effect of one variable on another, one may instead attempt to learn a function which maps the input variables to an interval representing the possible range of the output. Such models are known as interval predictor models. Sometimes the predicted intervals have an associated confidence level (or a bound on the confidence level), and as such they can be considered as bounds on the quantiles of a random variable [55]. Typically the obtained intervals represent an outer approximation, i.e. the intervals are overly wide and hence conservative in an engineering sense. An interval predictor model can be seen as prescribing the support of a Random Predictor Model, which is defined as a function which maps input variables to an output random variable. A Gaussian Process Model is a specific case of a Random Predictor Model.

In this section, we describe how interval predictor models can be trained in practice. We then describe how the theory of scenario optimisation can be used to provide guaranteed bounds on the reliability of the trained interval predictor models, for the purpose of validation.

4.3.1 Training interval predictor models

Let us consider a black box model (sometimes referred to as the Data Generating Mechanism or DGM) which acts on a vector of input variables $x \in \mathbb{R}^{n_x}$ to produce an output $y \in \mathbb{R}$. We wish to obtain the two functions $\overline{y}(x)$ and $\underline{y}(x)$ which enclose a fraction, ϵ, of samples from the DGM, i.e. samples of $y(x)$ where x is sampled from some unknown probability density. The functions $\overline{y}(x)$ and $\underline{y}(x)$ are bounds on a prediction interval, and as such we wish them to be as tight as possible. This can be written as a so-called chance constrained optimisation program: $\text{argmin}_{\boldsymbol{p}} \{\mathbb{E}_x(\overline{y}_{\boldsymbol{p}}(x) - \underline{y}_{\boldsymbol{p}}(x)) : P\{\overline{y}_{\boldsymbol{p}}(x) > y(x) > \underline{y}_{\boldsymbol{p}}(x)\} \leq \epsilon\}$, where $\boldsymbol{p}$ is a vector of function parameters to be identified, and ϵ is a parameter which constrains how often the constraints may be violated. Chance constrained optimisation programs can be solved by using a so-called scenario program, where the chance constraint is replaced with multiple sampled constraints based on data, i.e.

$$\text{argmin}_{\boldsymbol{p}} \{\mathbb{E}_x(\overline{y}_{\boldsymbol{p}}(x) - \underline{y}_{\boldsymbol{p}}(x)) : \overline{y}_{\boldsymbol{p}}(x^{(i)}) > y^{(i)} > \underline{y}_{\boldsymbol{p}}(x^{(i)}), i = 1, ..., N\}, \tag{3}$$

where $\mathcal{X}_{\text{train}} = \{\{x^{(1)}, y^{(1)}\}, ..., \{x^{(n)}, y^{(n)}\}\}$ are sampled from the DGM. Most of the literature on scenario optimisation Theory aims to obtain bounds on ϵ. Finding bounds on ϵ using scenario optimisation is easier in practice than other similar methods in statistical learning theory, since no knowledge of the Vapnik-Chervonenkis dimension (a measure of the capacity of the model, which is difficult to determine exactly) is required.

A key advantage over other machine learning techniques is that interval training data (i.e. where the training data inputs are given in the form $x^{(i)} \in [\underline{x}^{(i)}, \overline{x}^{(i)}]$ due to epistemic uncertainty or some other reason) fits into the scenario optimisation framework coherently [56]. This can be seen as equivalent to defending against the attack model of adversarial examples considered by [57], where the network is trained to produce the same outputs for small perturbations of the input data. The framework also permits robustness against uncertainty in training outputs, i.e. $y^{(i)} \in [\underline{y}^{(i)}, \overline{y}^{(i)}]$, where $y^{(i)}$ is a single training example output.

Convex interval predictor models

If the objective and constraints for the scenario program are convex then the program can be easily solved, and bounds can be put on ϵ. We will approximate the DGM with an interval predictor model (IPM) which returns an interval for each vector $x \in X$, the set of inputs, given by

$$I_y(x, P) = \{y = G(x, \boldsymbol{p}), \boldsymbol{p} \in P\}, \tag{4}$$

where G is an arbitrary function and $\boldsymbol{p}$ is a parameter vector. By making an approximation for G and considering a linear parameter dependency (4) becomes $I_y(x, P) = \{y = \boldsymbol{p}^T \phi(x), \boldsymbol{p} \in P\}$, where $\phi(x)$ is a basis (polynomial and radial bases are commonly used), and $\boldsymbol{p}$ is a member of a convex parameter set. The convex parameter set is usually assumed to be either ellipsoidal or hyper-rectangular [58]. [59] demonstrates that hyper-rectangular parameters sets result in an

IPM with bounds with a convenient analytical form. The hyper-rectangular parameter uncertainty set can be defined as $P = \{\boldsymbol{p} : \underline{\boldsymbol{p}} \leq \boldsymbol{p} \leq \overline{\boldsymbol{p}}\}$, where $\underline{\boldsymbol{p}}$ and $\overline{\boldsymbol{p}}$ are parameter vectors specifying the defining vertices of the hyper rectangular uncertainty set. The IPM with linear parameter dependency on the hyper-rectangular uncertain set of parameters is defined by the interval $I_y(x, P) = [\underline{y}(x, \overline{\boldsymbol{p}}, \underline{\boldsymbol{p}}), \overline{y}(x, \overline{\boldsymbol{p}}, \underline{\boldsymbol{p}})]$, where $\underline{y}$ and $\overline{y}$ are the lower and upper bounds of the IPM, respectively. Explicitly, the lower bound is given by $\underline{y}(x, \overline{\boldsymbol{p}}, \underline{\boldsymbol{p}}) = \overline{\boldsymbol{p}}^T \left(\frac{\phi(x)-|\phi(x)|}{2}\right) + \underline{\boldsymbol{p}}^T \left(\frac{\phi(x)+|\phi(x)|}{2}\right)$, and the upper bound is given by $\overline{y}(x, \overline{\boldsymbol{p}}, \underline{\boldsymbol{p}}) = \overline{\boldsymbol{p}}^T \left(\frac{\phi(x)+|\phi(x)|}{2}\right) + \underline{\boldsymbol{p}}^T \left(\frac{\phi(x)-|\phi(x)|}{2}\right)$. To identify the hyper-rectangular uncertainty set one trains the IPM by minimising the value of $\delta_y(x, \overline{\boldsymbol{p}}, \underline{\boldsymbol{p}}) = (\overline{\boldsymbol{p}} - \underline{\boldsymbol{p}})^T |\phi(x)|$, subject to the constraint that the training data points fall inside the bounds on the IPM, by solving the linear and convex optimisation problem

$$\{\hat{\underline{\boldsymbol{p}}}, \hat{\overline{\boldsymbol{p}}}\} = \text{argmin}_{\boldsymbol{u},\boldsymbol{v}} \{\mathbb{E}_x[\delta_y(x, \boldsymbol{v}, \boldsymbol{u})] : \underline{y}(x^{(i)}, \boldsymbol{v}, \boldsymbol{u}) \leq y^{(i)} \leq \overline{y}(x^{(i)}, \boldsymbol{v}, \boldsymbol{u}), \boldsymbol{u} \leq \boldsymbol{v}\}. \tag{5}$$

The constraints ensure that all data points to be fitted lie within the bounds and that the upper bound is greater than the lower bound. This combination of objective function and constraints is linear and convex [59]. In this thesis all interval predictor models have polynomial bases, i.e. $\phi(x) = \left[1, x^{i_2}, x^{i_3}, ...\right]$ with $x = [x_a, x_b, ...]$ and $i_j = [i_{j,a}, i_{j,b}, ...]$ with $i_j \neq i_k$ for $j \neq k$.

For illustrative purposes an example degree 2 IPM is shown without training data points in Fig. 2. The hyper rectangular uncertainty set corresponding to the IPM in Fig. 2 is plotted in Fig. 3. The discontinuity observed in the upper and lower bounds is a consequence of the chosen basis, and can be avoided by choosing a basis where $\phi(x) = |\phi(x)|$.

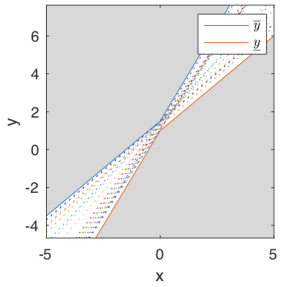


Fig. 2: The IPM's hyper rectangular uncertainty set plotted in 'parameter space'. The uniformly sampled parameter vectors of the polynomials shown in Fig. 3.

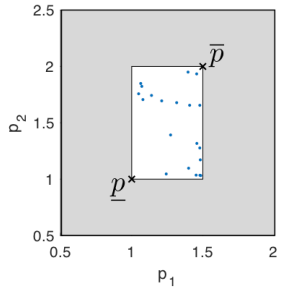


Fig. 3: The IPM's hyper rectangular uncertainty set plotted in 'parameter space'. The uniformly sampled parameter vectors of the polynomials shown in Fig. 2 are displayed as points in the uncertain set.

Non-convex interval predictor models

In some circumstances engineers may wish to represent more complex functions with IPMs, and hence the functions used to represent the bounds of the IPM may have more trainable parameters. The interior point method used to solve linear optimisation programs, such as those used for convex IPMs, has complexity $d^2 n_{\text{cons}}$, where d represents the number of optimisation variables and n_{cons} represents the number of constraints in the optimisation (which in this case scales linearly with the number of training data points), and hence the method does not scale well to IPMs with large numbers of trainable parameters [45].

Neural networks enable uncertainty models to be created with vast numbers of parameters in a feasible computational time. Neural networks with interval outputs were proposed by [60], and further described by [61]. In these papers the learning takes place by identifying the weights W, which solve the following program:

$$\text{argmin}_{\overline{W},\underline{W}} \left[\mathbb{E}_x(\overline{y}(x) - \underline{y}(x)) : \overline{y}(x^{(i)}) > y^{(i)} > \underline{y}(x^{(i)}) \; \forall \; i\right], \tag{6}$$

where $\overline{y}(x)$ and $\underline{y}(x)$ are obtained from two independent neural networks, such that $\overline{y}(x)$ and $\underline{y}(x)$ are the output layers of networks, such as those defined by (1). In practice this problem is solved by using a mean squared error loss function with a simple penalty function to model the constraints. In general, penalty methods require careful choice of hyper-parameters to guarantee convergence. These neural networks act in a similar way to interval predictor models, however the interval neural networks do not attempt to use the training data set to bound ϵ. [62] define similar networks with fuzzy parameters to operate on fuzzy data. The fuzzy neural networks are trained by minimising a least square loss function (a set inclusion constraint is not used), which can also be applied to time series data sets. These approaches are very different from the approach of [63], where a traditional neural network loss function is intervalised using interval arithmetic.

[64] extended the scenario approach to non-convex optimisation programs, and hence applied the approach to a single layer neural network, with a constant width interval prediction, which was trained using the interior-point algorithm in Matlab. In other words the following program is solved:

$$\text{argmin}_{W,h} \left[h : |y^{(i)} - \hat{y}(x^{(i)})| < h \; \forall \; i\right], \tag{7}$$

where h is a real number, and $\hat{y}$ represents the central line of the prediction obtained from the same network specified by (1). The bounds on the prediction interval are therefore given by $\overline{y}(x) = \hat{y}(x) + h$ and $\underline{y}(x) = \hat{y}(x) - h$. The constant width interval neural network expresses homoscedastic uncertainty. The solution to the optimisation program in (7) can also be obtained by finding the neural network weights which minimise the so-called maximum-error loss:

$$\mathcal{L}_{\text{max-error}} = \max_i |y^{(i)} - \hat{y}(x^{(i)})|, \equiv \lim_{p \to \infty} \left(\sum_i |y^{(i)} - \hat{y}(x^{(i)})|^p \right)^{\frac{1}{p}}, \tag{8}$$

where h is the minimum value of the loss. It is trivial to show this is true, since the set inclusion constraint in (7) requires that h is larger than the absolute error for each data point in the training set [65].

In [66] a back propagation algorithm for Neural Networks with interval predictions is proposed. A modified maximum-error loss is used. The approach can accommodate incertitude in the training data. A key result is that, by using mini-batches, the complexity of the proposed approach does not directly depend upon the number of training data points as with other Interval Predictor Model methods.

4.3.2 Validating models with the scenario approach

We will first present an overview of the scenario optimisation theory for the validation of models in the convex case, before describing more general techniques which apply in the non-convex case.

Convex case

Intuition tells us that the solution of the scenario program will be most accurate when the dimensionality of the design variable is low and we take as many samples of the constraints as possible (in fact, an infinite number of sampled constraints would allow us to reliably estimate $P\{\overline{y}_{\boldsymbol{p}}(x) > y(x) > \underline{y}_{\boldsymbol{p}}(x)\}$, and hence solve the program exactly). However, in practice obtaining these samples is often an expensive process. Fortunately, the theory of scenario optimisation provides robust bounds on the robustness of the obtained solution. The bounds generally take the following form: $P^n(V(\hat{z}_n) > \epsilon) \leq \beta$. This equation states that the probability of observing a bad set of data (i.e. a bad set of constraints) in future, such that our solution violates a proportion greater than ϵ of the constraints (i.e. $V(\hat{z}_n) > \epsilon$ where $V(\hat{z}_n) = \frac{1}{n}\sum_i^n V^{(i)}$ and $V^{(i)} = 1$ only if $\overline{y}_{\boldsymbol{p}}(x) > y(x) > \underline{y}_{\boldsymbol{p}}(x)$), is no greater than β. The scenario approach gives a simple analytic form for the connection between ϵ and β in the case that the optimisation program is convex:

$$\beta = \frac{1}{\epsilon}\frac{d}{n+1}, \tag{9}$$

where n is the number of constraint samples in the training data set used to solve the scenario program, and d is the dimensionality of the design variable, z. For a fixed d and n we obtain a confidence-reliability relationship: by decreasing ϵ slightly, $1-\beta$ can be made to be insignificantly small. Other tighter bounds exist in the more recent scenario optimisation literature, e.g. [67] [68] [69], for example

$$\beta = \sum_{i=0}^{d-1}\binom{n}{i}\epsilon^i(1-\epsilon)^{n-i}. \tag{10}$$

Crucially the assessment of $V(\hat{z}_n)$ is possible a priori, although other techniques exist [70]. [71] analyse the reliability of solutions of the maximum error loss functions ((8)) in the scenario framework when $\hat{y}(x)$ is convex in x and the function weights.

In the convex case, the a priori assessment is made possible by the fact that the number of support constraints (the number of constraints which if removed result in a more optimal solution) for a convex program is always bounded by the dimensionality of the design variable. [72] explore this connection for convex programs in further detail, by analysing the number of support constraints after a solution is obtained. In fact, the bound in (10) if often overly conservative, because in many cases the number of support constraints is less than the dimensionality of the design variable, and hence a more accurate bound on the reliability of the IPM can be obtained. The improved bound is given by letting ϵ be a function of the number of support constraints s_n^* such that $\epsilon(s_n^*) = 1 - t(s_n^*)$. Then for $0 < \beta < 1$ and $0 < s_n^* < d$ the equation

$$\frac{\beta}{n+1}\sum_{m=k}^{n}\binom{m}{k}t^{m-k} - \binom{n}{k}t^{n-k} = 0 \tag{11}$$

has one solution, $t(k)$ in the interval $[0, 1]$.

This idea has a deep connection with the concept of regularisation in machine learning [73]. [74] demonstrates how the number of support constraints of a scenario program can be used to iteratively increase the number of sampled constraints, which requires fewer sampled constraints in total than (10) for equivalent ϵ and β. For a non-convex program, the number of support constraints is not necessarily less than the dimensionality of the design variable, and therefore a new approach is required, which we describe in the following section.

Non-convex case

[75] provide the following bound for the non-convex case: $P^n(V(\hat{z}_n) > \epsilon(s)) < \beta$, where

$$\epsilon(s) = \begin{cases} 1, & \text{for } s = n, \\ 1 - \sqrt[n-s]{\frac{\beta}{n\binom{n}{s}}}, & \text{otherwise}, \end{cases} \tag{12}$$

and s is the cardinality of the support set (in other words, the number of support constraints). The behaviour of this bound is similar to the convex case since in general increasing n should increase the size of the support set.

Finding the cardinality of the support set is in general a computationally expensive task, since the scenario program must be solved n times. [64] present a time-efficient algorithm which only requires that the scenario problem is solved s times.

A posteriori frequentist analysis

When data is available in abundance, as is typically the case in most machine learning tasks where a neural network is currently used, $V(\hat{z}_n)$ can be evaluated more easily by using a test set to collect samples from $V(\hat{z}_n)$. Estimating $V(\hat{z}_n)$ is similar to estimating a probability of failure in the well known reliability theory. Therefore one can construct a Monte Carlo estimator of $V(\hat{z}_n)$, or use more advanced techniques from reliability analysis if it is possible to interact with the data generating mechanism. For example, if the number of test data points is large we can use the normal approximation Monte Carlo estimator of $V(\hat{z}_n)$ with $V(\hat{z}_n) \approx \frac{N_v}{N_t}$ and standard deviation $\sqrt{\frac{\frac{N_v}{N_t}(1-\frac{N_v}{N_t})}{N_t}}$, on a test set of size N_t, where N_v data points fall outside the interval bounds of the neural network.

A particularly robust method of estimating the probability of a binary outcome involves using the binomial confidence bounds. In this case specifically, one can bound $V(\hat{z}_n)$ with the desired confidence using the binomial confidence bounds: $\sum_{i=0}^{N_t-N_v} \binom{N_t}{i}(1-\underline{v})^i \underline{v}^{N_t-i} = \frac{\beta}{2}$ and $\sum_{i=N_t-N_v}^{N_t} \binom{N_t}{i}(1-\overline{v})^i \overline{v}^{N_t-i} = \frac{\beta}{2}$, where $P(V(\hat{z}_n) < \overline{v} \cap V(\hat{z}_n) > \underline{v}) = \beta$. Estimating $V(\hat{z}_n)$ using a test set also offers the advantage that when the neural network is used for predictions on a different data set, $V(\hat{z}_n)$ can be evaluated easily. If the value of $V(\hat{z}_n)$ obtained on the test set is higher than that on the training dataset, one can apply regularisation in order to implicitly reduce the size of the support set and increase $V(\hat{z}_n)$ on the test set (e.g. dropout regularisation, or ℓ_2 regularisation on the weights).

This methodology is ideal for models with a complex training scheme, where determining the support set would be prohibitively expensive. Note that the probabilistic assessment of the reliability of the model takes place separately from the training of the regression model, such that it is still robust, even if there is a problem with the regression model training. This is an important advantage over Variational Inference methods which are often used with neural networks.

4.3.3 Software for interval predictor models

[76] describe the first open source software implementation of interval predictor models in the generalised uncertainty quantification software OpenCossan, which is written in Matlab. The OpenCossan software allows convex IPMs to be trained, with hyper-rectangular uncertainty sets. The OpenCossan software is modular and allows the IPMs to be automatically trained as approximations of expensive engineering models, and then used in other engineering calculations, e.g. design optimisation. A partial Python port of the OpenCossan IPM code was released as open source software by [77].

The introduced software has been applied in [78], to study fatigue damage estimation of offshore wind turbines jacket substructure.

4.3.4 Worked example : linear regression

We repeat the previous linear regression example using the `PyIPM` Interval Predictor Model library. Since the IPM makes fewer assumptions than the parametric model we will require more data to learn an effective model. Therefore we will generate training data from a toy function $1000x^2 + \epsilon$, where ϵ is randomly distributed noise with standard deviation 20000.

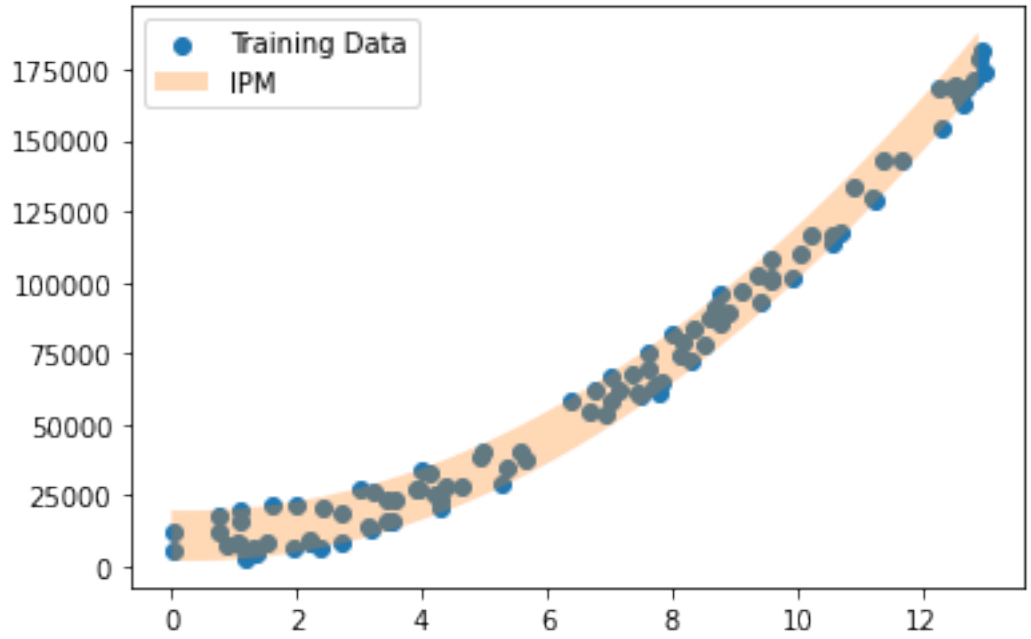
```
import PyIPM
import numpy as np
import matplotlib.pyplot as plt

rainfall = 13 * np.random.rand(100).reshape(-1, 1)
umbrellas = 1000 * rainfall.squeeze(-1) ** 2 + 20000 * np.random.rand(100)
```

(continues on next page)

(continued from previous page)

```
model = PyIPM.IPM(polynomial_degree=2)

model.fit(rainfall, umbrellas)

x_plot = np.arange(0, 13, 0.1)
upper_bound, lower_bound = model.predict(x_plot.reshape(-1, 1))

plt.scatter(rainfall, umbrellas, label="Training Data")
plt.fill_between(
    x_plot,
    lower_bound.squeeze(-1),
    upper_bound.squeeze(-1),
    alpha=0.3,
    label="IPM"
)
plt.legend()
plt.show()

print("For a test set generated from the same function as the training data, "
      "our model predictions will be enclose the test set with probability "
      "greater than {}".format(model.get_model_reliability()))
```

```
     pcost       dcost       gap    pres   dres   k/t
 0:  2.5650e-11 -4.0101e-11  2e+06  1e-01  1e+02  1e+00
 1:  1.9689e+04  2.1652e+04  5e+05  2e-02  2e+01  2e+03
 2:  1.9919e+04  2.0073e+04  5e+04  3e-03  3e+00  2e+02
 3:  1.9305e+04  1.9404e+04  2e+04  1e-03  8e-01  1e+02
 4:  1.8544e+04  1.8567e+04  4e+03  2e-04  2e-01  2e+01
 5:  1.8385e+04  1.8389e+04  7e+02  4e-05  4e-02  4e+00
 6:  1.8355e+04  1.8355e+04  5e+01  3e-06  3e-03  3e-01
 7:  1.8354e+04  1.8354e+04  6e+00  4e-07  3e-04  3e-02
 8:  1.8354e+04  1.8354e+04  6e-02  4e-09  3e-06  3e-04
 9:  1.8354e+04  1.8354e+04  6e-04  4e-11  3e-08  3e-06
Optimal solution found.
```




```
For a test set generated from the same function as the training data, our model␣
↪predictions will be enclose the test set with probability greater than 0.
↪7704137287382764
```

This example demonstrates the benefits of Interval Predictor Models: few assumptions are required, but a model which fits the data well can be obtained, alongside rigorous uncertainty quantification.

4.4 Chapter summary

This chapter presents a review of regression models with predictive uncertainty which can be used to describe the relationship between variables in engineering models, in addition to describing how the models can be created in practice from data. We reviewed probabilistic models and non-probabilistic models. Probabilistic regression models use probability distributions to express information about the variability and uncertainty in the modelled output; they are currently the most widely used regression models. Non-probabilistic models are useful in cases where only limited or imprecise data may be available, and prior knowledge of regression model parameters may be difficult to obtain. A particular advantage of Convex IPMs are the a priori bounds on the model bound violation, which can be used to validate the model at training time without test data.

Part IV

Reliability Analysis

The purpose of reliability analysis is to study the performance and safe operation of a system subject to uncertainty. In this chapter we review probabilistic, non-probabilistic and imprecise probabilistic notions of reliability analysis.

CHAPTER

FIVE

RELIABILITY ANALYSIS WITH RANDOM VARIABLES

5.1 Problem definition

5.1.1 Reliability theory

The aim of structural reliability analysis is to compute the probability that the performance of a system is less than some specified threshold; this probability is known as the failure probability of the system. Firstly, the performance of the system, $g(\boldsymbol{x})$, is defined as a function of the vector of system variables, $\boldsymbol{x} = (x_1, x_2, \ldots, x_i, \ldots)$. The performance function is negative when the system fails, and otherwise positive. Then the failure probability can be found by solving the integral

$$P_f = P(g(\boldsymbol{x}) < 0) = \int \mathbb{I}_f(\boldsymbol{x}) f_X(\boldsymbol{x}) d\boldsymbol{x}, \tag{1}$$

where the indicator function, $\mathbb{I}_f(\boldsymbol{x})$, is defined as $\mathbb{I}_f(\boldsymbol{x}) = \left\{ \begin{array}{ll} 1, & \text{for } g(\boldsymbol{x}) < 0 \\ 0, & \text{for } g(\boldsymbol{x}) \geq 0 \end{array} \right\}$, and the probability density function of the system random variables is $f_X(\boldsymbol{x})$ [106]. It is common for the performance function to be defined in the load resistance form, e.g.

$$g(\boldsymbol{x}) = \sum_{i \in R} x_i - \sum_{i \in L} x_i, \tag{2}$$

where R are indices corresponding to resistance factors and L are indices corresponding to load factors, so that the system fails when the sum of loads is greater than the sum of resistances [107]. When the resistance and load are balanced, $g(\boldsymbol{x}) = 0$ and the system is on the interface of the safety and failure regions. The limit state surface is specified by the $\boldsymbol{x}$ for which $g(\boldsymbol{x}) = 0$.

In fault tree analysis the failure event of a system is written in terms of failure events for smaller subsystems or components, using Boolean algebra. Probability arithmetic can be used with the fault tree to combine failure probabilities for individual subsystems to obtain the failure probability for the whole system. This requires knowledge of the dependencies between the probability of failure events for the considered sub-systems [106].

5.1.2 Reliability based design optimisation

In reliability based design optimisation (RBDO), a cost function, e.g. the weight or construction cost of the system, is minimised subject to the constraint that the failure probability of the system does not fall below a certain value. The reliability based design optimisation problem can be stated as the optimisation program

$$\text{argmin} \left\{ \text{cost}(\boldsymbol{d}) : P_f(\boldsymbol{d}) < P_{\text{target}} \right\}, \tag{3}$$

where $\boldsymbol{d}$ is the vector of design variables, $\text{cost}(\boldsymbol{d})$ is the cost function of the design, $P_f(\boldsymbol{d})$ is the failure probability of the design, and P_{target} is the target failure probability. Usually the vector of design variables, $\boldsymbol{d}$, will be parameters of

the random variables, $f_X(\boldsymbol{x})$, associated with the resistance, such that (3) usually finds a balance between a cost effective design and a design where the resistance of the system is sufficiently greater than the load [106].

In engineering practice the model of the structure is often computationally expensive to evaluate, and therefore it may be more convenient to find a sub-optimal solution to (3), by designing the structure based on engineering judgement. Approximate rules of thumb, such as partial safety factors, allow the reliability of the system to be constrained approximately using analytical equations [106]. Then the full reliability analysis can be performed with the proposed design to ensure that the reliability of the system satisfies the constraints in (3). Hence a safe and efficient design can be obtained with reduced computational effort.

5.1.3 Sensitivity analysis

Sensitivity analysis allows the effect of each uncertain variable on the variability of the model response to be quantified. This can be achieved either by local methods, which describe variability of the model response at the expected value of the system variables, or global methods, which describe the total variability of the model response.

Local sensitivity analysis is often achieved by evaluating the first derivative of the model response with respect to the system variables. Similarly, [108] defines component importance as the partial derivative of the system reliability with respect to the reliability of the component.

Global sensitivity analysis is usually performed by evaluating the Sobol indices,

$$S_i = \frac{\text{Var}_{x_i}(\mathbb{E}_{x_{\sim i}}(g(\boldsymbol{x})))}{Var(g(\boldsymbol{x}))}, \tag{4}$$

which describe the contribution of the variance of x_i to the total variance of the model response $g(\boldsymbol{x})$, when only x_i is varied. $x_{\sim i}$ represents all random variables other than x_i. The Total Effect indices, given by

$$T_i = 1 - \frac{\text{Var}_{x_{\sim i}}(\mathbb{E}_{x_i}(g(\boldsymbol{x})))}{Var(g(\boldsymbol{x}))}, \tag{5}$$

include the effect of interactions caused by varying x_i whilst varying other variables [109].

The most simple way to evaluate the Sobol indices and total effect indices is by using a Monte Carlo estimator for the expectation and variance terms in (4) and (5). In some cases, for example when the model has too many parameters or the model is very computationally expensive, it is necessary to use a more complex method to compute the Sobol and total sensitivity indices. For example, the upper bound of the total sensitivity index can be efficiently calculated by integrating the local sensitivity analysis over the whole space of the inputs [110], and the Sobol indices can be efficiently calculated by use of the Fourier Amplitude Sensitivity Testing (FAST) method [111].

5.2 Methods to compute the failure probability

In some circumstances the failure probability can be computed analytically, for example when the system variables $\boldsymbol{x}$ are normally distributed and the performance function is linear [106]. However, often a closed form solution of (1) is not available and hence alternative methods must be used.

5.2.1 Monte Carlo simulation

In general, the failure probability can be computed by Monte Carlo simulation, as discussed for general functions in *Models of Uncertainty*. The Monte Carlo estimator for the failure probability is

$$\hat{P}_f = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}_f(\boldsymbol{x}^{(i)}), \tag{6}$$

where N samples, $\boldsymbol{x}^{(i)}$, are drawn from the the probability density function of the system random variables $f_X(\boldsymbol{x})$. The coefficient of variation of the failure probability estimator is $\text{CoV}[\hat{P}_f] = \sqrt{\frac{1-P_f}{NP_f}}$. Therefore, obtaining order of magnitude estimates of P_f with Monte Carlo simulation requires at least $\frac{1}{P_f}$ samples, and for an accurate estimate even more samples are required. If P_f is small and $g(\boldsymbol{x})$ is expensive to evaluate then the number of samples required is unreasonably large, and more efficient strategies are required to evaluate P_f.

5.2.2 Efficient sampling strategies

Several sampling strategies have been proposed to choose a set of samples which can be used to reduce the variance of the Monte Carlo estimator in (6) without expending additional computational effort. Low-discrepancy sampling strategies aim to choose a set of samples which cover the sampling domain with the desired density. This is often not the case with a small random set of samples, which may fall disproportionately in one area of the sampling domain before the law of large numbers takes effect. Stratified Sampling strategies, such as Latin Hypercube Sampling divide the probability density of the system variables into an n-dimensional grid, where each grid element contains equal probability density. Then a sample may be chosen at random in each grid element, resulting in a set of samples which covers the sampling domain well. For linear functions it can be shown that this sampling strategy has a lower variance than the traditional Monte Carlo estimator [112]. The main disadvantage of efficient sampling strategies is that the reduction in the coefficient of variation is small when compared to other techniques, e.g. Line Sampling.

5.2.3 First Order Reliability Method

The First Order Reliability Method (FORM) allows the probability of failure of a system to be computed without Monte Carlo simulation. Assuming the system variables are distributed normally and independently, the probability of failure can be obtained analytically if the performance function, $g(\boldsymbol{x})$, is linear. If the performance function is not linear, a Taylor expansion can be used to find a linear approximation of the limit state function as shown in Fig. 1. If the system random variables are not normally distributed then a transformation must first be applied to the random variables and the limit state function, so that FORM can be applied [106].

The performance function, $g(\boldsymbol{x})$, is written as the Taylor series expansion

$$g(\boldsymbol{x}) = g(\boldsymbol{x}^*) + (\boldsymbol{x}-\boldsymbol{x}^*)\nabla g(\boldsymbol{x}^*) + \ldots = (\boldsymbol{x}-\boldsymbol{x}^*)\nabla g(\boldsymbol{x}^*) + \ldots, \tag{7}$$

about the point $\boldsymbol{x}^*$, which is usually chosen to be the point on the limit state surface with the highest probability density. This point is known as the design point, and can be obtained by solving the optimisation program $\boldsymbol{x}^* = \text{argmin}_{\boldsymbol{x}}\{|\boldsymbol{x}|^2 : g(\boldsymbol{x}) = 0\}$. Alternatively, using the assumption of a linear performance function, the design point can be determined using the gradient of the performance function. The reliability index is defined as $\beta = \sqrt{|\boldsymbol{x}^*|^2}$, and in the case of normally distributed random variables and a linear limit state function $P_f = \phi(-\beta)$. This can be shown by observing that when $\boldsymbol{x}^*$ has a standard normal distribution and $g(\boldsymbol{x})$ is linear (as in (7), the system performance will have a normal distribution with mean $\mathbb{E}_{\boldsymbol{x}}(g(\boldsymbol{x})) = (\mathbb{E}_{\boldsymbol{x}}(\boldsymbol{x}) - \boldsymbol{x}^*)\nabla g(\boldsymbol{x}^*) = -\boldsymbol{x}^*\nabla g(\boldsymbol{x}^*)$ and variance $\text{Var}_{\boldsymbol{x}}(g(\boldsymbol{x})) = |\nabla g(\boldsymbol{x}^*)|^2$. Therefore, since $\boldsymbol{x}^* = \beta\frac{\nabla g(\boldsymbol{x}^*)}{\sqrt{|\nabla g(\boldsymbol{x}^*)|^2}}$, $P_f = \phi\left(\frac{-\mathbb{E}_{\boldsymbol{x}}(g(\boldsymbol{x}))}{\sqrt{\text{Var}_{\boldsymbol{x}}(g(\boldsymbol{x}))}}\right)$ leads to the desired expression.

A similar method relying on a more accurate approximation of the limit state function is the Second Order Reliability Method.

The main advantage of FORM is the small number of samples required to estimate the failure probability. The main disadvantage of FORM is that for non-linear limit state surfaces the method is likely to be extremely inaccurate, due to the degradation of the Taylor series approximation for limit state surfaces with high curvature. Non-linear limit state surfaces are often induced by the transformation of the system's random variables to the standard normal space.

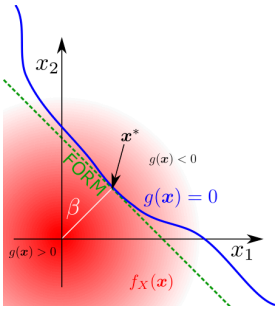


Fig. 1: A diagram of the First Order Reliability Method for two system variables, shown with random variables in the standard normal space.

5.2.4 Line sampling

The fundamental idea behind Line Sampling is to refine estimates obtained from the First-order reliability method (FORM), which may be incorrect due to the non-linearity of the limit state function. Conceptually, this is achieved by averaging the result of different FORM simulations [113]. Firstly, the approximate direction of the failure region from the origin in standard normal space must be determined. This is known as the importance direction. It is usually obtained by finding the design point, by approximate means if necessary. Following this, samples are randomly generated in the standard normal space and lines are drawn parallel to the importance direction in order to compute the distance to the limit state function, which enables the probability of failure to be estimated for each sample.

For each sample of $\boldsymbol{x}$, the probability of failure in the line parallel to the important direction is defined as: $P_f(\boldsymbol{x}) = \int_{-\infty}^{\infty} \mathbb{I}(\boldsymbol{x}+\beta\boldsymbol{\alpha})d\beta$, where $\boldsymbol{\alpha}$ is the importance direction, and ϕ is the probability density function of a Gaussian distribution (and β is a real number). In practice, the roots of a nonlinear function must be found to estimate the partial probabilities of failure along each line. This is either done by interpolation of a few samples along the line, or by using the Newton-Raphson method. The global probability of failure is the mean of the probability of failure on the lines: $P_f = \frac{1}{N_L}\sum_i^{N_L} P_f^{(i)}$ where N_L is the total number of lines used in the analysis, and $P_f^{(i)}$ are the partial probabilities of failure estimated along all the lines.

For problems in which the dependence of the performance function is only moderately non-linear with respect to the parameters modelled as random variables, setting the importance direction as the gradient vector of the performance function in the underlying standard normal space leads to highly efficient Line Sampling. [26] describes enhancements which can be made to Line Sampling to increase the efficiency. For example, the solution of the Newton-Raphson search used on the previous line can be used to inform the search on the next line, if the lines are sorted by proximity. In addition, the importance direction can be updated during simulation based on the completed subset of lines.

The Line Sampling methodology is more expensive than FORM, but far less expensive than Monte Carlo simulation. It is likely to perform poorly for highly non-linear limit state surfaces, but in general offers a good balance between accuracy and computational expense.

5.2.5 Importance sampling

In Importance Sampling, samples are drawn from a distribution with a higher density in the failure region and then re-weighted to obtain a Monte Carlo estimator with reduced variance. The re-weighted estimator is written as

$$P_f = \int h(\boldsymbol{x}) \frac{\mathbb{I}_f(\boldsymbol{x}) f_X(\boldsymbol{x})}{h(\boldsymbol{x})} d\boldsymbol{x} = \frac{1}{N} \sum_{i=1}^{N} \frac{\mathbb{I}_f(\boldsymbol{x}_i) f_X(\boldsymbol{x}_i)}{h(\boldsymbol{x}_i)}, \tag{8}$$

where $\boldsymbol{x}_i$ are drawn from the proposal density $h(\boldsymbol{x})$. The optimal proposal density, which results in the greatest reduction of the variance of the estimator is

$$h(\boldsymbol{x})_{\text{optimal}} = \frac{\mathbb{I}_f(\boldsymbol{x}) f_X(\boldsymbol{x})}{P_f}, \tag{9}$$

which is not useful in practice because of the dependence on the quantity to be estimated, P_f. However, the optimal proposal density can be used to motivate the choice of the proposal density in practice. An appropriate $h(\boldsymbol{x})$ can be chosen by finding the design point with an approximate method and centring the proposal density on the design point, since (9) indicates that the failure region has a higher proposal density. A complete discussion of the technique is given in [114] and [106].

Importance Sampling is useful as it offers an unbiased estimator which can estimate the failure probability with few samples. The main difficultly is determining the proposal distribution $h(\boldsymbol{x})$. This is usually achieved by engineering judgement and knowledge of the design point.

5.2.6 Subset simulation

Subset simulation aims to calculate P_f by decomposing the space of the random variables into several intermediate failure events with decreasing failure probability. The conditional probabilities for the intermediate failure regions can then be used to calculate P_f which is given by $P_f = P(F_m) = P(F_m) \prod_{i=1}^{m-1} P(F_{i+1}|F_i)$ where F_i represents intermediate failure event i. By making the conditional probability of samples falling in the intermediate failure regions large, the coefficient of variation of each individual failure event can be minimised, hence minimising the coefficient of variation of P_f. Markov chains are used to generate conditional samples between intermediate failure regions in order to calculate $P(F_{i+1}|F_i)$. A complete description of the method is given in [115].

The main advantage of Subset Simulation is that it can estimate failure probabilities for non-linear limit state surfaces in a black box manner with relatively low computational expense. However, [116] shows that subset simulation is not accurate for some limit state surfaces, for example limit state surfaces with multiple importance directions.

5.2.7 Metamodels

If inexpensive samples of $g(\boldsymbol{x})$ or $\mathbb{I}_f(\boldsymbol{x})$ are available then the estimator $\hat{P}_f$ can be evaluated trivially. Therefore, the problem of estimating P_f can be effectively reduced to a machine learning problem. In the case of modelling $g(\boldsymbol{x})$, the problem is one of regression. In the case of modelling $\mathbb{I}_f(\boldsymbol{x})$, the problem is classification of the failure region. A machine learning model which fulfils this purpose is known as a metamodel or surrogate model.

In the literature many machine learning techniques have been applied to the reliability analysis problem: linear regression (known as the response surface methodology) [117], support vector machines [118], polynomial chaos expansions [119], neural networks [120] and Gaussian process emulators (sometimes known as Kriging) [121]. Neural networks and Gaussian processes have the advantage of being able to quantify their uncertainty accurately, so the required number of training samples can be assessed. Polynomial chaos expansions allow the sensitivity indices of $g(\boldsymbol{x})$ to be computed analytically from the trained metamodel [122]. In [123], Interval Predictor Models are used to solve the reliability analysis problem.

In general, the most useful metamodels produce the most accurate estimates of P_f, whilst requiring the smallest number of training samples. An 'experimental design' specifies where the samples of $g(\boldsymbol{x})$ will be made for training. Usually a

uniform design is chosen, but other sampling strategies can be used [124]. Active learning can be used to sequentially choose the samples required to train the metamodel. These samples are usually chosen based on where the uncertainty of the metamodel is largest. In adaptive Kriging Monte Carlo simulation (AK-MCS) the samples are chosen at points with large uncertainty, close to the limit state surface. This strategy achieves state of the art efficiency [125]. This strategy is known as active learning, and the function which is used to choose the subsequent sample is known as the probability of misclassification acquisition function.

The main advantage of metamodels is that the estimator for the failure probability based on the metamodel can be made arbitrarily accurate. The main disadvantage is that the metamodel introduces uncertainties, so the problem is effectively shifted to trying to create an accurate metamodel with a small number of samples.

5.3 Worked example : First order reliability analysis method (FORM)

For the simple example of a cantilever beam with a point load, F, at any point on the beam, the maximum deflection of the end of the beam is given by $\delta_{max} = \frac{Fa^2}{6EI}(3l - a)$, where I is the moment of inertia of the beam, a is the distance of the point load from the fixed end of the beam, l is the length of the beam and E is the modulus of elasticity of the beam [126]. E, I and a were fixed, and l and F were given by random variables with normal distributions. The chosen values of the parameters are shown below:

Variable	Distribution	Mean	Standard Deviation
E	Fixed	200000 N/mm^2	N/A
I	Fixed	78125000 mm^4	N/A
l	Normal	5000 mm	20 mm
a	Fixed	3000 mm	N/A
F	Normal	30000 N	20 N

It is assumed that the beam 'fails' when the maximum deflection is greater than 35 mm.

First we need to determine the reliability index (β). This can be achieved by several methods:

```
# Using optimisation
import scipy.optimize
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt


def deflection(force, distance, elasticity, inertia, length):
    """
    Deflection of a cantilever beam

    Args:
        force: Force
        distance: Distance of the point load from the fixed end of the beam
        elasticity: Modulus of elasticity of the beam
        inertia: Beam moment of inertia
        length: Beam length
    Returns:
        deflection
    """
    return (force * distance ** 2 * (3 * length - distance)
            / (6 * elasticity * inertia))
```

(continues on next page)

(continued from previous page)

```
performance_threshold = 35


def performance(length, force):
    return (performance_threshold
            - deflection(force, 3000, 200000, 78125000, length))


mean_l = 5000
std_l = 20
mean_F = 30000
std_F = 20


def normalise(variable, mean, std):
    """
    Transform variable to standard normal space
    """
    return (variable - mean) / std


def unnormalise(variable, mean, std):
    """
    Transform variable from standard normal space
    """
    return (variable * std + mean)


def transform(length, force):
    return normalise(length, mean_l, std_l), normalise(force, mean_F, std_F)


def inverse_transform(length, force):
    return (unnormalise(length, mean_l, std_l),
            unnormalise(force, mean_F, std_F))


results = scipy.optimize.minimize(
    lambda x: np.linalg.norm(x),
    x0=[0, 0],
    constraints=scipy.optimize.NonlinearConstraint(
        lambda x: performance(*inverse_transform(*x)), lb=0, ub=0),
)
beta = results.fun
design_point = results.x
P_f = norm.cdf(-beta)
print(P_f)
```

```
0.0058112134307528535
```

```
# Using gradient of performance function

def gradient_performance_function_lf(force, distance, elasticity, inertia,
                                     length):
    """
    Gradient of performance function with respect to l and F
```

(continues on next page)

(continued from previous page)

```
        Args:
        force: Force
        distance: Distance of the point load from the fixed end of the beam
        elasticity: Modulus of elasticity of the beam
        inertia: Beam moment of inertia
        length: Beam length
    """
    dg_dl = -3 * force * distance ** 2 / (6 * elasticity * inertia)
    dg_dF = -distance ** 2 * (3*length - distance) / (6 * elasticity * inertia)
    return np.array([dg_dl, dg_dF])


def unnormalised_gradient_at_mean(length, force):
    return gradient_performance_function_lf(force, 3000, 200000, 78125000,
                                            length)


gradient_at_mean = unnormalised_gradient_at_mean(
    *inverse_transform(mean_l, mean_F))
importance_direction = - gradient_at_mean / np.linalg.norm(gradient_at_mean)
beta = performance(mean_l, mean_F) / np.linalg.norm(gradient_at_mean)
design_point = importance_direction * beta
P_f = norm.cdf(-beta)
print(P_f)
```

```
0.008363326195416835
```

```
# Check result using Monte Carlo
n_samples = 1000000
samples_standardised = np.random.randn(n_samples, 2)
samples_l, samples_F = inverse_transform(
    samples_standardised[:, 0], samples_standardised[:, 1])
P_f = np.mean(performance(samples_l, samples_F) < 0)
print(P_f)
```

```
0.005767
```

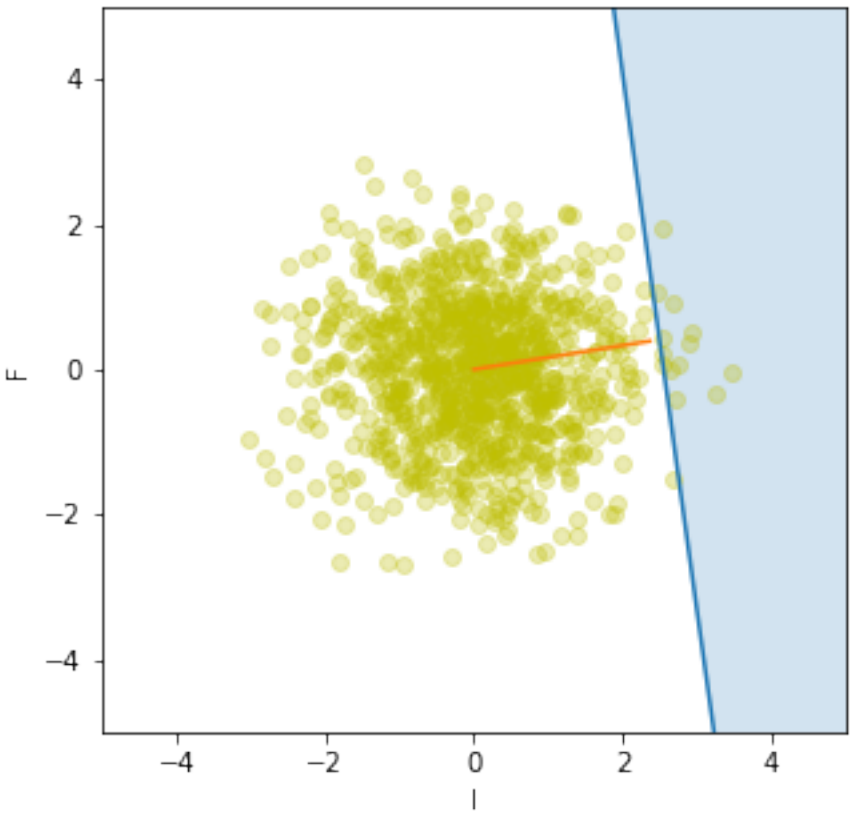
Below, a plot is shown of the model:

```
def limit_state_F(length, distance, elasticity, inertia):
    """
    Parametric equation for limit state surface for F as a function of l
    """
    return 35 / (distance ** 2 * (3 * length - distance)
                 / (6 * elasticity * inertia))


# Plot the limit state surface - where performance function is zero
x_plot = np.arange(-10, 10)
unnormalised_x = unnormalise(x_plot, mean_l, std_l)
limit_state = limit_state_F(unnormalised_x, 3000, 200000, 78125000)
normalised_limit_state = normalise(limit_state, mean_F, std_F)
plt.plot(x_plot, normalised_limit_state)
# Shade failure region - to the right of limit state surface
```

(continues on next page)

(continued from previous page)

```
# (performance function < 0)
plt.fill_between(x_plot, normalised_limit_state, 5, alpha=0.2)

# Plot some samples from the Monte Carlo simulation
n_samples_to_plot = 1000
plt.scatter(
    samples_standardised[:n_samples_to_plot, 0],
    samples_standardised[:n_samples_to_plot, 1],
    alpha=0.3, color="y"
)

# Plot beta - the line linking design point with the origin
plt.plot([0, design_point[0]], [0, design_point[1]])

# Set plot properties
fig = plt.gcf()
fig.set_size_inches(5, 5)
plt.xlim(-5, 5)
plt.ylim(-5, 5)
plt.xlabel("l")
plt.ylabel("F")
plt.show()
```



CHAPTER

SIX

CONVEX SET MODELS FOR RELIABILITY

6.1 Problem definition

In contrast to probabilistic models of uncertainty, where the statistics of $g(\boldsymbol{x})$ — and hence the failure probability — can be determined, in convex uncertainty models the focus is on the best and worst possible values for $g(\boldsymbol{x})$. This is good, because underestimation of the worst case due to inaccurate sampling in the tails of probability distributions can be avoided. In addition, the analysis can include extreme lack of knowledge of the possible loads the system will be subjected to. However, sometimes the analysis may be overly conservative if the worst case is extremely unlikely to occur. Convex set models of uncertainty also offer a framework to analyse a possible set of future designs, before the design for a system has been finalised.

For system parameters $\boldsymbol{x} \in X$, the structural response is given by the interval $[\min_{\boldsymbol{x}\in X} g(\boldsymbol{x}), \max_{\boldsymbol{x}\in X} g(\boldsymbol{x})]$, which can be determined by the methods discussed in *Convex set models of uncertainty* [97]. For a system with response $g_{\text{load}}(x)$ which must not exceed $g_{\text{threshold}}$, [98] proposes a non-probabilistic figure of merit, $R_{\text{non-probabilistic}} = 1 - \frac{\overline{g}_{\text{load}}(\boldsymbol{x})}{g_{\text{threshold}}}$, where $\overline{g}_{\text{load}}(x) = \max_{\boldsymbol{x}\in X} g_{\text{load}}(\boldsymbol{x})$. $R_{\text{non-probabilistic}}$ describes how close the system is to failure, but it cannot be interpreted as a probability or rate of failure.

CHAPTER

SEVEN

RELIABILITY ANALYSIS WITH PROBABILITY BOXES

7.1 Problem definition

Probability boxes offer a hybrid of the convex set and probabilistic approaches for reliability analysis. In structural reliability analysis with probability boxes, the objective is to compute the failure probability in the same sense as with traditional random variables in *Reliability theory*. However, it is now impossible to compute an exact value for the failure probability; only bounds on the failure probability are available [99]. For distributional probability boxes, $f_{X_{\boldsymbol{\theta}}}(\boldsymbol{x})$ for $\boldsymbol{\theta} \in \Theta$, the bounds on the failure probability can be found by solving the integrals

$$\underline{P}_f = \underline{\mathrm{P}}[g(\boldsymbol{x}) < 0] = \min_{\boldsymbol{\theta} \in \Theta} \int_{\mathbb{R}^N} \mathbb{I}_f(\boldsymbol{x}) f_{X_{\boldsymbol{\theta}}}(\boldsymbol{x}) d\boldsymbol{x}, \tag{1}$$

and

$$\overline{P}_f = \overline{\mathrm{P}}[g(\boldsymbol{x}) < 0] = \max_{\boldsymbol{\theta} \in \Theta} \int_{\mathbb{R}^N} \mathbb{I}_f(\boldsymbol{x}) f_{X_{\boldsymbol{\theta}}}(\boldsymbol{x}) d\boldsymbol{x}. \tag{2}$$

For distribution-free probability boxes given by $[\underline{F}_i(x_i), \overline{F}_i(x_i)]$, each system variable can be written as a function of separate probabilistic and set based variables as $x_i' = \underline{F}_i^{-1}(\alpha_i) + (\overline{F}_i^{-1}(\alpha_i) - \underline{F}_i^{-1}(\alpha_i))\theta_i$, where the aleatory variable $\alpha = (\alpha_1, \alpha_2, \ldots)$ is a uniformly distributed random vector with the same dimensionality as $\boldsymbol{x}$, and $\boldsymbol{\theta} \in \Theta$ is the unit hyper-cube with the same dimensionality as $\boldsymbol{x}$ [26]. This enables the performance function $g(\boldsymbol{x})$ to be rewritten in terms of $\boldsymbol{\alpha}$ and $\boldsymbol{\theta}$, i.e. $g(\boldsymbol{\alpha}, \boldsymbol{\theta})$. Bounds on the failure probability can then be obtained by evaluating

$$\underline{P}_f = \underline{\mathrm{P}}[g(\boldsymbol{x}) < 0] = \mathrm{P}[\underline{g}(\boldsymbol{\alpha}) < 0] \tag{3}$$

and

$$\overline{P}_f = \overline{\mathrm{P}}[g(\boldsymbol{x}) < 0] = \mathrm{P}[\overline{g}(\boldsymbol{\alpha}) < 0], \tag{4}$$

where the upper and lower performance function are obtained from $\underline{g}(\boldsymbol{\alpha}) = \min_{\boldsymbol{\theta} \in \Theta} g(\boldsymbol{\alpha}, \boldsymbol{\theta})$ and $\overline{g}(\boldsymbol{\alpha}) = \max_{\boldsymbol{\theta} \in \Theta} g(\boldsymbol{\alpha}, \boldsymbol{\theta})$

By finding the envelope of a distributional probability box, the algorithm for computing the failure probability for the distribution-free case can be applied. As expected, [100] demonstrate that this results in overly conservative bounds on the failure probability, since clearly information is lost by taking the envelope of the distributional probability box. Therefore, only (1) and (2) should be applied when computing failure probabilities with distributional probability boxes.

7.2 Methods to compute the failure probability

In this section we concentrate on methods to compute the failure probability for distributional probability boxes.

7.2.1 Monte Carlo estimators

A Monte Carlo estimator can be applied for the integrals in the failure probability computation with distribution-free probability boxes ((1) and (2) to yield the bounds $[\underline{P}_f, \overline{P}_f] = [\min_{\boldsymbol{\theta}\in\Theta} \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}_f(\boldsymbol{x}_{\boldsymbol{\theta}}^{(i)}), \max_{\boldsymbol{\theta}\in\Theta} \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}_f(\boldsymbol{x}_{\boldsymbol{\theta}}^{(i)})]$, where the samples $\boldsymbol{x}_{\boldsymbol{\theta}}^{(i)}$ are drawn from $f_{X_{\boldsymbol{\theta}}}(\boldsymbol{x})$. This is the double loop Monte Carlo approach; an inner loop is used to compute a Monte Carlo estimator which is optimised over in the outer loop [28]. The outer loop optimisation can be evaluated using brute force grid sampling of $\boldsymbol{\theta}$, which is known as naïve double loop Monte Carlo simulation. It is usually more efficient to use an efficient global optimisation algorithm to evaluate the optimisation loop, such as Bayesian Optimisation, or a genetic algorithm [26].

Evaluating the failure probability using double loop Monte Carlo simulation is computationally expensive, since now each inner loop Monte Carlo estimator must be computed multiple times. This is particularly the case when the problem dimensionality is large or the failure probability is small. [29] shows that the bias of the estimator is negative, and the magnitude of the bias decreases as more samples are made.

7.2.2 Imprecise First Order Reliability Method

A generalisation of FORM for systems with components which are described by probability boxes was introduced by [101]. The system's performance function must be written in the load resistance form ((2)), and the system must have one strength and one load component. Therefore, the system variables consist of the resistance variable with mean $\mu_R \in [\underline{\mu_R}, \overline{\mu_R}]$ and standard deviation $\sigma_R \in [\underline{\sigma_R}, \overline{\sigma_R}]$, and load variable with mean $\mu_L \in [\underline{\mu_L}, \overline{\mu_L}]$ and standard deviation $\sigma_L \in [\underline{\sigma_L}, \overline{\sigma_L}]$. Then the failure probability lies in the interval $[\underline{P}_f, \overline{P}_f] = [\phi(-\overline{\beta}), \phi(-\underline{\beta})]$, where $\overline{\beta} = \frac{\overline{\mu}_R - \underline{\mu}_L}{\underline{\sigma_L}^2 + \underline{\sigma_R}^2}$, and $\underline{\beta} = \frac{\underline{\mu}_R - \overline{\mu}_L}{\overline{\sigma}_L^2 + \overline{\sigma}_R^2}$.

In more complex cases, one may need to solve an optimisation program to find the reliability index [102]. For example, one could imagine a system which fails if the sum of many different products of probability box distributed variables falls below a threshold.

7.2.3 Line sampling

[26] describes two ways in which Line Sampling can be used to increase the efficiency of probability box propagation. Line Sampling can be applied as an alternative to the Monte Carlo estimator used to approximate the integral in the double loop approach ((1) and (2)). Alternatively, Line Sampling can be applied to the aleatory variables $\boldsymbol{\alpha}$ for the upper and lower performance functions in (3) and (4). When Line Sampling is applied in the aleatory space, the importance direction updating strategy proposed by [26] significantly increases the accuracy of the computation. Judged by number of samples required for computation, Line Sampling is close to the state of the art. However, Line Sampling is ineffective on highly non-linear limit state surfaces.

7.2.4 Importance sampling

The Importance Sampling estimator in (8) can be applied to greatly reduce the number of samples required when computing the failure probability for a system subject to probability box random variables [100]. The bounds on the failure probability are given by $\underline{P}_f = \min_{\boldsymbol{\theta}\in\Theta} \int \mathbb{I}_f(\boldsymbol{x}) \frac{f_{X_{\boldsymbol{\theta}}}(\boldsymbol{x})}{h(\boldsymbol{x})} h(\boldsymbol{x}) d\boldsymbol{x} = \min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}_f(\boldsymbol{x}^{(i)}) \frac{f_{X_{\boldsymbol{\theta}}}(\boldsymbol{x}^{(i)})}{h(\boldsymbol{x}^{(i)})}$ and $\overline{P}_f = \max_{\boldsymbol{\theta}\in\Theta} \int \mathbb{I}_f(\boldsymbol{x}) \frac{f_{X_{\boldsymbol{\theta}}}(\boldsymbol{x})}{h(\boldsymbol{x})} h(\boldsymbol{x}) d\boldsymbol{x} = \max_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}_f(\boldsymbol{x}^{(i)}) \frac{f_{X_{\boldsymbol{\theta}}}(\boldsymbol{x}^{(i)})}{h(\boldsymbol{x}^{(i)})}$, where the samples $\boldsymbol{x}^{(i)}$ are drawn from the proposal distribution $h(\boldsymbol{x})$. The proposal distribution can be iteratively updated to provide more accurate results. [103] demonstrates that the bias in both cases is negative and decreases in magnitude as more samples are collected. The Importance Sampling estimator requires a similar number of samples to Line Sampling.

7.2.5 Multi level metamodels

Multilevel Metamodelling requires the creation of two Gaussian Process emulators [104]. The first metamodel is created for the performance function in the space of the system variables $\boldsymbol{x}$, using Adaptive Kriging Monte Carlo simulation. The second metamodel is used to perform Bayesian Optimisation on the obtained failure probability from the first metamodel, in the epistemic space. This greatly reduces the amount of repeated similar evaluations of the system model, and hence the Multi-level metamodelling technique is close to state of the art when judged by number of required samples.

7.2.6 Interval Predictor Models

[105] uses an IPM as a metamodel to propagate probability boxes. In the first two algorithms, the performance function is modelled as a function with unknown noise structure in the aleatory space and supplemented by a modified performance function. In the third algorithm, an Interval Predictor Model is constructed and a re-weighting strategy used to find bounds on the probability of failure.

CHAPTER

EIGHT

CHAPTER SUMMARY

In this chapter we reviewed the application of the uncertainty models introduced in *Models of Uncertainty* to the field of reliability engineering. Specifically, this chapter demonstrated how probabilistic, set-based and imprecise probability models can be used to calculate the reliability of systems under uncertainty. The optimal design of a system under uncertainty can also be computed, and the local or global sensitivity of the response of a system to changes in the system variables can be determined. For systems where the probability of failure is small, computing the failure probability using a Monte Carlo estimator can be computationally expensive. For this reason, it is necessary to apply advanced techniques in order to calculate the probability of failure in a feasible computational time. For imprecise probability models, computation of the failure probability of the system is even more expensive, and hence efficient computational techniques are also required. This motivates the novel contributions introduced in the following chapters.

Part V

Conclusion

The purpose of this book is to explain how uncertainty quantification can be accurately and reliably achieved for engineering systems, when the available data is of poor quality or if a limited quantity of data is available. The computational techniques required to perform this simulation in a feasible computational time are of particular importance. The developed uncertainty quantification techniques have been discussed in the context of reliability engineering, however the potential applications of these techniques are much wider than this.

Chapter *Models of Uncertainty* presented a review of uncertainty models which model joint uncertainty of unknown variables (e.g. *generative* probabilistic models, convex sets and probability boxes). Techniques for general computation with such models are described. The construction of such models with or without experimental data was discussed.

Chapter *Machine Learning of Regression Models* discussed techniques to create uncertainty models, where the uncertainty in one variable is dependent on the uncertainty in other variables. These models include Bayesian regression, neural networks, Gaussian processes and interval predictor models. Techniques to validate the performance of these models is discussed.

Chapter *Reliability Analysis* presented a state of the art review of techniques for reliability analysis, where the probability of failure of a system under the influence of uncertainty is calculated. Reliability measures were discussed for probabilistic models, convex set models and imprecise probability models. Efficient computational techniques were discussed for performing the simulation required to calculate the reliability of expensive black-box models of systems.

CHAPTER

NINE

UNANSWERED QUESTIONS

The field of uncertainty quantification is still developing, and undoubtably many future interesting applications of uncertainty quantification are yet to be made.

There are several areas where development is likely to focus in future:

- Further application of the latest developments in neural networks and Gaussian processes to engineering.
- Algorithms amenable to massive data sets, particularly algorithms involving imprecise probabilities.
- Further development and application of imprecise probability models in some form, though they may not be labelled explicitly as such.

CHAPTER

TEN

WHAT SHOULD I READ NEXT?

After reading this book and being presented with an overview of the field, engineers may be searching for further reading material. A useful recommendation is largely dependent on the interests of the reader. Readers with a general interest in statistics may benefit from reading textbooks such as [9], [7], [8] and [1]. Readers wishing to understand more about engineering applications of uncertainty should refer to [86] or [106] (many similar books exist). For those in industry, whitepapers from government organisations may also be informative. For those beginning a postgradutate programme of study, I highly recommend reading the original papers for techniques of interest and then reviewing the code of open source software libraries such as [127].

Part VI

Bibliography

BIBLIOGRAPHY

[1] Edwin T Jaynes. *Probability theory: The logic of science*. Cambridge University Press, 2003.

[2] Silvia Tolo, Xiange Tian, Nils Bausch, Victor Becerra, TV Santhosh, Gopika Vinod, and Edoardo Patelli. Robust on-line diagnosis tool for the early accident detection in nuclear power plants. *Reliability Engineering & System Safety*, 186:110–119, 2019. doi:10.1016/j.ress.2019.02.015.

[3] L Zadeh. On the validity of dempster's rule of combination, memo m 79/24. *Univ. of California, Berkeley*, 1979.

[4] Simon Maskell. A Bayesian approach to fusing uncertain, imprecise and conflicting information. *Information Fusion*, 9(2):259–277, 2008. doi:10.1016/j.inffus.2007.02.003.

[5] Michael S Balch. A corrector for probability dilution in satellite conjunction analysis. In *18th AIAA Non-Deterministic Approaches Conference*, 1445. 2016. doi:10.2514/6.2016-1445.

[6] Michael Scott Balch, Ryan Martin, and Scott Ferson. Satellite conjunction analysis and the false confidence theorem. *Proceedings of the Royal Society A*, 475(2227):20180565, 2019. doi:10.1098/rspa.2018.0565.

[7] Andrew Gelman, Hal S Stern, John B Carlin, David B Dunson, Aki Vehtari, and Donald B Rubin. *Bayesian data analysis*. Chapman and Hall/CRC, 2013. doi:10.1201/b16018.

[8] Albert Tarantola. *Inverse problem theory and methods for model parameter estimation*. Volume 89. SIAM, 2005. doi:10.1137/1.9780898717921.

[9] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Volume 1. Springer series in statistics New York, 2001. doi:10.1007/BF02985802.

[10] PL Green and Simon Maskell. Estimating the parameters of dynamical systems from big data using sequential monte carlo samplers. *Mechanical Systems and Signal Processing*, 93:379–396, 2017. doi:10.1016/j.ymssp.2016.12.023.

[11] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

[12] Katalin Csilléry, Michael GB Blum, Oscar E Gaggiotti, and Olivier François. Approximate bayesian computation (abc) in practice. *Trends in ecology & evolution*, 25(7):410–418, 2010. doi:10.1016/j.tree.2010.04.001.

[13] Jonathan Sadeghi, Marco de Angelis, and Edoardo Patelli. Frequentist history matching with interval predictor models. *Applied Mathematical Modelling*, 61:29–48, September 2018. doi:10.1016/j.apm.2018.04.003.

[14] Samuel S Wilks. Determination of sample sizes for setting tolerance limits. *The Annals of Mathematical Statistics*, 12(1):91–96, 1941. doi:10.1214/aoms/1177731788.

[15] Victoria Montgomery. *New statistical metliods in risk assessment by probability bounds*. PhD thesis, Durham University, 2009.

[16] James O Berger. *Statistical Decision Theory and Bayesian Analysis*. Springer Science & Business Media, 1985. doi:10.1007/978-1-4757-4286-2.

[17] Scott Ferson, Jason O'Rawe, and Michael Balch. Computing with confidence: imprecise posteriors and predictive distributions. In *Vulnerability, Uncertainty, and Risks: Quantification, Mitigation, and Management*, pages 895–904. 2014. doi:10.1061/9780784413609.091.

[18] Andrey Kolmogoroff. Confidence limits for an unknown distribution function. *The annals of mathematical statistics*, 12(4):461–463, 1941. doi:10.1214/aoms/1177731684.

[19] Scott Ferson, Vladik Kreinovich, Lev Ginzburg, Davis S Myers, and Kari Sentz. Constructing probability boxes and dempster-shafer structures. Technical Report, Technical report, Sandia National Laboratories, 2003.

[20] NIST/SEMATECH. E-handbook of statistical methods. http://www.itl.nist.gov/div898/handbook, 2012.

[21] Sheldon Ross. *A first course in probability*. Pearson, 2014.

[22] C Sundararajan. Expert opinion in probabilistic structural mechanics. In *Probabilistic Structural Mechanics Handbook*, pages 261–279. Springer, 1995. doi:10.1007/978-1-4615-1771-9_12.

[23] JE Oakley and A O'Hagan. SHELF: The Sheffield Elicitation Framework (Version 2.0). School of Mathematics and Statistics, University of Sheffield. 2010. URL: http://www.tonyohagan.co.uk/shelf/.

[24] John Kruschke. *Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan*. Academic Press, 2014.

[25] Scott Ferson, William L Oberkampf, and Lev Ginzburg. Validation of imprecise probability models. *Int. J. Reliab. Saf*, 3(1-3):3–22, 2009. doi:10.1504/IJRS.2009.026832.

[26] Marco De Angelis. *Efficient Random Set Uncertainty Quantification by means of Advanced Sampling Techniques*. PhD thesis, University of Liverpool, 2015.

[27] Diego A Alvarez. On the calculation of the bounds of probability of events using infinite random sets. *International journal of approximate reasoning*, 43(3):241–267, 2006. doi:10.1016/j.ijar.2006.04.005.

[28] Edoardo Patelli, Diego A Alvarez, Matteo Broggi, and Marco de Angelis. Uncertainty management in multidisciplinary design of critical safety systems. *Journal of Aerospace Information Systems*, 12(1):140–169, 2014. doi:10.2514/1.I010273.

[29] Matthias CM Troffaes. Imprecise Monte Carlo simulation and iterative importance sampling for the estimation of lower previsions. *International Journal of Approximate Reasoning*, 101:31–48, 2018. doi:10.1016/j.ijar.2018.06.009.

[30] Roberto Rocchetta and Edoardo Patelli. Stochastic analysis and reliability-cost optimization of distributed generators and air source heat pumps. In *System Reliability and Safety (ICSRS), 2017 2nd International Conference on*, 31–35. IEEE, 2017. doi:10.1109/ICSRS.2017.8272792.

[31] Andrei N. Kolmogorov. *Foundations of the Theory of Probability*. Chelsea Publishing Company, New York, 1950.

[32] Jonathan Sadeghi, Marco De Angelis, and Edoardo Patelli. Analytic probabilistic safety analysis under severe uncertainty. *ASCE-ASME Journal of Risk and Uncertainty in Engineering Systems, Part A: Civil Engineering*, 6(1):04019019, 2020.

[33] Reuven Y Rubinstein and Dirk P Kroese. *Simulation and the Monte Carlo method*. Volume 10. John Wiley & Sons, 2016.

[34] William H Press, Brian P Flannery, Saul A Teukolsky, and William T Vetterling. Numeric recipes in c: the art of scientific computing. 1992.

[35] Eric A Wan and Rudolph Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No. 00EX373)*, 153–158. Ieee, 2000. doi:10.1109/ASSPCC.2000.882463.

[36] Jeffrey K Uhlmann. *Dynamic map building and localization: New theoretical foundations*. PhD thesis, University of Oxford Oxford, 1995.

[37] Han Zhang, Yabing Li, Jianjun Xiao, and Thomas Jordan. Preliminary study of condensation heat transfer uncertainty quantification using cfd code gasflow-mpi. In *ANS Best Estimate Plus Uncertainty International Conference (BEPU 2018)*. 2018.

[38] G. Perret, S. Rahman, and D. Wicaksono. Deterministic sampling and gaussian process metamodel approach applied to a station black out accident model. In *ANS Best Estimate Plus Uncertainty International Conference (BEPU 2018)*. 2018.

[39] Ramon E. Moore, R. Baker Kearfott, and Michael J. Cloud. *Introduction to interval analysis*. Society for Industrial and Applied Mathematics, 2009. doi:10.1137/1.9780898717716.bm.

[40] Vladik Kreinovich, Wolfram Luther, and Evgenija D. Popova. Special issue on "uncertainty modeling and analysis with intervals: foundations, tools, applications". *Soft Computing*, 17(8):1315–1317, Aug 2013. doi:10.1007/s00500-013-1004-z.

[41] Eldon R Hansen. A generalized interval arithmetic. In *International Symposium on Interval Mathematics*, 7–18. Springer, 1975. doi:10.1007/3-540-07170-9_2.

[42] Kyoko Makino and Martin Berz. Taylor models and other validated functional inclusion methods. *International Journal of Pure and Applied Mathematics*, 6:239–316, 2003.

[43] Nedialko S Nedialkov, Vladik Kreinovich, and Scott A Starks. Interval arithmetic, affine arithmetic, Taylor series methods: why, what next? *Numerical Algorithms*, 37(1-4):325–336, 2004. doi:10.1023/B:NUMA.0000049478.42605.cf.

[44] Luis G Crespo, Daniel P Giesy, and Sean P Kenny. A unifying framework to uncertainty quantification of polynomial systems subject to aleatory and epistemic uncertainty. *Reliable Computing*, 17(2):97–127, 2012.

[45] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[46] Sébastien Bubeck and others. Convex optimization: algorithms and complexity. *Foundations and Trends® in Machine Learning*, 8(3-4):231–357, 2015. doi:10.1561/2200000050.

[47] V Kreinovich, J Beck, C Ferregut, A Sanchez, GR Keller, M Averill, and SA Starks. Monte-carlo-type techniques for processing interval uncertainty, and their engineering applications. In *Proceedings of the workshop on reliable engineering computing*, 15–17. 2004.

[48] Yakov Ben-Haim and Isaac Elishakoff. *Convex models of uncertainty in applied mechanics*. Volume 25. Elsevier, 2013. doi:10.1137/1035076.

[49] Jorge Stolfi and Luiz Henrique de Figueiredo. An introduction to affine arithmetic. *Trends in Applied and Computational Mathematics*, 4(3):297–312, 2003.

[50] Bernard Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete & Computational Geometry*, 10(4):377–409, Dec 1993. doi:10.1007/BF02573985.

[51] Shiqi Wang, Yizheng Chen, Ahmed Abdou, and Suman Jana. Mixtrain: scalable training of formally robust neural networks. *arXiv preprint arXiv:1811.02625*, 2018.

[52] Zhi-yue Huang and Bin-wu He. Volume of unit ball in an n-dimensional normed space and its asymptotic properties. *Journal of Shanghai University (English Edition)*, 12(2):107–109, Apr 2008. doi:10.1007/s11741-008-0204-1.

[53] David Meyer. Notes on maximization of inner products over norm balls. 2017. URL: http://www.1-4-5.net/~dmm/ml/sgn.pdf (visited on 2019-05-02).

[54] Andrew Y Ng and Michael I Jordan. On discriminative vs. generative classifiers: a comparison of logistic regression and naive bayes. In *Advances in neural information processing systems*, 841–848. 2002.

[55] Will Dabney, Mark Rowland, Marc G Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. *The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, 2017.

[56] Márcio J Lacerda and Luis G Crespo. Interval predictor models for data with measurement uncertainty. In *American Control Conference (ACC), 2017*, 1487–1492. IEEE, 2017. doi:10.23919/ACC.2017.7963163.

[57] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *ICML 2017 Workshop on Principled Approaches to Deep Learning*, 2017.

[58] Marco C Campi, Giuseppe Calafiore, and Simone Garatti. Interval predictor models: identification and reliability. *Automatica*, 45(2):382–392, 2009. doi:10.1016/j.automatica.2008.09.004.

[59] Luis G Crespo, Sean P Kenny, and Daniel P Giesy. Interval predictor models with a linear parameter dependency. *Journal of Verification, Validation and Uncertainty Quantification*, 2016. doi:10.1115/1.4032070.

[60] Hisao Ishibuchi, Hideo Tanaka, and Hidehiko Okada. An architecture of neural networks with interval weights and its application to fuzzy regression analysis. *Fuzzy Sets and Systems*, 57(1):27–39, 1993. doi:10.1016/0165-0114(93)90118-2.

[61] Lei Huang, Bai-Ling Zhang, and Qian Huang. Robust interval regression analysis using neural networks. *Fuzzy sets and systems*, 97(3):337–347, 1998. doi:10.1016/S0165-0114(96)00325-9.

[62] Steffen Freitag, Wolfgang Graf, and Michael Kaliske. Recurrent neural networks for fuzzy data. *Integrated Computer-Aided Engineering*, 18(3):265–280, 2011. doi:10.3233/ICA-2011-0373.

[63] Raquel E Patiño-Escarcina, Benjamín R Callejas Bedregal, and Aarão Lyra. Interval computing in neural networks: one layer interval neural networks. In *International Conference on Intelligent Information Technology*, 68–75. Springer, 2004. doi:10.1007/978-3-540-30561-3_8.

[64] Marco C Campi, Simone Garatti, and Federico A Ramponi. Non-convex scenario optimization with application to system identification. In *2015 54th IEEE Conference on Decision and Control (CDC)*, 4023–4028. IEEE, 2015. doi:10.1109/CDC.2015.7402845.

[65] Algo Carè and Enrico Camporeale. Chapter 4 - regression. In Enrico Camporeale, Simon Wing, and Jay R. Johnson, editors, *Machine Learning Techniques for Space Weather*, pages 71 – 112. Elsevier, 2018. doi:10.1016/B978-0-12-811788-0.00004-4.

[66] Jonathan Sadeghi, Marco De Angelis, and Edoardo Patelli. Efficient training of interval neural networks for imprecise training data. *Neural Networks*, 118:338–351, 2019.

[67] Giuseppe Carlo Calafiore. Random convex programs. *SIAM Journal on Optimization*, 20(6):3427–3464, 2010. doi:10.1137/090773490.

[68] Marco C Campi and Simone Garatti. The exact feasibility of randomized solutions of uncertain convex programs. *SIAM Journal on Optimization*, 19(3):1211–1230, 2008. doi:10.1137/07069821X.

[69] Teodoro Alamo, Roberto Tempo, Amalia Luque, and Daniel R Ramirez. Randomized methods for design of uncertain systems: sample complexity and sequential algorithms. *Automatica*, 52:160–172, 2015. doi:10.1016/j.automatica.2014.11.004.

[70] Javiera Barrera, Tito Homem-de-Mello, Eduardo Moreno, Bernardo K Pagnoncelli, and Gianpiero Canessa. Chance-constrained problems and rare events: an importance sampling approach. *Mathematical Programming*, 157(1):153–189, 2016. doi:10.1007/s10107-015-0942-x.

[71] Algo Carè, Simone Garatti, and Marco C Campi. Scenario min-max optimization and the risk of empirical costs. *SIAM Journal on Optimization*, 25(4):2061–2080, 2015. doi:10.1137/130928546.

[72] Marco C Campi and Simone Garatti. Wait-and-judge scenario optimization. *Mathematical Programming*, 167(1):155–189, 2018. doi:10.1007/s10107-016-1056-9.

[73] Marco C Campi and Algo Caré. Random convex programs with l$_1$-regularization: sparsity and generalization. *SIAM Journal on Control and Optimization*, 51(5):3532–3557, 2013. doi:10.1137/110856204.

[74] S. Garatti and M. C. Campi. Complexity-based modulation of the data-set in scenario optimization. In *2019 18th European Control Conference (ECC)*, volume, 1386–1391. June 2019. doi:10.23919/ECC.2019.8796160.

[75] Marco Claudio Campi, Simone Garatti, and Federico Alessandro Ramponi. A general scenario theory for nonconvex optimization and decision making. *IEEE Transactions on Automatic Control*, 63(12):4067–4078, 2018. doi:10.1109/TAC.2018.2808446.

[76] Edoardo Patelli, Matteo Broggi, Silvia Tolo, and Jonathan Sadeghi. Cossan software: a multidisciplinary and collaborative software for uncertainty quantification. In *2nd ECCOMAS Thematic Conference on Uncertainty Quantification in Computational Sciences and Engineering*, number UNCECOMP 2017. 2017. doi:10.7712/120217.5364.16982.

[77] Jonathan Sadeghi. PyIPM. *GitHub repository*, 2018. URL: https://github.com/JCSadeghi/PyIPM, doi:10.5281/zenodo.3349670.

[78] Sebastian Brandt, Matteo Broggi, Jan Hafele, Cristian Guillermo Gebhardt, Raimund Rolfes, and Michael Beer. Meta-models for fatigue damage estimation of offshore wind turbines jacket substructures. *Procedia engineering*, 199:1158–1163, 2017. doi:10.1016/j.proeng.2017.09.292.

[79] Francisco Alejandro Diaz De la O. *Gaussian process emulators for the analysis of complex models in engineering*. PhD thesis, Swansea University, 2011.

[80] Peter I Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

[81] Radford M Neal. *Bayesian learning for neural networks*. Volume 118. Springer Science & Business Media, 2012. doi:10.1007/978-1-4612-0745-0.

[82] James Hensman, Nicolò Fusi, and Neil D Lawrence. Gaussian processes for big data. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*, 282–290. AUAI Press, 2013.

[83] Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh. Neural processes. *arXiv preprint arXiv:1807.01622*, 2018.

[84] S. M. Ali Eslami, Danilo Jimenez Rezende, Frederic Besse, Fabio Viola, Ari S Morcos, Marta Garnelo, Avraham Ruderman, Andrei A Rusu, Ivo Danihelka, Karol Gregor, and others. Neural scene representation and rendering. *Science*, 360(6394):1204–1210, 2018. doi:10.1126/science.aar6170.

[85] Roland Schöbi. *Surrogate models for uncertainty quantification in the context of imprecise probability modelling*. PhD thesis, ETH Zurich, 2017.

[86] C Raj Sundararajan. *Probabilistic structural mechanics handbook: theory and industrial applications*. Springer Science & Business Media, 2012. doi:10.1007/978-1-4615-1771-9.

[87] Yarin Gal. *Uncertainty in deep learning*. PhD thesis, PhD thesis, University of Cambridge, 2016.

[88] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, 2672–2680. 2014.

[89] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[90] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[91] Diederik P Kingma and Jimmy Ba. Adam: a method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[92] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[93] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: representing model uncertainty in deep learning. In *international conference on machine learning*, 1050–1059. 2016.

[94] Yarin Gal, Jiri Hron, and Alex Kendall. Concrete dropout. In *Advances in Neural Information Processing Systems*, 3581–3590. 2017.

[95] Yarin Gal. Heteroscedastic dropout uncertainty. https://github.com/yaringal/HeteroscedasticDropoutUncertainty, 2016.

[96] V. N. Vapnik and A. Ya. Chervonenkis. *On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities*, pages 11–30. Springer International Publishing, Cham, 2015. doi:10.1007/978-3-319-21852-6_3.

[97] Isaac Elishakoff and Makoto Ohsaki. *Optimization and anti-optimization of structures under uncertainty*. World Scientific, 2010. doi:10.1142/p678.

[98] Yakov Ben-Haim. Convex models of uncertainty: applications and implications. *Erkenntnis*, 41(2):139–156, 1994. doi:10.1007/BF01128824.

[99] Diego A Alvarez. *Infinite random sets and applications in uncertainty analysis*. PhD thesis, Ph. D. Thesis, Leopold-Franzens Universität Innsbruck, Innsbruck, Austria, 2007.

[100] Thomas Fetz and Michael Oberguggenberger. Imprecise random variables, random sets, and monte carlo simulation. *International Journal of Approximate Reasoning*, 78:252–264, 2016. doi:10.1016/j.ijar.2016.06.012.

[101] Zhiping Qiu, Di Yang, and Isaac Elishakoff. Probabilistic interval reliability of structural systems. *International Journal of Solids and Structures*, 45(10):2850–2860, 2008. doi:10.1016/j.ijsolstr.2008.01.005.

[102] C Jiang, J Zheng, and X Han. Probability-interval hybrid uncertainty analysis for structures with both aleatory and epistemic uncertainties: a review. *Structural and Multidisciplinary Optimization*, pages 1–18, 2017. doi:10.1007/s00158-017-1864-4.

[103] Arne Decadt, Gert de Cooman, and Jasper De Bock. Monte carlo estimation for imprecise probabilities: basic properties. *arXiv preprint arXiv:1905.09301*, 2019.

[104] Roland Schöbi and Bruno Sudret. Structural reliability analysis for p-boxes using multi-level meta-models. *Probabilistic Engineering Mechanics*, 48:27–38, 2017. doi:10.1016/j.probengmech.2017.04.001.

[105] Jonathan Sadeghi, Marco de Angelis, and Edoardo Patelli. Robust propagation of probability boxes by interval predictor models. *Structural Safety*, 82:101889, 2020.

[106] Robert E Melchers and André T Beck. *Structural reliability analysis and prediction*. John Wiley & Sons, 2018.

[107] Nawal K. Prinja, Azeezat Ogunbadejo, Jonathan Sadeghi, and Edoardo Patelli. Structural reliability of pre-stressed concrete containments. *Nuclear Engineering and Design*, December 2016. doi:10.1016/j.nucengdes.2016.11.036.

[108] Zygmund William Birnbaum. On the importance of different components in a multicomponent system. Technical Report, Washington Univ Seattle Lab of Statistical Research, 1968.

[109] Andrea Saltelli, Stefano Tarantola, Francesca Campolongo, and Marco Ratto. *Sensitivity Analysis in Practice: A Guide to Assessing Scientific Models*. John Wiley & Sons, 2004. doi:10.1002/0470870958.

[110] Edoardo Patelli, Helmut J Pradlwarter, and Gerhart I Schuëller. Global sensitivity of structural variability by random sampling. *Computer Physics Communications*, 181(12):2072–2081, 2010. doi:10.1016/j.cpc.2010.08.007.

[111] Stefano Tarantola, Debora Gatelli, and Thierry Alex Mara. Random balance designs for the estimation of first order global sensitivity indices. *Reliability Engineering & System Safety*, 91(6):717–727, 2006. doi:10.1016/j.ress.2005.06.003.

[112] Ronald L. Iman. *Latin Hypercube Sampling*. New York: John Wiley & Sons Ltd, 2014. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118445112.stat03803, doi:10.1002/9781118445112.stat03803.

[113] Marco de Angelis, Edoardo Patelli, and Michael Beer. Advanced line sampling for efficient robust reliability analysis. *Structural safety*, 52:170–182, 2015. doi:10.1016/j.strusafe.2014.10.002.

[114] Gerhart Iwo Schuëller and Reinhard Stix. A critical appraisal of methods to determine failure probabilities. *Structural Safety*, 4(4):293–309, 1987. doi:10.1016/0167-4730(87)90004-X.

[115] Siu-Kui Au and James L Beck. Estimation of small failure probabilities in high dimensions by subset simulation. *Probabilistic engineering mechanics*, 16(4):263–277, 2001. doi:10.1016/S0266-8920(01)00019-4.

[116] Karl Breitung. The geometry of limit state function graphs and subset simulation: counterexamples. *Reliability Engineering & System Safety*, 182:98–106, 2019. doi:10.1016/j.ress.2018.10.008.

[117] Christian G Bucher and U Bourgund. A fast and efficient response surface approach for structural reliability problems. *Structural safety*, 7(1):57–66, 1990. doi:10.1016/0167-4730(90)90012-E.

[118] Claudio M Rocco and José Alí Moreno. Fast monte carlo reliability evaluation using support vector machine. *Reliability Engineering & System Safety*, 76(3):237–243, 2002. doi:10.1016/S0951-8320(02)00015-7.

[119] Marc Berveiller, Bruno Sudret, and Maurice Lemaire. Stochastic finite element: a non intrusive approach by regression. *European Journal of Computational Mechanics/Revue Européenne de Mécanique Numérique*, 15(1-3):81–92, 2006. doi:10.3166/remn.15.81-92.

[120] Shaowen Shao and Yoshisada Murotsu. Structural reliability analysis using a neural network. *JSME International Journal Series A Solid Mechanics and Material Engineering*, 40(3):242–246, 1997. doi:10.1299/jsmea.40.242.

[121] Irfan Kaymaz. Application of kriging method to structural reliability problems. *Structural Safety*, 27(2):133–151, 2005. doi:10.1016/j.strusafe.2004.09.001.

[122] Bruno Sudret. Global sensitivity analysis using polynomial chaos expansions. *Reliability engineering & system safety*, 93(7):964–979, 2008. doi:10.1016/j.ress.2007.04.002.

[123] Matthias Faes, Jonathan Sadeghi, Matteo Broggi, Marco De Angelis, Edoardo Patelli, Michael Beer, and David Moens. On the robust estimation of small failure probabilities for strong non-linear models. *ASCE-ASME Journal of Risk and Uncertainty in Engineering Systems Part B: Mechanical Engineering*, 2019. URL: https://www.researchgate.net/publication/330765220_On_the_robust_estimation_of_small_failure_probabilities_for_strong_non-linear_models.

[124] Bruno Sudret. Meta-models for structural reliability and uncertainty quantification. In *"Asian-Pacific Symposium on Structural Reliability and its Applications"*, 1–24. 2012.

[125] B Echard, N Gayton, and M Lemaire. Ak-mcs: an active learning reliability method combining kriging and monte carlo simulation. *Structural Safety*, 33(2):145–154, 2011. doi:10.1016/j.strusafe.2011.01.002.

[126] Barry J Goodno and James M Gere. *Mechanics of Materials*. Cengage Learning, 2016.

[127] Edoardo Patelli, Silvia Tolo, Hindolo George-Williams, Jonathan Sadeghi, Roberto Rocchetta, Marco de Angelis, and Matteo Broggi. Opencossan 2.0: an efficient computational toolbox for risk, reliability and resilience analysis. In *Proceedings of the joint ASCE ICVRAM ISUMA UNCERTAINTIES conference*. 2018. URL: http://icvramisuma2018.org/cd/web/PDF/ICVRAMISUMA2018-0022.PDF.

[128] Jonathan Sadeghi. *Uncertainty Modelling for Scarce and Imprecise Data in Engineering Applications*. PhD thesis, University of Liverpool, 2020. doi:10.17638/03089368.

[129] Wellison JS Gomes. Structural reliability analysis using adaptive artificial neural networks. *ASCE-ASME Journal of Risk and Uncertainty in Engineering Systems, Part B: Mechanical Engineering*, 2019. doi:10.1115/1.4044040.

[130] US Nuclear Regulatory Commission. Guidance on the treatment of uncertainties associated with pras in risk-informed decision making. *NUREG-1855*, March 2009.

[131] US Nuclear Regulatory Commission. Probabilistic risk assessment and regulatory decision making: some frequently asked questions. *NUREG-2201*, September 2016.

[132] James Berger. *The robust Bayesian viewpoint (with discussion). Robustness of Bayesian Analysis*. Amsterdam: North Holland, 1984.

[133] James O Berger, Elías Moreno, Luis Raul Pericchi, M Jesús Bayarri, José M Bernardo, Juan A Cano, Julián De la Horra, Jacinto Martín, David Ríos-Insúa, Bruno Betrò, and others. An overview of robust Bayesian analysis. *Test*, 3(1):5–124, 1994. doi:10.1007/BF02562676.