TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Civil Engineering Institute for Structural Analysis

Project Work

IMPROVEMENTS FOR ENGINEERING ANALYSIS WITH POLYMORPHIC UNCERTAINTY

Petromichelakis Ioannis

Project Work

Improvements for engineering analysis with polymorphic uncertainty

Petromichelakis Ioannis

Supervised by:
Prof. Dr.-Ing. Wolfgang Graf

and:
Dipl.-Ing. Marco Götz

submitted on March 21, 2013

ABSTRACT

The present project work is devoted to present improvements in engineering uncertainty analysis with further objective, the accession in the efficiency of the existing uncertainty analysis tool 'Winfuz'. The task of calculating small failure probabilities, common in civil engineering reliability problems including uncertainty, is dealt by means of overcoming the deficiency of the direct Monte-Carlo simulation in sampling rare events. The *Subset Sampling* method, which adopts the *Markov Chain Monte Carlo simulation* method, is implemented and proved to increase the efficiency of the stochastic solution, with decreasing computational effort. Moreover, in order to achieve realism and efficiency in the uncertainty analysis, a multivariate global optimization process is required. Hence, the existing evolutionary optimization subroutine 'evolu' is analysed and upgraded.

CONTENTS

1 INTRODUCTION

1.1 MOTIVATION AND OBJECTIVES

Besides the deterministic and semi-probabilistic methods for safety assessment, widely introduced in the standards, there is an increasing tendency in the application of probabilistic solutions (see section 1.2) according to the first and the second order reliability methods (FORM and SORM). Thus, it is required a structural analysis that matches suitably the input data with the computational models. Both the data and the models contain various types of uncertainty (see section 1.3). In order to assess the analysis realistically, these uncertainties should be accounted for. For that reason, different computational models (see section 1.4), that consider the different types of uncertainty, have been developed. In addition to that, special kinds of analyses have been introduced, that adopt such models and treat them in a computational manner in order to yield realistic results. A non-commercial program capable of processing *fuzzy analysis*, *stochastic analysis* and *fuzzy stochastic analysis* [17] is 'Winfuz', part of which engages the present project work.

Concerning the stochastic analysis, the drawback of Monte-Carlo simulation in calculating small failure probabilities was to be confronted. For that reason, the *subset sampling* method introduced in [1] was applied. Subset sampling, assumes the failure probability as a product of smaller conditional probabilities and uses the *Markov Chain Monte Carlo* simulation method (MCMC) to calculate them (see chapter 2). Markov Chain Monte Carlo simulation, is a powerful simulation method that is appropriate to calculate conditional probabilities, a very important aspect in modern reliability assessment. After the application of subset sampling, the convergence and the efficiency of the algorithm is tested in contrast to the direct Monte Carlo algorithm, by means of case studies. Finally, it should be mentioned that the subset sampling algorithm was designed, developed and tested using the R programming language and the translation into Fortran, as well as the implementation in 'Winfuz' was accomplished subsequently.

Additionally, the fuzzy analysis is based on the α-*level optimization* technique, which requires a reliable multivariate, global optimization subroutine (see chapter 3). Due to the fact that multivariate optimization is a common problem in applied mathematics research, new developments are often proposed and existing software needs updating until a reliable method with global applicability is invented. For improvement reasons, in chapter 3, a modification in the geometrical properties of the optimum search procedure is proposed. First, the subroutine was studied analytically, the proposed modifications were assigned and finally, the efficiency was tested by means of test functions.

1.2 CAUSALITY VERSUS RANDOMNESS

Causality is the relationship between an event (the cause) and a second event (the effect), where the second event is understood as a consequence of the first. In other words, causality is the relationship between a set of factors (causes) and a phenomenon (the effect). Anything that influences an effect is a factor of that effect [22]. On the other hand, the concept of randomness suggests a non-order or non-coherence in a sequence of symbols or steps, such that there is no intelligible pattern or combination or predictability [5].

As a result, there is no conflict between causality and randomness or between determinism and probability, if we take into account the fact that scientific theories are not discoveries of the laws

of nature but rather inventions of the human mind [19]. Their consequences are presented in deterministic form, if we examine the outcome of one trial, and as probabilistic statements, if we are interested in averages of many trials.

Assuming the problem of calculating the deflection $\boldsymbol{\delta}$ of a cantilever beam with length L subjected to a force F on its free end (see figure 1.2), derived from the theory of elasticity, we have

$$\delta = \frac{F \cdot L^3}{3 \cdot E \cdot I} \tag{1.1}$$

where I is the moment of inertia, a geometric parameter, and E is the Young's modulus, a material parameter.

This seems to be an unqualified consequence of a causal law; however, this is not so. The result is approximate and can be given a probabilistic interpretation.

Indeed, (1.1) is not the solution of a real problem but of an idealized model in which we have neglected imperfections in geometry, exact material behaviour and other uncertainties in the values of I and E, as well as in F and in L. We must, therefore, accept (1.1) only with qualifications.

Suppose now that we have a background with numbered notches (see figure 1.2) and we want to know until which notch, the beam will be deflected. Because of the uncertainties in I and in E, we are not able to give a deterministic answer to that problem. We can, however, ask a different question: If we construct many beams, with nominally the same construction properties, and load them with the same force, what percentage of them will reach the i^{th} notch? This question no longer has a causal answer; it can only be given a random interpretation.

Thus, the same engineering problem can be subjected either to a deterministic or to a probabilistic analysis. It is possible to argue, that the problem is inherently deterministic because the beam has precise I and E even if we do not know them. If we did, we would know the exact deflection of the cantilever beam. Mathematical interpretations including uncertainty are, therefore, necessary because of our ignorance [19].

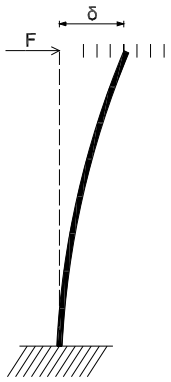


Figure 1.1: Cantilever beam subjected to lateral loading

1.3 TYPES OF UNCERTAINTY

Uncertainty can be defined as knowledge incompleteness due to inherent deficiencies in acquired knowledge [2]. Or in a more mathematically strict way, uncertainty is the gradual assessment of the truth content of a postulation, which may be referred to the occurrence of a defined event [18]. It interferes in the analysis as *data uncertainty*, in materials, geometry, loads etc. and *model uncertainty*, through simplifications and generalizations during the calculation process. Concerning

the paradigm in section 1.2, data uncertainty refers to knowledge incompleteness in the parameters *I*, *E*, *L* and *F*, while model uncertainty refers to simplified assumptions and generalizations of the theory of elasticity, from which, equation (1.1) is derived.

Another classification of uncertainty, according to the source that produces it, distinguishes between *stochastic* and *non-stochastic* uncertainty. Accordingly, if an event, as a random result of a test, can be observed on an almost unlimited number of occasions under constant boundary conditions, this concerns *stochastic uncertainty*. This type of uncertainty is related to the characteristic *randomness* (see subsection 1.4.1) and it is being described and investigated by the methods of *probability analysis*. A basis of the application of such methods, is the validity of statistical laws for stochastic input parameters (see appendix A.1). Probability distributions are used to model system parameters and input variables that are uncertain in the random sense.

More recently, a realization of a non-stochastic, *cognitive* or *epistemic* type of uncertainty was established. This type of uncertainty consists of two main components, the *informal uncertainty* and the *lexical uncertainty*. Informal uncertainty arises when the boundary conditions of the random test are not constant or the number of observations are only available to a limited extend. Lexical uncertainty, is related to the semantic vagueness in defining system parameters and can be expressed as linguistic variables representing quantified verbal postulations. Apparently, informal and lexical uncertainty, have different influencing mechanisms but they have the same source. They both arise from mind-based abstractions of reality with lack of precision, and described by the characteristic *fuzziness* (see subsection 1.4.2). As a result, fuzzy set theory and possibility theory are appropriate to consider this type of uncertainty, as well as uniform and triangular probability distributions, the so called *membership functions* are used to model it [24].

The uncertainty characteristic *fuzzy randomness* (see subsection 1.4.3), arises when the statistical description of a random variable is informally and/or lexically uncertain. The theory of fuzzy random variables, as a combination of fuzzy set theory and probabilistic methods, is appropriate to take into account this characteristic. Randomness, fuzziness and fuzzy randomness, described in section 1.4, may occur in the form of both data and model uncertainty.

1.4 MODELLING OF UNCERTAINTY

As a result of the distinction between different types of uncertainty, there are different mathematical models to describe them. Uncertain parameters of a problem, that are related with the characteristic randomness, are described using *random variables* (see subsection 1.4.1). On the other hand, an uncertain parameter related to the characteristic fuzziness, is best described using *fuzzy variables* (see subsection 1.4.2). Finally, *fuzzy random variable* (see subsection 1.4.3) is the mathematical model, which assumes that an uncertain parameter may be described by a random variable, the definition parameters of which, are fuzzy variables.

1.4.1 Randomness

random variables

Random variables are used to model uncertain parameters for which, sufficient statistical data is available and the reproduction conditions are constant. Random variable or stochastic variable is a variable whose value is subject to variations due to chance. As opposed to other mathematical variables, a random variable conceptually does not have a single, fixed value (even if unknown); rather, it can take on a set of possible different values, each with an associated probability. A random variable is usually denoted by boldface $\mathbf{X}$ and the values x, this random variable can take,

are called realizations of $\mathbf{X}$. This gives a footing for the formal definition of a random variable derived from [19].

Definition 1.1. ***random variable*** *Given an experiment specified by the space $S \subseteq \mathbb{R}$, a random variable $\boldsymbol{X}$ is a process of assigning a number $\boldsymbol{X}(\zeta)$ to every outcome $\zeta \in S$ of the experiment. The resulting function must satisfy the following two conditions but is otherwise arbitrary:*

1. *The set $\{\boldsymbol{X} \leq x\}$ is an event for every x.*
2. *The probabilities of the events $\{\boldsymbol{X} = \infty\}$ and $\{\boldsymbol{X} = -\infty\}$ equal 0:*
   . $P\{\boldsymbol{X} = \infty\} = 0 \qquad P\{\boldsymbol{X} = -\infty\} = 0$

For a random variable, it may be defined, *cumulative distribution function* (CDF)

$$F(x) = P\{\mathbf{X} \leq x\}, \tag{1.2}$$

which is the probability that the random variable $\mathbf{X}$ will be less or equal to the value x, and *probability density function* (PDF)

$$f(x) = \frac{dF(x)}{dx}. \tag{1.3}$$

There are various specific distribution functions with their associate density functions such as, uniform distribution and normal distribution, but it is beyond the scope of the present project to describe them. They can be found in every comprehensive probability text such as [2] and [19]. The following are only abbreviations to denote some commonly used distribution functions encountered in the examples of the present project.

- $\mathbf{X} \sim U(\alpha, \beta)$ means that the random variable $\mathbf{X}$ is distributed according to the *uniform distribution* defined by the lower bound α and the upper bound β.
- $\mathbf{X} \sim N(\mu, \sigma)$ means that the random variable $\mathbf{X}$ is distributed according to the *normal distribution* defined by the mean value μ and the standard deviation σ.
- $\mathbf{X} \sim LN(\mu, \sigma, x_0)$ means that the random variable $\mathbf{X}$ is distributed according to the *log-normal distribution* defined by the moments *m* (moment of 1^{st} order, mean value) and *sd* (square root of the central moment of 2^{nd} order, standard deviation) and the location parameter x_0[1].
- $\mathbf{X} \sim Be(\alpha, \beta)$ means that the random variable $\mathbf{X}$ is distributed according to the *beta distribution* defined by the two shape parameters α and β.

random numbers

An important aspect in the numerical treatment of random variables, is the generation of random numbers. It is chosen to discuss about random numbers here, due to their relation with randomness. In a computer simulation process, random numbers are normalized (so that they belong in the range [0, 1]) real values, that are uniformly distributed. The importance of uniform random numbers is that they can be transformed into real values that follow any distribution of interest. Therefore, they are the basis for most simulation based processes.

It is self-evident, that a set of random numbers, in order to be random, should satisfy the condition of no serial correlation. This condition is not possible to be absolutely fulfilled in a random number generator algorithm, because the word random, implies the phrase 'without pattern' but an algorithm is a calculation procedure that follows a specific pattern. For that reason, a variety of clever

[1] Often, the log-normal distribution is defined by the parameters μ_u and σ_u which are not equal to its moments but they are related to them by $\mu_u = \ln[1 + (\sigma/(\mu - x_0)^2]$ and $\sigma_u = \ln(\mu - x_0) - \sigma^2/2$. The parameter x_0 is optional in both cases, if not defined, equals to 0.

algorithms have been developed which generate sequences of numbers which pass every statistical test used to distinguish random sequences from those containing some pattern or internal order [26]. This type of random numbers are called *pseudo-* or *quasi- random numbers* (PRNGs).

In these generators, often referred as *arithmetic generators*, a random number is obtained based on fixed mathematical equations and a previous value, the so called *seed*. Arithmetic generators import a seed (or seeds) and they result in a stream of random values. The same stream of random values will be generated if the same seed is used again. This property of repeatability, should be avoided in comparative studies of design alternatives of a system.

There are various arithmetic random number generators, such as, midsquare, linear congruential generator, mixed generators, multiplicative generators, general congruences, composite generators, and Tausworthe generators. All of these generators, are based on the principle of repeating a recursive process starting with a seed and obtain N random values (for N runs), using fixed mathematical equations that use the generated value to compute the next one. In all such recursive processes, the period is of concern. The *period* is defined as the number of generated random values after the stream of values starts to repeat itself. It thus desirable, to have large periods, much higher than the required number of simulation runs.

Pseudo-random number generators, is the most convenient way to produce sequences of random numbers, but it is not the only one. There are other generators, such as the *hardware random number generators* (T(rue)RNGs), which use a physical process, rather than a computer program to obtain a random value. Moreover, there are *random number servers*, such as [26] and [25], that provide sequences of true random numbers, taking advantage of the inherent uncertainty in the quantum mechanical laws of nature.

1.4.2 Fuzziness

The uncertainty description of parameters, is strongly dependent on the expert judgement or on samples which are not validated statistically. In this case, the description by the uncertainty model fuzziness is recommended [24]. The advantage of that model, is that it takes into account both objective and subjective information.

crisp set and fuzzy set

In classical set theory[2] the membership of elements in relation to a set is assessed in binary terms according to a crisp condition [17]. This means that an element either belongs or does not belong to the set, the boundary of the set is crisp, and the set is called *crisp set*, see figure 1.4.2.

As a further development of classical set theory, fuzzy set theory permits the gradual assessment of the membership of elements in relation to a set, see figure 1.4.2. This is described with the aid of a membership function. A definition of a fuzzy set, is derived from [17].

Definition 1.2. ***fuzzy set*** *If* $\boldsymbol{X}$ *represents a fundamental set and x are the elements of this set, to be assessed according to an (lexically or informally, see subsection 1.3) uncertain postulation and assigned to a subset A of* $\boldsymbol{X}$*, the set*

$$\tilde{A} = \{(x, \mu_A(x)) | x \in \boldsymbol{X}\} \qquad \mu_A(x) \geq 0 \forall x \in \boldsymbol{X} \tag{1.4}$$

is referred to as fuzzy set on $\boldsymbol{X}$*.* $\mu_A(x)$ *is the membership function (characteristic function) of the fuzzy set* $\tilde{A}$*. The fuzzy set* $\tilde{A}$ *is also referred to as uncertain set* $\tilde{A}$ *and as fuzzy variable or fuzzy value* $\tilde{x}$*.*

[2] Set is a combination of particular, well-distinguishable objects (which are called 'elements' of the set) to an ensemble.

Fuzzy variables may be utilized to describe the imprecision of structural parameters directly, as well as to specify the parameters of fuzzy random variables. The membership function, usually is normalized so that, has a peak value of 1.

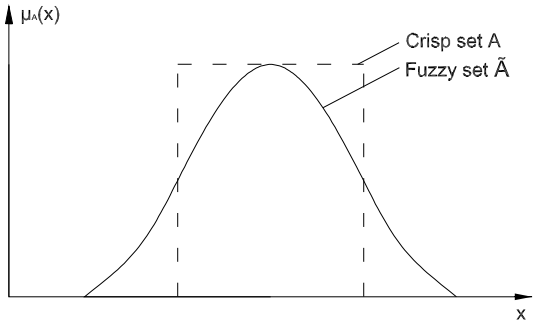


Figure 1.2: Comparison of crisp set and fuzzy set

linguistic variables

The fundamental set **X** referred in the definition of a fuzzy set, can be constructed using physical or linguistic variables. For the purpose of structural analysis, however, numerical values are required. It is therefore necessary to transform linguistic variables into numerical values [18]. In the example described in section 1.2, it is possible to adopt e.g. the linguistic variable, force F. An assessment of the force F is carried out by assigning degrees (linguistic values) high, medium and low in combination with a selection of modifiers such as extreme and very. These degrees must then be referred to a numerical scale, which e.g. represents a sustainable loading of the cantilever beam of the figure 1.2. This approach is illustrated in figure 1.4.2.

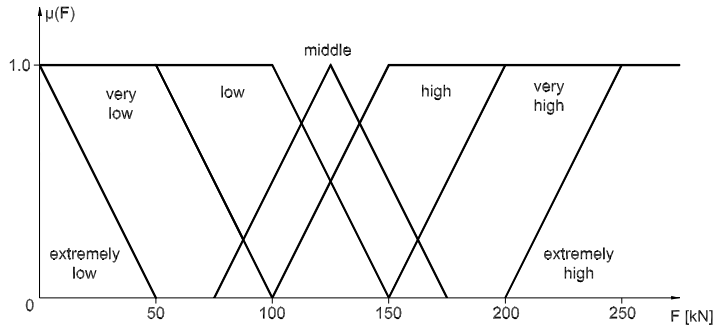


Figure 1.3: Assessment of the linguistic variable, force F

Further mathematical definitions related to the fuzzy set theory

Definition 1.3. ***support of a fuzzy set*** *The support $S(\tilde{A})$ of a fuzzy set $\tilde{A}$ is a crisp set. This contains the elements*

$$S(\tilde{A}) = \{x \in \boldsymbol{X} | \mu_A(x) > 0\} \tag{1.5}$$

[18].

Definition 1.4. ***convex fuzzy set*** *A fuzzy set $\tilde{A}$ is referred to as convex if its membership function $\mu_A(x)$ monotonically decreases on each side of the maximum value, i.e. when the following applies*

$$\mu_A(x_2) \geq \min[\mu_A(x_1), \mu_A(x_3)] \qquad \forall x_1, x_2, x_3 \in \boldsymbol{X} \quad with \quad x_1 \leq x_2 \leq x_3 \tag{1.6}$$

[18].

Definition 1.5. ***fuzzy number*** *A fuzzy number $\tilde{\alpha}$ is a convex, normalized fuzzy set $\tilde{A} \subseteq \mathbb{R}$, whose membership function is at least segmentally continuous and has the functional value $\mu_A(x) = 1$ at precisely one of the x values. This point x is referred to as the mean value of the fuzzy number. Fuzzy numbers with a linear membership function are referred to as fuzzy triangular numbers. These may be represented with the aid of the number triplet $\tilde{\alpha} = \langle x_1, x_2, x_3 \rangle$, whereby x_1 and x_3 are the interval bounds of the support and x_2 is the mean value with the functional value $\mu_A(x_2) = 1$ [18] .*

Definition 1.6. ***α-level set*** *From the fuzzy set $\tilde{A}$ the crisp sets*

$$A_{\alpha_k} = \{x \in \boldsymbol{X} | \mu_A(x) \geq \alpha_k\} \tag{1.7}$$

may be extracted for real numbers $\alpha_k \in [0, 1]$. These crisp sets are called α-level sets. All α-level sets A_{α_k} are crisp subsets of the support $S(\tilde{A})$. For several α-level sets of the same fuzzy set $\tilde{A}$ the following holds,

$$A_{\alpha_k} \subseteq A_{\alpha_i} \qquad \forall \alpha_i, \alpha_k \in [0, 1] \quad \textit{with} \quad \alpha_i \leq \alpha_k. \tag{1.8}$$

If the fuzzy set $\tilde{A}$ is convex, each α-level set A_{α_k} is an interval $[x_{\alpha_k l}, x_{\alpha_k r}]$ in which

$$x_{\alpha_k l} = \min[x \in \boldsymbol{X} | \mu_A(x) \geq \alpha_k] \tag{1.9}$$

$$x_{\alpha_k l} = \max[x \in \boldsymbol{X} | \mu_A(x) \geq \alpha_k] \tag{1.10}$$

[18].

This last definition of the α-level set, is very important in the procedures of fuzzy and fuzzy stochastic analysis because it provides the basis for the α-level optimization. The α-level optimization, is a computational tool used in the process of combining fuzzy input data to yield the fuzzy result and is introduced in subsection 2.6.1 and analytically described in chapter 3.

1.4.3 Fuzzy randomness

In the case of uncertain parameters for which, the reproduction conditions vary during the period of observation or expert knowledge completes the (probably insufficient) statistical data, an adequate uncertainty quantification succeeds with *fuzzy random variables* [24]. The uncertain data model fuzzy randomness, which is a generalized model that arises from the combination of stochastic and non-stochastic characteristics, provides the basis of the theory of *fuzzy random variables*. A fuzzy random variable $\tilde{X}$ is defined as the fuzzy set of their originals, whereby each original is a real-valued random variable X. A formal definition of a fuzzy random variable which is based on [21] follows.

Definition 1.7. ***fuzzy random variable*** *In accordance with probability theory, the space of the random elementary events Ω is introduced. Instead of a real realization, a fuzzy realization of the form $\tilde{x}(\omega) = \tilde{x}$ is now assigned to each elementary event $\omega \in \Omega$. Where $\tilde{x}$ is an element of the set $\boldsymbol{F}(\mathbb{R})$ of all fuzzy variables on $\mathbb{R}$. Accordingly from the definition of a fuzzy variable in subsection 1.4.2, a fuzzy random variable $\tilde{X}$ is a fuzzy result of the mapping given by*

$$\tilde{X} : \Omega \mapsto \boldsymbol{F}(\mathbb{R}) \tag{1.11}$$

[18],[24].

Based on this formal definition, a fuzzy random variable is described by its *fuzzy cumulative distribution function* (fuzzy CDF) $\tilde{F}(x)$. The function $\tilde{F}(x)$ is defined as the set of all real-valued cumulative distribution functions $F(x)$ which are gradually assessed by the membership $\mu_F(F(x))$. $F(x)$ is the CDF of the random variable X and as a result, each realization x_i of X is assigned to a fuzzy (functional) value $\tilde{F}(x_i)$ (see figure 1.4.3). Thus, $\tilde{F}(x)$ is a *fuzzy function*, definitions of which can be found in [18] and in [24]. It may also be defined accordingly, a *fuzzy probability density function*

$$\tilde{f}(x) = \{(f(x), \mu_f(f(x)))|f \in \mathbf{f}\} \qquad \mu_f(f(x)) \geq 0 \forall f \in \mathbf{f}. \tag{1.12}$$

Where **f** represents the set of all probability density functions defined on X.

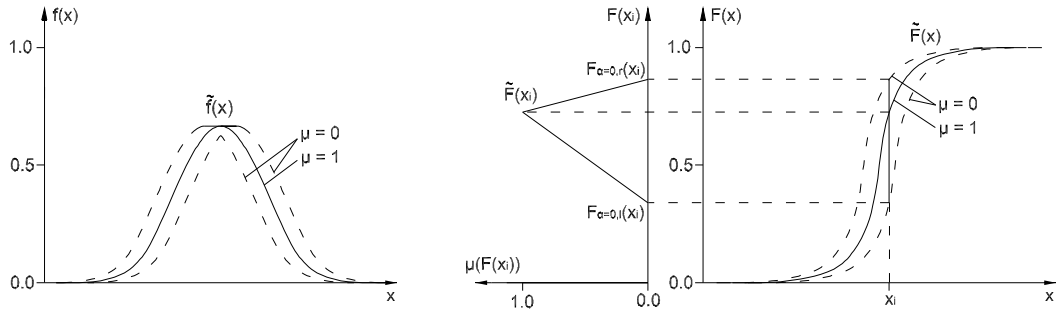


Figure 1.4: Fuzzy probability density and fuzzy cumulative distribution function

In engineering structural problems, a stochastically and non-stochastically uncertain parameter, may by assumed as an elementary event ω in the space of the random elementary events Ω. As a result, the modelling of such an uncertain parameter as a fuzzy random variable, may be understood by the following simplified approach. Realizations of the given uncertain parameter, are distributed according to a specific probability density function (or cumulative distribution function) the parameters of which, e.g. the mean μ and the standard deviation σ are fuzzy variables, $\tilde{\mu}$ and $\tilde{\sigma}$.

2 STOCHASTIC METHODS FOR THE CALCULATION OF FAILURE PROBABILITIES

In this chapter, the development of methods to be used as an improved alternative of the ordinary Monte Carlo Simulation (MCS) was attempted. The main goal was to improve the efficiency of the stochastic solution of the existing uncertainty analysis tool 'Winfuz' for the usual problem of calculating small probabilities of failure, where the MCS seems to be computationally expensive.

2.1 INTRODUCTION

Ordinary Monte Carlo method, is a very useful tool which provides the main scheme of any simulation based process, but it appears to be inefficient in some reliability problems. More specifically, in structural reliability applications, the main target is to calculate probabilities of failure P_f, often with magnitude of order $10^{-5} - 10^{-6}$. As a result, in a simulation procedure, MCS, demands a large number of simulation runs in order to achieve acceptable accuracy. The capability of the method in exploring the failure region, is limited for a reasonable sample size. It is noteworthy that MCS obeys the so called *square root law*: statistical accuracy is inversely proportional to the square root of the sample size [7]. Each additional significant figure, a tenfold increase in accuracy, requires a hundredfold increase in the sample size.

In order to compensate this drawback, a more advanced simulation method called *subset sampling* or *subset simulation* is used. This method, was introduced by [1] and is based on the idea of subdivision of the failure event into a sequence of partial failure events, the subsets. The numerical efficient sampling within the subset is realized with the aid of the Markov Chain Monte Carlo simulation (MCMC).

In the next sections, the theoretical background of the Markov Chain Monte Carlo simulation and the subset sampling method, is being presented. Finally, the developed algorithm for the application of the subset sampling is demonstrated, and its efficiency is being discussed. It should be noted here, that for the development of the algorithm the language 'R' was used for convenience and then it was translated to Fortran, and applied on the existing program for uncertainty analysis 'Winfuz'.

2.2 DIRECT MONTE CARLO SIMULATION (MCS)

The principle behind *Monte Carlo simulation techniques*, is to develop a computer-based analytical model that predicts the behaviour of a system [2]. This model, is capable to assess the reliability of structures, by calculating the probability of failure of the system-structure, through the robust numerical approximation of integrals which are impossible to calculate using analytical methods [15]. Probability of failure, is assumed the probability of exceedance of some specified limit state value by a performance function $g(\boldsymbol{\theta})$, which expresses the behaviour of the system-structure. Where $\boldsymbol{\theta}$, is the vector of the input uncertain variables of the problem. In the case of stochastic structural

analysis, $\boldsymbol{\theta}$ can be material strengths, geometric magnitudes, loads, etc. and g is a measure of damage i.e. stresses, strains, deflections, drifts. For the approximation of the failure probability, it holds

$$P_F = P(\boldsymbol{\theta}|g(\boldsymbol{\theta}) \leq 0) = \int_{\boldsymbol{\theta}|g(\boldsymbol{\theta})\leq 0} f(\boldsymbol{\theta})d\boldsymbol{\theta} \tag{2.1}$$

where $f(\boldsymbol{\theta})$, is the joint probability density function[1].

The failure probability can be computed by means of an indicator function

$$\mathbb{I}_F(\boldsymbol{\theta}) = \begin{cases} 1 & \text{if } \boldsymbol{\theta} \in F \\ 0 & \text{if } \boldsymbol{\theta} \notin F \end{cases} \qquad \text{with } F = \{\boldsymbol{\theta}|g(\boldsymbol{\theta}) \leq 0\} \tag{2.2}$$

where F denotes the failure region. Monte Carlo simulation samples vectors $\boldsymbol{\theta}$ using the probability densities of each component and it calculates how many times $\boldsymbol{\theta}$ found in the failure region, using the interpretation of the integral

$$P_F = \int_{\boldsymbol{\theta}} \mathbb{I}_F(\boldsymbol{\theta}) f(\boldsymbol{\theta}) d\boldsymbol{\theta} = E(\mathbb{I}_F(\boldsymbol{\theta})) \tag{2.3}$$

for the estimation of the failure probability P_F. The application of the *Law of Large Numbers* (see appendix A.1), yields

$$E(\mathbb{I}_F(\boldsymbol{\theta})) = \lim_{N\to\infty} \frac{1}{N} \sum_{k=1}^{N} \mathbb{I}_F(\boldsymbol{\theta}) \tag{2.4}$$

Finally, for the evaluation of equation (2.4), the estimator $\tilde{P}_F$ is used,

$$\tilde{P}_F = \frac{1}{N} \sum_{k=1}^{N} \mathbb{I}_F(\boldsymbol{\theta}) \tag{2.5}$$

$\tilde{P}_F$ converges to the failure probability P_F as the number of samples N approaches infinity [15]. The expected value E and the variance Var of the estimated failure probability are

$$E(\tilde{P}_F) = P_F, \tag{2.6}$$

$$Var(\tilde{P}_F) = \frac{(1 - P_F)P_F}{N}. \tag{2.7}$$

The coefficient of variation is determined by

$$\delta_{P_F} = \frac{\sqrt{Var[\tilde{P}_F]}}{E[\tilde{P}_F]} = \sqrt{\frac{1 - P_F}{P_F \cdot N}}. \tag{2.8}$$

The coefficient of variation does not depend on the dimensionality of the random vector $\boldsymbol{\theta}$ [15]. The determination of small probabilities of failure P_F with an acceptable level of accuracy demands a large number of samples N. An overview about the required number of samples for different values of the failure probability P_F and the coefficient of variation δ_{P_F} derived from [15], is given indicatively in table 2.1.

Table 2.1: Required number of samples for Monte Carlo simulation

P_F	10^{-1}	10^{-2}	10^{-3}	10^{-4}
$\delta_{P_F} = 0.3$	$1 \cdot 10^2$	$1.1 \cdot 10^3$	$1.11 \cdot 10^4$	$1.111 \cdot 10^5$
$\delta_{P_F} = 0.1$	$9 \cdot 10^2$	$1 \cdot 10^4$	$1 \cdot 10^5$	$1 \cdot 10^6$
$\delta_{P_F} = 0.0$	$9 \cdot 10^4$	$1 \cdot 10^6$	$1 \cdot 10^7$	$1 \cdot 10^8$

[1]The joint probability density function of n discrete (the discrete case is explained for simplicity) random variables $\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_n$, equals to: $P(\mathbf{X}_1 = x_1, ..., \mathbf{X}_n = x_n) = P(\mathbf{X}_1 = x_1) \cdot P(\mathbf{X}_2 = x_2|\mathbf{X}_1 = x_1) \cdot P(\mathbf{X}_3 = x_3|\mathbf{X}_1 = x_1, \mathbf{X}_2 = x_2) \cdot ... \cdot P(\mathbf{X}_n = x_n|\mathbf{X}_1 = x_1, \mathbf{X}_2 = x_2, ..., \mathbf{X}_{n-1} = x_{n-1})$. As a result, if the random variables $\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_n$ are independent, their joint probability density function becomes: $P(\mathbf{X}_1 = x_1, ..., \mathbf{X}_n = x_n) = P(\mathbf{X}_1 = x_1) \cdot P(\mathbf{X}_2 = x_2) \cdot ... \cdot P(\mathbf{X}_n = x_n)$, where $P(\mathbf{X}_i = x_i)$ is the probability density function of the i-th random variable.

2.3 MARKOV CHAIN MONTE CARLO SIMULATION (MCMC)

Markov Chain Monte Carlo Simulation (MCMC) is a class of powerful simulation techniques for generating samples according to any given probability distribution, at least in the asymptotic sense as the number of samples increases [1].

2.3.1 Basic concept

Definition 2.1. ***Markov chain*** *A Markov chain is a sequence of random vectors* $\boldsymbol{X}_0, \boldsymbol{X}_1, ..., \boldsymbol{X}_k$ *in the state space* $\mathbb{S} \subseteq \mathbb{R}^n$*, where n is the size of the vectors, with the Markov property*

$$P(\boldsymbol{X}_{k+1} = \boldsymbol{s}_{k+1} | \boldsymbol{X}_k = \boldsymbol{s}_k, \boldsymbol{X}_{k-1} = \boldsymbol{s}_{k-1}, ..., \boldsymbol{X}_0 = \boldsymbol{s}_0) = P(\boldsymbol{X}_{k+1} = \boldsymbol{s}_{k+1} | \boldsymbol{X}_k = \boldsymbol{s}_k) \tag{2.9}$$

while $\boldsymbol{s}_0, ..., \boldsymbol{s}_k \in \mathbb{S}$*. The values* $0, 1, ..., k$ *form the index space* $\mathbb{I}$ *where* $\mathbb{I} \subseteq \mathbb{R}$*. The Markov property signifies that the probability to reach an arbitrary state* $\boldsymbol{s}_{k+1}$ *depends only on the state* $\boldsymbol{s}_k$ *and is independent from all former states. That is, the state of the system is important but the history of how we arrived at that state is not. The Markov chain is called homogeneous, if the probability for transition*

$$P(\boldsymbol{X}_{k+1} = s_{k+1} | \boldsymbol{X}_k = s_k)$$

is independent from the index k [15],[14].

The approach of generating samples from a specially designed Markov chain which are approximately distributed as an arbitrary given PDF is called Markov Chain Monte Carlo simulation. If the Markov chain is *ergodic*, the samples have a limiting stationary distribution which tends to the so called *target PDF* $\pi(\boldsymbol{\theta})$ as the length of the Markov chain increases. If the requirement of ergodicity has been met and the Markov chain has been constructed properly, the expected value for any quantity of interest $h(\boldsymbol{\theta})$ is

$$E(h) = \lim_{N \to \infty} \frac{1}{N} \sum_{k=1}^{N} h(\boldsymbol{\theta}_k) \tag{2.10}$$

Thus the estimator $\tilde{P}_F$ for the failure probability P_F can be expressed by

$$\tilde{P}_F = \tilde{E}(\mathbb{I}_F(\boldsymbol{\theta})) = \frac{1}{N} \sum_{k=1}^{N} \mathbb{I}_F(\boldsymbol{\theta}_k) \tag{2.11}$$

where $\boldsymbol{\theta}_k$ for $k = 1, ..., N$ are realizations of the Markov chain and $\mathbb{I}_F$ is an indicator function which equals to 1 for failure and to 0 for no failure [15].

Two noteworthy aspects in the MCMC simulation method, are the choice of the function for the transition from a state k to the next state $k + 1$, the so-called *proposal function* which is explained analytically in the subsection 2.3.4 and the choice of the first sample $\boldsymbol{\theta}_1$ which should be defined in advanced. A distinction between two cases for this sample is made. First: $\boldsymbol{\theta}_1$ is distributed exactly as the target PDF $\pi(\boldsymbol{\theta})$ and second: $\boldsymbol{\theta}_1$ is not distributed as $\pi(\boldsymbol{\theta})$. In the first, the subsequent samples $\boldsymbol{\theta}_2, \boldsymbol{\theta}_3, ...$ are also distributed as $\pi(\boldsymbol{\theta})$. In the second, under *mild regularity conditions*, the distribution of the subsequent samples tends to $\pi(\boldsymbol{\theta})$, $p(\boldsymbol{\theta}_N) \to \pi(\boldsymbol{\theta}_N)$ as $N \to \infty$ [1]. Proper utilization of the Markov chain samples can lead to better estimates of the quantity of interest, i.e. the failure probability.

Generally, the Markov chain samples are not independent and not identically distributed (i.i.d.), when the initial sample is not distributed as $\pi(\boldsymbol{\theta})$. In spite that fact, the samples can still be used for statistical averaging as if they where i.i.d. by virtue of the Law of Large Numbers (see appendix A.1) [1].

MCMC provides a convenient way for generating samples according to the conditional PDF $q(\boldsymbol{\theta}|F)$ given failure occurs. This is the reason of the significance of the method in reliability problems where the calculation of such probabilities is a main challenge. As the Markov chain develops, it explores the state space and gains information about the failure region. As a result, MCMC appears to be much more efficient than MCS in structural reliability.

2.3.2 Metropolis-Hastings Algorithm

The Metropolis Hastings algorithm is an algorithm presented by Metropolis in [16] and enhanced by Hastings in [11]. It generates Markov chain samples according to any given probability density function, the so called *target PDF* $\pi(\boldsymbol{\theta})$. The procedure for the transition from a the state k to the next state $k+1$ is explained below.

Repeat for k = 1,2,...:

1. Generate a 'candidate' state $\tilde{\boldsymbol{\theta}}_{k+1}$:

   (a) Simulate a 'pre-candidate' state $\boldsymbol{\xi}_{k+1}$ according to $p^*(\boldsymbol{\xi}_{k+1}|\boldsymbol{\theta}_k)$.

   (b) Compute the acceptance ratio:

   $$r_{k+1} = \frac{\pi(\boldsymbol{\xi}_{k+1})p^*(\boldsymbol{\theta}_k|\boldsymbol{\xi}_{k+1})}{\pi(\boldsymbol{\theta}_k)p^*(\boldsymbol{\xi}_{k+1}|\boldsymbol{\theta}_k)} \tag{2.12}$$

   (c) Set $\tilde{\boldsymbol{\theta}}_{k+1} = \boldsymbol{\xi}_{k+1}$ with probability $\min\{1, r_{k+1}\}$ and set $\tilde{\boldsymbol{\theta}}_{k+1} = \boldsymbol{\theta}_k$ with the remaining probability $1 - \min\{1, r_{k+1}\}$

2. Accept/reject $\tilde{\boldsymbol{\theta}}_{k+1}$: according to F:

   If $\tilde{\boldsymbol{\theta}}_{k+1} = \boldsymbol{\theta}_k$, increment k by 1 and go to Step 1. Otherwise, check the location of $\tilde{\boldsymbol{\theta}}_{k+1}$. If $\tilde{\boldsymbol{\theta}}_{k+1} \in F$, accept it as the next sample, i.e. $\boldsymbol{\theta}_{k+1} = \tilde{\boldsymbol{\theta}}_{k+1}$; otherwise reject it and take the current sample as the next sample, i.e. $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k$, then increment k by 1 and go to Step 1.

Where $p^*(\boldsymbol{\xi}|\boldsymbol{\theta})$, is the *proposal PDF* (see subsection 2.3.4) and it is a chosen n-dimensional joint PDF for $\boldsymbol{\xi}$ that depends on $\boldsymbol{\theta}$.

Concerning the algorithm, in the first step, the candidate state $\tilde{\boldsymbol{\theta}}_{k+1}$ is being generated in dependency to $\boldsymbol{\theta}_k$ and its distribution is related to the proposal PDF. Proposals with higher density values, according to the target PDF, are more likely to be accepted. In the second step, it is being checked whether the proposal lies in the failure region or not. If yes, it is being accepted otherwise the next state becomes the same with the current. With the addition of step 2. in the main Metropolis-Hastings algorithm (step 1.), the limiting stationary distribution of the Markov chain becomes $q(\boldsymbol{\theta}|F)$, which is the probability density of the samples, given failure occurs.

It should be noted here, that the main disadvantage of MCMC simulation methods, especially in high dimensions, is the so called *burn-in* procedure which follows after the generation of all the MCMC samples. Burn-in, refers to the retraction of the first samples that have been generated before the chain has reached its stationary limiting distribution. The reason for that retraction, is that the remaining samples will be more accurate because they exactly come from the PDF that the chain has converged to. Despite this drawback, it is explained in [7], that such a procedure is not necessary because the Markov chain will even so converge to its PDF, and thus a large number of iterations will produce accurate results.

2.3.3 Stationarity

A sequence $X_1, X_2, ...$ of random elements of some set is called a *stochastic process*. A stochastic process is *stationary*, if for every positive integer k the distribution of the k-tuple

$$(X_{n+1}, ..., X_{n+k})$$

does not depend on n. A Markov chain is stationary if it is a stationary stochastic process [14]. In a Markov chain, the conditional distribution $(X_{n+2}, ..., X_{n+k}|X_{n+1})$ does not depend on n. It follows that a Markov chain is stationary if and only if the marginal distribution of X_n does not depend on n [7]. Proofs of the stationarity of the Markov chains generated by MCMC, can be found in [1] and in [23].

As mentioned before, in the (general) case that the initial sample $\boldsymbol{\theta}_1$ is not distributed as the target PDF, it can be shown that the distribution of the subsequent samples will still tend to the target PDF, under some mild regularity conditions. If the requirement of convergence has been met, the Markov chain is *ergodic*. In practice, ergodicity is whether the Markov chain samples can meet sufficiently the failure region. This determines whether the estimate for the expectation of the quantity of interest obtained by averaging over the Markov chain samples is unbiased. For overcoming the problem of insufficient 'exploration' of the failure region, the method of *subset sampling* has been developed in section 2.4.

2.3.4 Proposal PDF

Technically the *proposal PDF* (or *transition function*, or *transition kernel*) introduces a small random alternation from the current state in order to generate the 'candidate' to become the next state. Thus, the proposal PDF is important for the accuracy and convergence of the algorithm and successful applications of MCMC rely on a proper choice of it. Some proposals provide better results than others. More details about the optimal choice of the proposal PDF can be found in literature, e.g. in [7] and [13]. Moreover, adaptive algorithms have been introduced in order to choose themselves an efficient proposal PDF.

An interesting, convenient and very common type of proposal PDF is a symmetric distribution function centred at the current sample, hence $p^*(\boldsymbol{\theta}|\boldsymbol{\xi}) = p^*(\boldsymbol{\xi}|\boldsymbol{\theta})$. Using such a type of proposal PDF, introduces a major simplification in the Metropolis - Hastings algorithm. The acceptance ratio becomes

$$r_{k+1} = \frac{\pi(\boldsymbol{\xi})}{\pi(\boldsymbol{\theta})} \tag{2.13}$$

which is the original scheme of the Metropolis algorithm. This choice, makes MCMC more simple to be applied and is the type of proposal PDF used in this project.

Symmetric proposal PDF means that $\boldsymbol{\xi}_{k+1} = \boldsymbol{\theta}_k + \boldsymbol{\zeta}_{k+1}$, where $\boldsymbol{\zeta}_{k+1}$ ('proposal radius') are i.i.d. with a fixed symmetrical density, with some scaling factor $\sigma > 0$. More specifically, if we choose normal proposal distribution, $\boldsymbol{\zeta}_i \sim N(0, \sigma)$, and if we choose uniform proposal distribution, $\boldsymbol{\zeta}_i \sim U(-\sigma, \sigma)$. If σ is very small, then almost all the proposed moves will be accepted, but the state space will get explored too slow, the chain will not *mix* well and the results will not be satisfactory (see subsection 2.3.5). On the other hand, if σ is too large, most of the new proposals will be very far from the current sample and thus more likely to be rejected. Hence, the Markov chain will have many repeated samples which increases the dependency between them, and thus, bad results will be observed. A good way to check if the chosen proposal performs well, is to view the *mixing* of the algorithm. This can be done using trace plots, see subsections 2.3.5 and 2.5.2.

In the high dimensional case however, achieving high acceptance rate for all dimensions (good mixing) is not a simple task. Choosing an n-dimensional proposal PDF, with only one group of n uncertain parameters, will generally lead to zero acceptance phenomena. For that reason the modified Metropolis - Hastings algorithm was introduced [1],[15]. This algorithm, allows a grouping

of the uncertain parameters with regard to the special complications of the system of interest. Two issues related on this procedure are important. First, how to group the parameters and second, what proposal should be chosen for each group. For the first, in general, parameters with high correlation between each other should be put in the same group and the number of parameters in each group should be kept small. For the second, deciding what proposal PDF to choose for every group, depends on the available information for constructing the proposal PDF, in addition to the role each group of uncertain parameters has in the failure.

In the present project, the standard Metropolis - Hastings algorithm was used instead of the modified Metropolis - Hastings algorithm. This choice was made after the observation that the modified MH algorithm is strongly dependent on the grouping pattern of the input variables of the problem. Accordingly from the fact that the grouping should be based on the correlation between the input variables, a deeper investigation in the nature of the problem is required. As a result the standard MH algorithm is more efficient than the modified MH algorithm if a method with global applicability, i.e. without consideration of any special characteristics of the problem, is to be developed. On the other hand, for a specific problem, the use of the modified MH algorithm in addition to a proper grouping of the input variables, may yield improved efficiency in contrast to the standard MH algorithm.

2.3.5 Numerical example

In order to illustrate the procedure of generating samples using the Markov Chain Monte Carlo simulation method (MCMC) and for checking its efficiency, an example is developed is the present subsection.

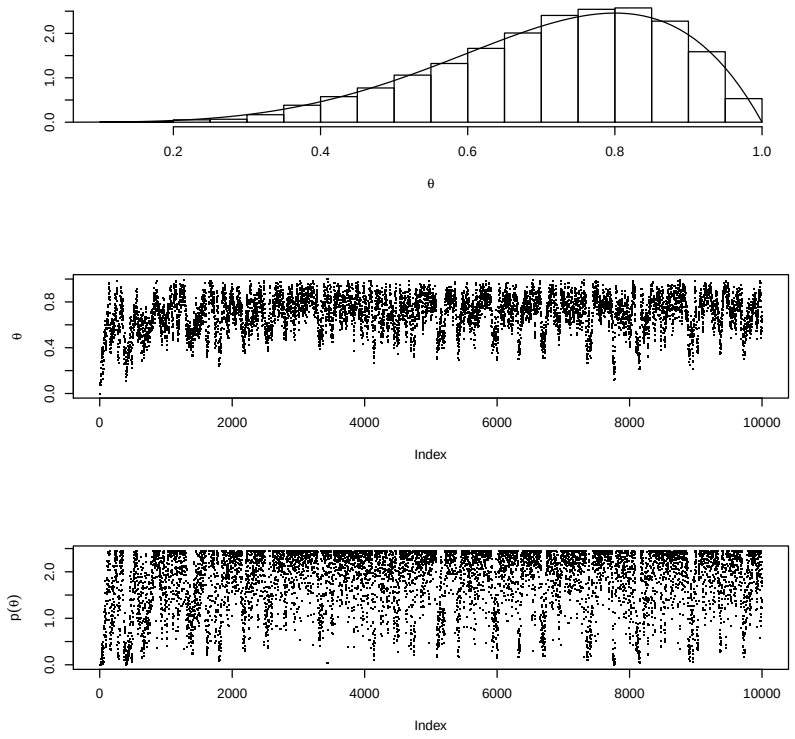


Figure 2.1: 10,000 MCMC samples of the Be(5,2) density. **Top panel**: Histogram of samples from the Metropolis-Hastings algorithm with proposal radius equal to 0.1. The solid line is the Be(5,2) density. **Middle panel**: θi plotted against i. **Bottom panel**: $p(\theta i)$ plotted against i

Suppose we want to generate a sample $\theta_1, ..., \theta_{10,000}$ from the beta distribution with shape parameters $\alpha = 5$ and $\beta = 2$ (Be(5,2)). We arbitrarily choose a proposal density $p^*(\theta_{i+1}|\theta_i) = U(\theta_i - 0.1, \theta_i + 0.1)$ and arbitrarily choose $\theta_1 = 0$. Using the Metropolis-Hastings algorithm with the mentioned characteristics, we obtain the results shown in figure 2.1.

The top panel of figure 2.1 shows the result. The solid curve is the Be(5,2) density and the histogram is made from the samples generated by the Metropolis-Hastings algorithm. They match closely, showing that the algorithm performed well.

Assuming that convergence conditions have been met and that the algorithm is well constructed, MCMC chains are guaranteed eventually to converge and deliver samples from the desired distribution. But the guarantee is asymptotic and in practice the output from the chain should be checked to diagnose potential problems that might arise in finite samples.

The main thing to check is *mixing*. An MCMC algorithm operates in the space of $\boldsymbol{\theta}$. At each iteration of the chain, i.e. for each value of i, there is a current location θ_i. At the next iteration the chain moves to a new location θ_{i+1}. In this way the chain explores the $\boldsymbol{\theta}$ space. While it is exploring it also evaluates $p(\theta_i)$. In theory, the chain should spend many iterations at values of θ where $p(\theta)$ is large, and hence deliver many samples of θ's with large posterior density, and few iterations at values where $p(\theta)$ is small. For the chain to perform, it must find the mode or modes[2] of $p(\theta)$, it must move around in their vicinity, and it must move between them. The process of moving from one part of the space to another is called *mixing*.

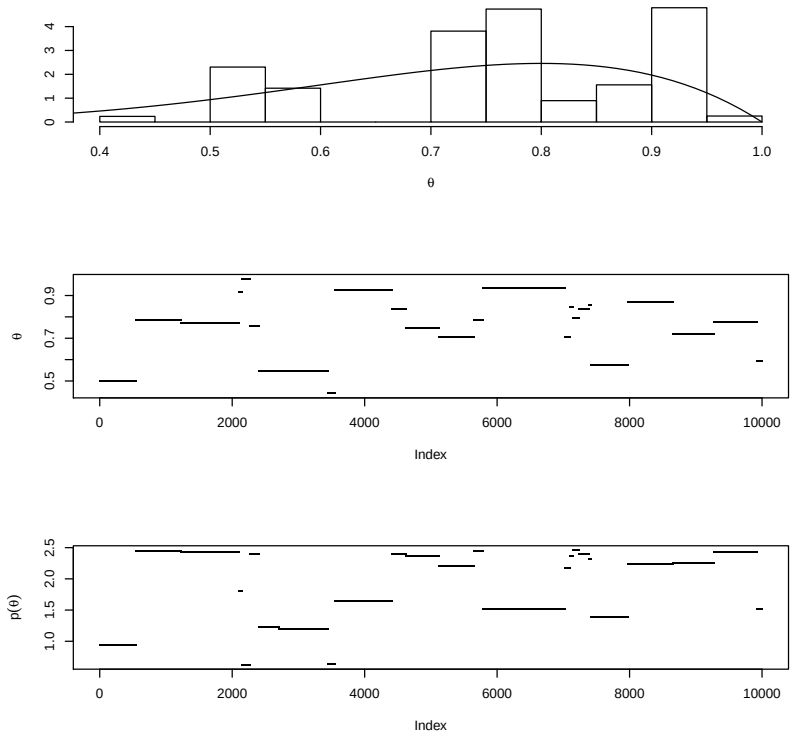


Figure 2.2: 10,000 MCMC samples of the Be(5,2) density. **Top panel**: histogram of samples from the Metropolis-Hastings algorithm with proposal radius equal to 100. The solid line is the Be(5,2) density. **Middle panel**: θi plotted against i. **Bottom panel**: $p(\theta i)$ plotted against i

[2]The mode is the value that appears most often in a set of data. Like the statistical mean and median, the mode is a way of expressing, in a single number, important information about a random variable or a population. The numerical value of the mode is the same as that of the mean and median in a normal distribution, and it may be very different in highly skewed distributions. The mode is not necessarily unique, since the same maximum frequency may be attained at different values. The most extreme case occurs in uniform distributions, where all values occur equally frequently.

The middle and bottom panels of figure 2.1 illustrate mixing. The middle panel plots θ_i against i. It shows that the chain spends most of its iterations in values of θ between about 0.6 and 0.9 but makes occasional excursions down to 0.4 or 0.2 or so. After each excursion it comes back to the mode around 0.8. The chain has taken many excursions, so it has explored the space well. The bottom panel plots $p(\theta_i)$ against i. It shows that the chain spent most of its time near the mode where $p(\theta) \approx 2.4$ but made multiple excursions down to places where $p(\theta)$ is around 0.5, or even less. This chain mixed well.

To illustrate poor mixing we will use the same MCMC algorithm but with different proposal densities. First we will use $p^*(\theta_{i+1}|\theta_i) = U(\theta_i - 100, \theta_i + 100)$ and then $p^*(\theta_{i+1}|\theta_i) = U(\theta_i - 0.00001, \theta_i + 0.00001)$. Figures 2.2 and 2.3 show the results. The top panel shows a very much rougher histogram than figure 2.1; the middle and bottom panels show why. The proposal radius is so large that most proposals are rejected; therefore, $\theta_{i+1} = \theta_i$ for many iterations; and thus, we get the flat spots in the middle and bottom panels. The plots reveal that the sampler explored fewer than 30 separate values of θ. That is too few; the sampler has not mixed well. In contrast, figure 2.3, shows the results for a proposal density $p^*(\theta_{i+1}|\theta_i) = U(\theta_i - 0.00001, \theta_i + 0.00001)$. It should noted, that these two examples of poor mixing where initialized from $\theta_1 = 0.5$ and not 0 as the previous one[3].

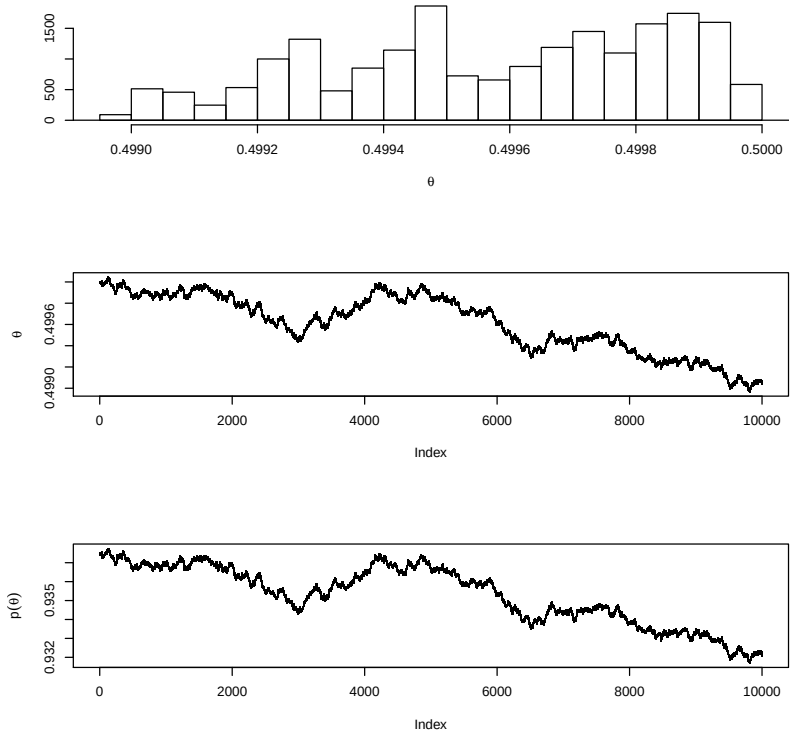


Figure 2.3: 10,000 MCMC samples of the Be(5,2) density. **Top panel**: histogram of samples from the Metropolis-Hastings algorithm with proposal radius equal to 0.00001. The solid line is the Be(5,2) density. **Middle panel**: θi plotted against i. **Bottom panel**: $p(\theta i)$ plotted against i

It is revealed, that θ has drifted steadily downward, but over a very small range. There are no flat spots, so the sampler is accepting most proposals, but the proposal radius is so small that the sampler has not yet explored most of the space. It also has not mixed well.

Plots such as the middle and bottom plots of figures 2.1 to 2.3, are called *trace* plots because

[3]Initialization of this specific MCMC algorithm with too large (100) and too small (0.00001) proposal radius from $\theta_1 = 0$, reveals the problem of the choice of θ_1 and produces acceptance ratios equal to 0. For that reason θ_1 was chose equal to 0.5.

they trace the path of the sampler. For comparison reasons, in appendix A.4, the same results are shown but for the simple Monte Carlo simulation, which realizes i.i.d. samples from the Be(5,2) distribution and can be assumed as the perfect mixing case.

In this problem, good mixing depends on getting the proposal radius not too large and not too small, but just right [10]. To be sure, if we run the MCMC chain long enough, all three samplers would yield good samples from Be(5,2). But the first sampler mixed well with only 10,000 iterations while the others would require many more iterations to yield a good sample. In practice, we have to examine the output of an MCMC chain to diagnose mixing problems.

The problem that developed as example in the present subsection, was derived from [13], and illustrates the results of Markov chain samplers that generate realizations of a simple probability density (step 2. of the Metropolis-Hastings algorithm is not included) which in this case is the Be(5,2). In section 2.4, the developed method of subset sampling, uses Markov chain samplers that generate samples according to conditional probability densities simply by adding step 2. in the Metropolis-Hastings algorithm, which conditions the samples on F.

2.4 SUBSET SAMPLING METHOD

Subset sampling or *subset simulation* method is a method for solving general problems of calculating small failure probabilities in high dimensional spaces which was first introduced as *umbrella sampling* [15]. For the application of the method it is used the Markov Chain Monte Carlo simulation which fits ideally in the basic scheme of Subset sampling procedure. It should be noted, that the present method is suitable for solving general reliability problems due to the fact that it does not involve any special characteristics of the system.

2.4.1 Basic concept

The subset sampling method, is based on the idea that a small failure probability can be expressed as a product of larger failure events. As a result, the rare failure event simulation problem is turning into several simulation problems of more frequent events, the subsets, easier to simulate. Assume a failure event F subdivided into a decreasing sequence of m failure events $F_1 \supset F_2 \supset ... \supset F_m = F$, so that $F_k = \bigcap_{i=1}^{k} F_i$, $k = 1, 2, ..., m$. The determination of the intermediate failure events can be realized as a series g_i, $i = 1, ..., m$ of limit values, also referred as threshold level values, this is

$$F_i = \{\boldsymbol{\theta} : g(\boldsymbol{\theta}) \geq g_i\} \tag{2.14}$$

where $g(.)$ expresses an uncertain demand and $g = g_m > ... > g_2 > g_1$ expresses a specified capacity of the system. In a structural problem, $g(.)$ may represent loads, stresses, etc. while g_m could be roughly understood as deterministic material strengths, component stiffness, etc. or just a fixed value, usually equal to 0.

By definition of the conditional probability we have

$$\begin{aligned} P_F = P(F_m) &= P(\bigcap_{i=1}^{m} F_i) \\ &= P(F_m|\bigcap_{i=1}^{m-1} F_i)P(\bigcap_{i=1}^{m-1} F_i) \\ &= P(F_m|F_{m-1})P(\bigcap_{i=1}^{m-1} F_i) = ... \\ &= P(F_1)\prod_{i=1}^{m-1} P(F_{i+1}|F_i) \end{aligned} \tag{2.15}$$

In equation (2.15), the probability of failure is expressed as a product of a sequence of conditional probabilities $\{P(F_{i+1}|F_i) : i = 1, ..., m-1\}$ and $P(F_1)$. This makes the implementation of the idea of subset sampling possible, due to the fact that a small probability of failure P_F can be estimated by determining these sufficiently larger valued quantities. The smaller the P_F, the more rare the failure event, and the more number of simulations required for having one sample within the failure region. It is possible to choose the intermediate failure events in such a way that the probabilities involved in equation (2.15) have a value around 0.1, this is $P(F_1), P(F_{i+1}|F_i) \sim 0.1, i = 1, 2, 3, 4$ and then the probability of failure is $P_F \sim 10^{-5}$ which is a very small probability for estimating with direct Monte Carlo Simulation. On the other hand, probabilities of the class of 0.1, are large enough to be simulated. Consequently, we have turned a simulation problem which requires a large amount of simulation runs, into (only) five separate simulation problems each of which requires sufficiently less simulation runs to yield efficient results.

In order to accomplish the whole task, it is necessary to calculate the quantities $P(F_1)$,$\{P(F_{i+1}|F_i) : i = 1, \ldots, m-1\}$. Concerning the fact that F_1 is sufficiently large, the determination of $P(F_1)$ does not seem very complicated, it can be accomplished using direct MCS. On the other hand, the determination of the conditional probabilities $\{P(F_{i+1}|F_i) : i = 1, ..., m-1\}$ seems much more complicated in the sense of simulation. Markov Chain Monte Carlo simulation is used, for the reason that it is ideal for simulating conditional probabilities and its applicability on the subset sampling method is very efficient. More specifically, an estimator of the form $q(\boldsymbol{\theta}|F_i) = q(\boldsymbol{\theta})\mathbb{I}_{F_i}(\boldsymbol{\theta})/P(F_i)$ necessitates the simulation of samples according to the conditional distribution of $\boldsymbol{\theta}$ given that it lies in F_i, by using Markov chain samples with limiting stationary distribution equal to $q(.|F_i), i = 1, ..., m-1$. The whole procedure of subset sampling method is explained in subsection 2.4.2.

2.4.2 Subset Simulation procedure

The first task to be accomplished, is the estimation of the probability $P(F_1)$. This is done by using the estimator $\tilde{P}_1$ in a direct Monte Carlo simulation [1]

$$P(F_1) \approx \tilde{P}_1 = \frac{1}{N}\sum_{k=1}^{N} \mathbb{I}_{F_1}(\boldsymbol{\theta}_k) \tag{2.16}$$

Where $\{\boldsymbol{\theta}_k : k = 1, ...N\}$ are independent and identically distributed (i.i.d) samples simulated according to the (target) PDF $q(\cdot p)$. From these MCS samples we can directly obtain some samples distributed as $q(\cdot p|F_1)$, simply as those which lie in F_1. Starting from each (or some) of these, we can generate Markov chain samples by using the Metropolis algorithm. These ones will also be distributed as $q(\cdot p|F_1)$ simply by generating samples distributed as $q(\cdot p)$ in Step 1 of the Metropolis algorithm and rejecting those which do not lie in F_1 in Step 2. These samples can be used to estimate $P(F_2|F_1)$ by enabling an estimator $\tilde{P}_2$ similar to equation (2.16). Observe that the Markov chain

samples which lie in F_2 are distributed as $q(\cdot p|F_2)$ and thus they provide more 'seeds'[4] for simulating more samples according to $q(\cdot p|F_2)$ in order to estimate $P(F_3|F_2)$. Repeating this process we can compute all the conditional probabilities required to reach the failure event of interest. At the i-th conditional level, $1 \leq i \leq m-1$, let $\boldsymbol{\theta}_k^{(i)} : k = 1, ..., N$ be the Markov chain samples with distribution $q(\cdot p|F_i)$, possibly coming from different chains generated by different 'seeds', then according to [1],

$$P(F_{i+1}|F_i) \approx \tilde{P}_{i+1} = \frac{1}{N}\sum_{k=1}^{N} \mathbb{I}_{F_{i+1}}(\boldsymbol{\theta}_k^{(i)}) \tag{2.17}$$

Finally, inserting the equations (2.16) and (2.17) into (2.15), the failure probability estimator is

$$P_F \approx \tilde{P}_F = \prod_{i=1}^{m} \tilde{P}_i \tag{2.18}$$

The MCS estimator $\tilde{P}_1$ in equation (2.16) computed using the i.i.d. samples $\{\boldsymbol{\theta}_1, ..., \boldsymbol{\theta}_N\}$, converges almost surely to P_1 (Strong Law of Large Numbers, see appendix A.1), is unbiased, consistent, and Normally distributed as $N \to \infty$ (Central Limit Theorem, see appendix A.1) [13],[1]. The estimators of the conditional probabilities in (2.15), are also unbiased [1] but the Markov chain samples are dependent. Despite that dependency, all the estimators $\tilde{P}_i$, still have the usual convergence properties of estimators using independent samples [12]. For example, $\tilde{P}_i$, converges almost surely to $P(F_i|F_{i-1})$ (Strong Law of Large Numbers), is consistent, and Normally distributed as $N \to \infty$ (Central Limit Theorem) [1],[13]. Concerning the estimator $\tilde{P}_F$, it converges almost surely to P_F as $N \to \infty$, because its components do. Due to the correlation between the conditional estimators $\tilde{P}_i$, $\tilde{P}_F$ is biased for every N but it is asymptotically unbiased [1]. This correlation arises by the fact that some samples used for computing $\tilde{P}_i$, are also used to compute $\tilde{P}_{i+1}$.

The generated samples of a case study with two input variables x_1, x_2 where $x_1 \sim N(2.5, 0.45)$ and $x_2 \sim N(3.5, 0.60)$ and six subsets, is shown in figures 2.4 to 2.9. In the figures, the intermediate and the target limit states are plotted as well. The objective function of this case study is $f(x_1, x_2) = {x_1}^2 + {x_2}^2$ and the intermediate threshold level values are: 10, 20, 30, 40, 50 and 60 which is the target threshold level value.

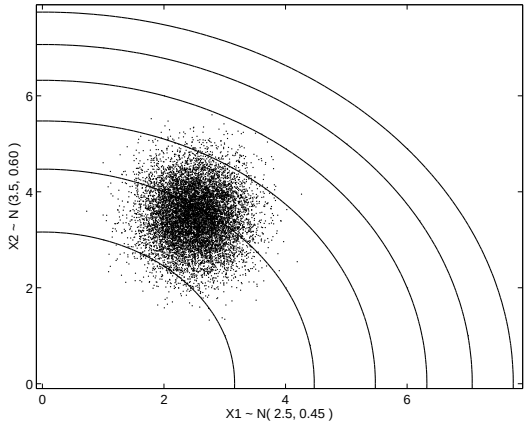


Figure 2.4: Direct MC simulation i.i.d. samples

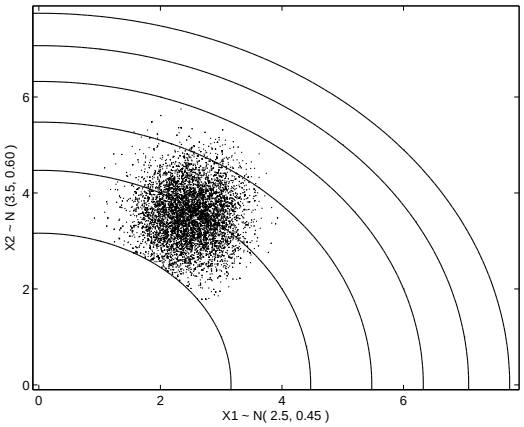


Figure 2.5: MCMC samples conditioned in F_1

The reason that the higher conditional levels, appear to have less samples, is that the conditioning is getting stronger, the acceptance ratio smaller and hence more samples are being rejected. As a result, these samples are equal and they coincide to each other on the plot. Moreover, in direct MCS there is no coincidence of samples at all, so the samples appear to be much more than in the other levels.

[4] Literally a seed is what we put in the ground and given water and time, becomes a plant. In this context, the word seed, refers to a Markov chain starting point. The starting point of a Markov chain, is a very important aspect and it should be specified in advance, see $\boldsymbol{\theta_1}$ in the third paragraph of section 2.3.

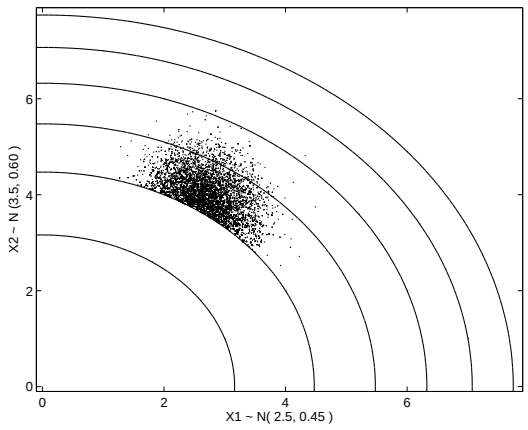


Figure 2.6: MCMC samples conditioned in F_2

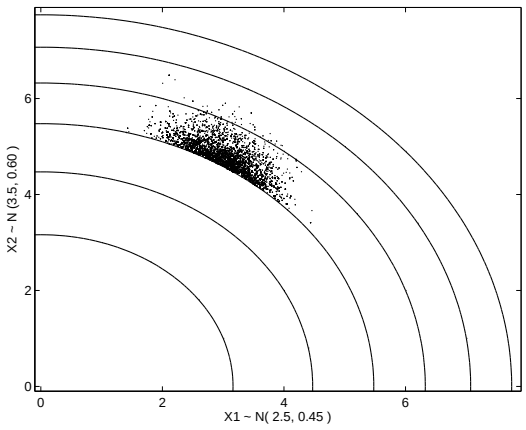


Figure 2.7: MCMC samples conditioned in F_3

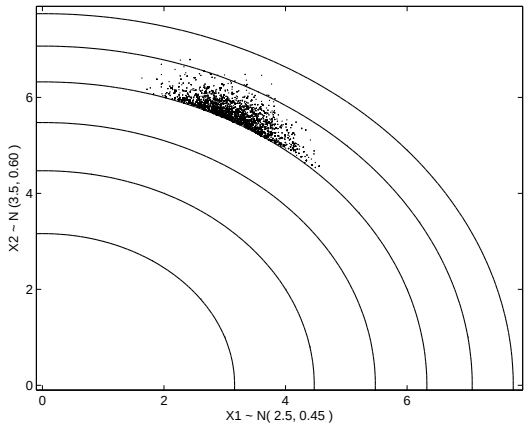


Figure 2.8: MCMC samples conditioned in F_4

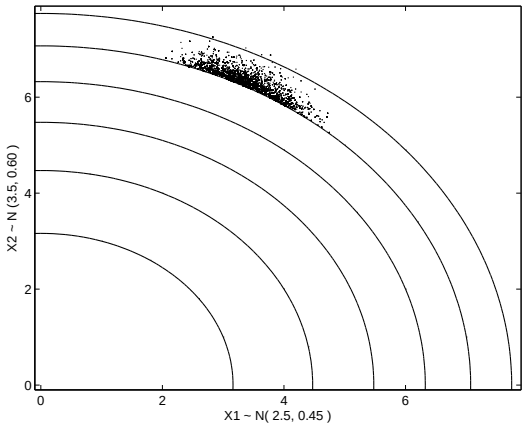


Figure 2.9: MCMC samples conditioned in F_5

2.4.3 Threshold levels and proposal PDF

As target failure event F, it is assumed the (positive) exceedance of a specified limit value g by a multivariate function $g(\boldsymbol{\theta})$ expressing the quantity of interest. Where $\boldsymbol{\theta}$ is the vector of variables which represents the input parameters of the problem. As a result, the intermediate threshold levels will be a sequence of ascending values to the target failure event value $g_1, ..., g_m = g$. Consequently, the choice of the intermediate threshold values $g_1, ..., g_m$ is essential for the efficiency of the subset simulation procedure.

There are two aspects in the choice of the threshold levels, that require special attention. The choice of the value of the first level g_1 and the choice of the distance between them. A slowly ascending sequence may produce large probabilities but it needs more threshold levels m to reach the target limit state increasing the total numbers of samples $N_T = mN$ in the whole procedure. On the other hand, a fast sequence means less threshold levels, i.e. more rare failure events which increases the total required simulation runs. It can be observed, that the choice of the intermediate threshold values, is a trade-off between the number of samples required in each simulation level and the number of simulation levels required to reach the target failure event.

In the present project, the choice of the intermediate threshold values accomplished as follows. The mean value of each the input variable, is inserted in the objective function to calculate the functional value of the means. This functional value, is set as the first threshold level g_1 and the rest of the

space till the target limit value g_m is divided by the number of subsets to calculate the distance between the threshold levels. This is not a convenient choice for general use but at least for the case studies of this project, it performs efficiently.

A sufficiently more effective way for choosing the intermediate threshold values g_i is described in [1],[15] and the basic idea is to estimate the g_i's adaptively within the algorithm. Adaptively means that the g_i's will be chosen in such a way that the resulting conditional probabilities $P(F_{i+1}|F_i)$ will be all equal to a fixed value $p_0 \in (0, 1)$. This needs a procedure of sorting the samples of every threshold level, but then if we set the $\{g_i : i = 1, ..., m-1\}$ threshold level equal to $(1 - p_0)N$, we can force the algorithm to choose a threshold value so that in a total number of samples N, $p_0 \cdot N$ of them lead to failure. Thus the conditional probability of failure is $p_0 \cdot N/N = p_0$ which is the given value. As proposed in [1] and in [15], a value of $p_0 = 0.1$ yields good efficiency for $P_F \sim (10^{-3}, 10^{-6})$. The main disadvantage of this procedure is the requirement of sorting the samples. Sorting is always a computationally expensive process and for that reason the adaptive specification of the threshold level values was not adopted by the present project.

As in the simple MCMC procedure described in the section 2.3, the choice of the proposal PDF is essential for the efficiency of the method especially for a high dimensional case. Generally the closer the proposal PDF to the conditional PDF, the more effective the algorithm. Moreover, applicability in high dimensions often reaches the limits of the Metropolis - Hastings algorithm and a grouping of the uncertain parameters is required using the modified Metropolis - Hastings algorithm described in [1],[15]. If some uncertain parameters belonging in different groups, are strongly correlated when conditional on a failure region, the distribution of the candidate state will not be close to the conditional PDF. Consequently, a solution to this drawback, is to group high correlated uncertain parameters together.

2.5 SUBSET SAMPLING ALGORITHM

This section contains a demonstration and some discussion over the developed subset sampling algorithm in addition to two case studies for testing the performance of the algorithm. Finally the subset sampling method, is compared to the standard MCS procedure for testing its efficiency.

2.5.1 The algorithm

The subset sampling algorithm, has been developed and tested using the open source programming language 'R', which is the most appropriate for programming statistical, probabilistic and stochastic processes. This choice has been made for convenience and the implementation in the existing stochastic analysis tool 'Winfuz', has been done after translating the present algorithm into Fortran. In the present subsection, the algorithm after the translation into Fortran is being demonstrated with the use of structograms and a brief explanation on them. It should be noted, that within the algorithm, the final failure event has not taken into account as a subset, despite the fact that in section 2.4 it has been suggested as failure event (subset) F_m. The performance of the algorithm is not affected, because the final failure event has been taken into account in the computation. The only confusion arises when specifying the parameter 'nls' for the number of subsets. Moreover, the Fortran source code of the subset sampling algorithm (subroutine 'MCMC'), can be found in appendix A.5.

The input parameters of the algorithm are:

```
limit_value    final limit state value, if exceeded by the objective function, failure occurs
underflow      yes / no - underflow / overflow of the limit_value by the objective function
```

```
mc              total number of samples per subset
nv              number of dimensions
nls             number of subsets
nmc             number of Markov Chains per subset
ipara(11)       1 / 2 - proposal PDF - uniform / normal
ipara(12)       0 / 1 - proposal radius - user specified / standard deviation
```

The control parameters of the algorithm are:

```
f1              functional value for the input parameter's means
step            distance between adjacent threshold level values
LS(1:nls+1)     threshold level values, they are the bounds of the subsets

r(1:mc,1:nv)    random numbers
f(:)            functional value of any vector x(1:nv)
SPi(1:nls+1)    number of points, the functional values of which, crossed the corresponding
                threshold level value LS
temp(:,1:nv)    temporary saves the points, the functional values of which, crossed the
                current subset's threshold value LS
Psing(1)        probability of exceedance of the threshold level value LS(1)
ZS(1:mc,1:nmc+1,1:nv)  temporary saves all the points of every chain of the current subset

prev(1:nv)      the previous state point
prop(1:nv)      the proposal point, to be the current state point

X(1:nv)         probability density function of the (still) proposal point ZS(i,m,1:nv)
Y(1:nv)         probability density function of the previous state point ZS(i-1,m,1:nv)
A               joint probability density for all dimensions of the proposal state
                point ZS(i,m,1:nv)
B               joint probability density for all dimensions of the previous state
                point ZS(i-1,m,1:nv)
ratio           acceptance ratio of the Metropolis - Hastings algorithm
p               random number taken from the last column (nv+1) of the
                random numbered matrix r
Psing(k)        k = 2:nls+1 conditional probability P(k|k-1), is the probability of a point
                to lie in the subset k given that it lies in the previous subset k-1
```

The only output parameter of the algorithm is:

```
Pf              probability of failure
```

The following structogram, shows the initial calculations of the algorithm.

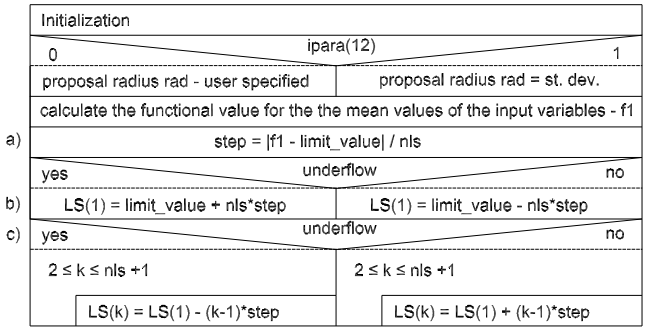


Figure 2.10: Structogram for the initialization

(*a*) It has been assumed that the functional value f1 for the means of the input parameters, is greater/less than the limit_value for underflow/overflow.

(*b*) We set the threshold level value LS(1) equal to f1 and divide the distance (step) between LS(1) and limit_value by nls to specify the distance between adjacent threshold level values LS. We have nls+1 subsets and nls+1 threshold level values but here we need the distances and for that reason the division is done by nls and not nls+1.

(*c*) Calculation of the next threshold level values $LS(2 : nls + 1)$ by adding step every time. The total number of subsets is *nls*, as a result the number of threshold values is $nls + 1$.

The direct MCS used to calculate the probability P(1) of a point to lie in the first subset, is demonstrated with the use of the following structogram.

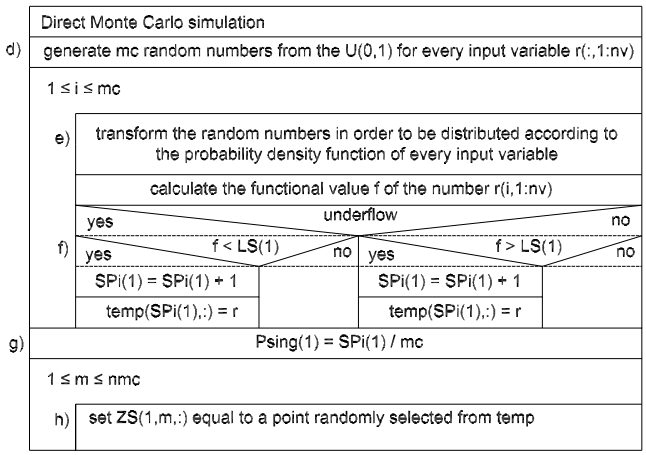


Figure 2.11: Structogram for the direct MCS

(*d*) The subroutine 'randomnumbers' is called and generates a 2-dimensional matrix ($mc * nv$) of random numbers. In Fortran it is more convenient to generate all the required random numbers in advance, save them in an array and use them subsequently.

(*e*) The subroutine 'transform' is called, has as input, a random number according to U(0,1) for one input variable, and transforms it as it was distributed according to this input variable's distribution.

(*f*) Checks for exceedance of the 1st threshold level value. If so, increases the counter $Spi(1)$ by 1 and temporarily saves the 'successful' point in the matrix temp.

(*g*) Calculation of the probability of exceedance of the 1st threshold level value (or probability of a point to lie in the 1st subset), using equation (2.16) in subsection 2.4.2.

(*h*) Sets the starting point of each Markov chain of the next step, to a randomly selects a point from the matrix *temp*. The chains of the 1st loop of the MCMC process, will be used to calculate the conditional probability $P(F_2|F_1)$. For that reason, the starting points should belong to the first subset F_1 and that's why they are chosen from the matrix *temp*.

The main loop of the subset sampling algorithm for the calculation of the conditional probabilities $P(F_k|F_{k-1})$ for $k = 2 : nls + 1$, is demonstrated in the following structogram.

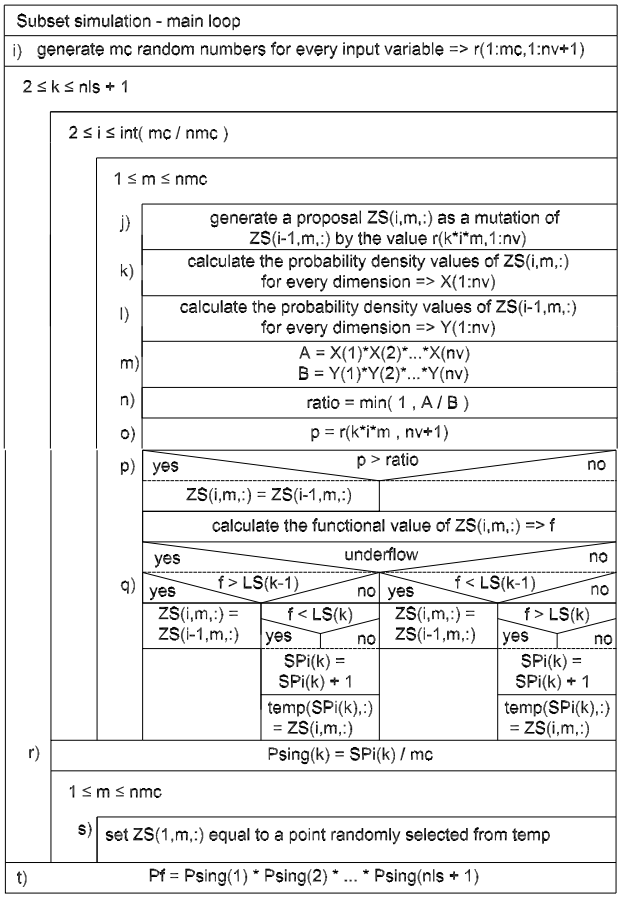


Figure 2.12: Structogram for the subset simulation - main loop

(*i*) Generation of the required random numbers, with an extra column $(nv+1)$ which will be used in *o*).

(*j*) The subroutine 'proposal' is called. It has as input the previous state point $ZS(i-1, m, 1 : nv)$ and a random numbered vector $r(1 : nv)$, and returns the proposal $ZS(i, m, 1 : nv)$ as a mutation of $ZS(i-1, m, 1 : nv)$. The mutation is done according to the proposal PDF, using the proposal density and the distribution parameters (or moments) of each input variable.

(*k*) The subroutine 'transform' is called. It calculates the probability density value $X(j)$ of the input (still proposal) point $ZS(i, m, j)$, according to the probability density function of the specific input variable of the dimension j. This procedure is repeated for every dimension separately, $j = 1 : nv$.

(*l*) The same as *k*), but the input point is the previous state point $ZS(i-1, m, j)$ this time.

(*m*) Calculates the joint probability density values A and B of the proposal and the previous state points respectively.

(*n*) Calculates the acceptance ratio according to equation (2.12) in subsection 2.3.2. The proposal probability density values p^*, are not included in the calculation because the two options for proposal PDF are both symmetric (uniform, normal) PDF's and as a result $p^*(\boldsymbol{\theta}_k|\boldsymbol{\xi}_{k+1}) = p^*(\boldsymbol{\xi}_{k+1}|\boldsymbol{\theta}_k)$.

(*o*) p is a random number from the extra column $nv + 1$ of the random numbered matrix r generated in *i*).

(*p*) This is an 'one out of one' binomial experiment[5]. It is used to set the current state point equal to the (still) proposal point with probability r and equal to the previous state point with probability $1 - r$. This is, if the acceptance ratio is close to 1, i.e. the probability density value of the (still) proposal point is greater that the probability density value of the previous point, *ratio* will probably be greater than the random number $p \sim U(0, 1)$ and the proposal will be accepted as the current point. If the acceptance ratio is not greater than p, the current point is set equal to the previous state point $ZS(i - 1, m, 1 : nv)$ and the proposal is being rejected.

(*q*) The sampled proposal points that do not lie in the previous subset $(k - 1)$, are being rejected[6] and the previous state point becomes the current as well. If the proposal lies on the previous subset, it is being checked if it also lies in the next subset (k). If it does, the counter $SPi(k)$ increases by 1 and this point is temporarily saved in the *temp* matrix.

(*r*) Calculation of the conditional probability $P(F_k|F_{k-1})$ using (2.17) in the subsection 2.4.2.

(*s*) Random selection of *nmc* (number of Markov chains per subset) points from the matrix *temp* to become the starting points $(ZS(1, m, 1 : nv))$ of the Markov chains of the next subset. These chains will be used to calculate the conditional probability $P(F_{k+1}|F_k)$, and thus the starting points should lie on the k-th subset. This is the reason we choose them from the *temp* matrix.

(*t*) Calculation of the final probability of failure *Pf* using equation (2.18) in the subsection 2.4.2.

The subroutine 'proposal', used for generating a proposal state by mutating the current sate point, is demonstrated in the following structogram.

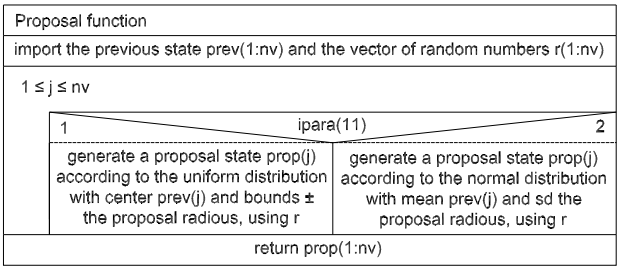


Figure 2.13: Structogram of the subroutine 'proposal'

2.5.2 Numerical examples and performance diagrams

In the present project, as reliability problem, has been assumed the calculation of the probability of exceedance of some given limit state value by an objective function. The objective function may

[5]A binomial experiment is an experiment with a fixed number of independent trials (here one), each of which has exactly two outcomes.

[6]We want to calculate the conditional probability $P(F_i|F_{i-1})$ and for that reason it is mandatory that all the samples of the i-th step should lie in F_{i-1}.

depend on more than one variables (multivariate function), which variables are uncertain, thus, each of them is defined by a probability density function (PDF). For the investigation of that problem, two different numerical examples, one with two input variables and another one with five input variables are performed in the present chapter.

1. Case - 2**-dimensions**

Objective Function: $f(x_1, x_2) = {x_1}^2 + {x_2}^2$

$x_1 \sim N\ (\ 2.5,\ 0.45\ )$

$x_2 \sim N\ (\ 3.5,\ 0.60\ )$

Limit State Value: $f_{limit} = 50$

2. Case - 5**-dimensions**

Objective Function: $f(\mathbf{x}) = \sum_{i=1}^{n-1} 100({x_{i+1}} - {x_i}^2)^2 + (1 - x_i)^2 \quad \text{for} \quad i = 1, ..., 4$

$x_1 \sim N\ (\ 0,\ 0.72\ )$

$x_2 \sim N\ (\ 0,\ 0.36\ )$

$x_3 \sim LN\ (\ 1.05,\ 0.27\ )$

$x_4 \sim N\ (\ -1.5,\ 0.20\ )$

$x_5 \sim N\ (\ -0.5,\ 0.50\ )$

Limit State Value: $f_{limit} = 5\ \cdot 10^5$

The Subset Sampling algorithm has been applied in both cases and the performance has been checked. This control process is realized by checking the *mixing* of the Markov chains using the so called *trace plots* (see subsection 2.3.5) and by checking if the algorithm converges to a value of the probability of failure P_F. Moreover, the essential aspect of ergodicity of the Markov chains is being checked through their limiting stationary distributions. The results are demonstrated by means of performance diagrams.

Initializing the algorithm for the 2-dimensional case ($x(1) \sim N(2.5, 0.45)$ and $x(2) \sim N(3.5, 0.60)$) with the following starting parameters,

```
limit_value = 43         final limit state value
underflow   = no         overflow of the limit_value by the objective function

mc          = 100000     total number of samples per subset
nv          = 2          number of dimensions
nls         = 5          number of subsets
nmc         = 10         number of Markov Chains per subset
ipara(11)   = 2          proposal PDF - normal
ipara(12)   = 1          proposal radius equal to the corresponding standard deviation
```

yields the histograms shown in figures 2.14 to 2.18 for the different limit states. In these plots, the histogram represents the frequencies of the samples, thus this is the limiting stationary distribution of the Markov chain, and the solid line is the probability density function of each variable.

The results were:

$$P(F_1) = 0.52012$$

$P(F_2) = 0.32381$

$P(F_3) = 0.18630$

$P(F_4) = 0.11766$

$P(F_5) = 0.07982$

$P(F_6) = 0.05559$

and finally, $P_F = 1.638113 \cdot 10^{-5}$

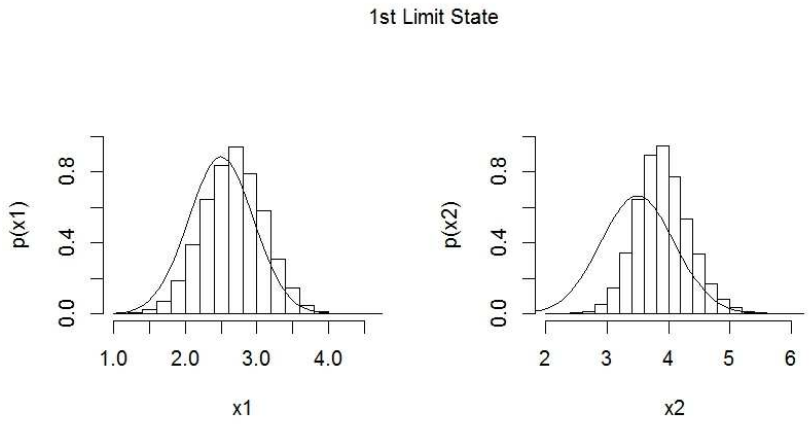


Figure 2.14: Histograms of the two input variables for the 1^{st} limit state

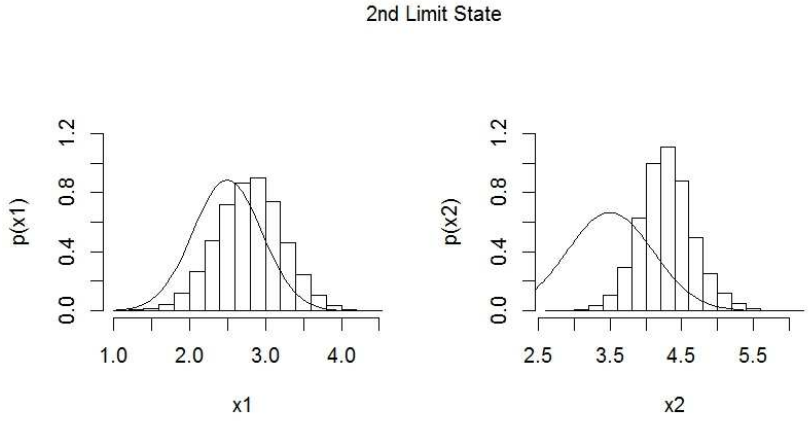


Figure 2.15: Histograms of the two input variables for the 2^{nd} limit state

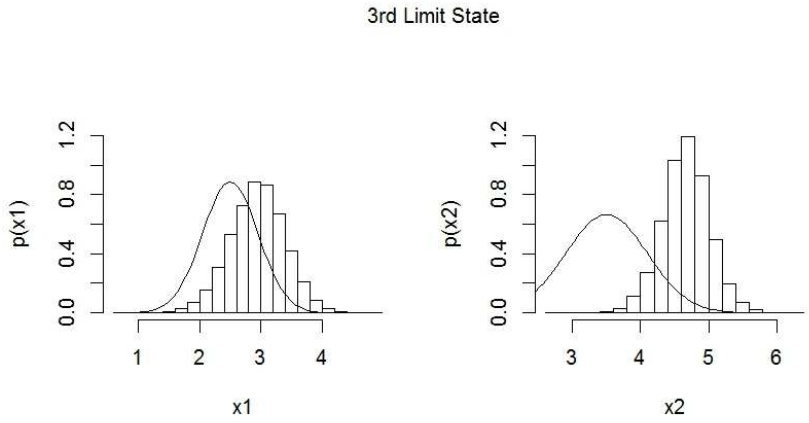


Figure 2.16: Histograms of the two input variables for the 3rd limit state

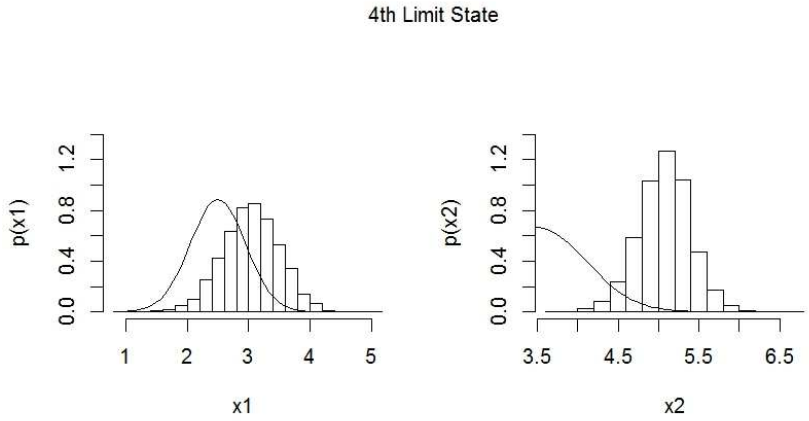


Figure 2.17: Histograms of the two input variables for the 4th limit state

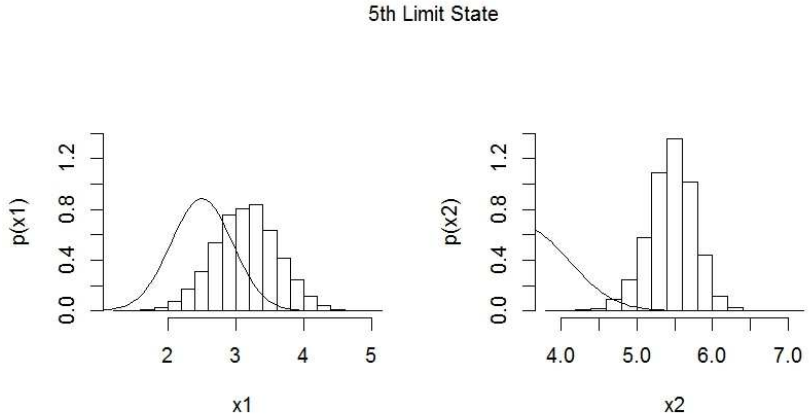


Figure 2.18: Histograms of the two input variables for the 5th limit state

In the plots in figures 2.14 to 2.18 it can be observed that for the first limit state the frequencies of the samples are close to the probability density function. This is because as first failure event, has assumed the exceedance of the functional value for the input variables means. Hence, these samples are not strongly forced to depart from their PDFs[7]. On the other hand, the histograms for the next limit states, feature a gradually larger alteration from the variable's PDF. This is explained by the conditioning on the failure regions. The more far from the definition domain of the variables, the largest the dependency on the condition of exceeding the current threshold value and thus, the largest alteration from the initial PDF.

Running the algorithm for the 5-dimensional case with the same initial parameters as before apart from *nv* which is now 5, we have the following results.

$P(F_1) = 0.99995$

$P(F_2) = 0.00560$

$P(F_3) = 0.12400$

$P(F_4) = 0.25443$

$P(F_5) = 0.33986$

$P(F_6) = 0.40264$

and finally, $P_F = 4.68394 \cdot 10^{-5}$

Indicatively, the histogram of the 3^{rd} input variable x_3 which follows a log-normal distribution for the 1^{st} limit state is shown in figure 2.19. The reason that the histogram matches almost perfectly with the line of the PDF, is that the functional value of the mean values of the input variables, which has been set as the first threshold level value, has coincidentally fallen very close to the objective function's minimum. As a result, the first subset is almost equal to the *x*-space domain, observe that $P(F_1) \approx 1$.

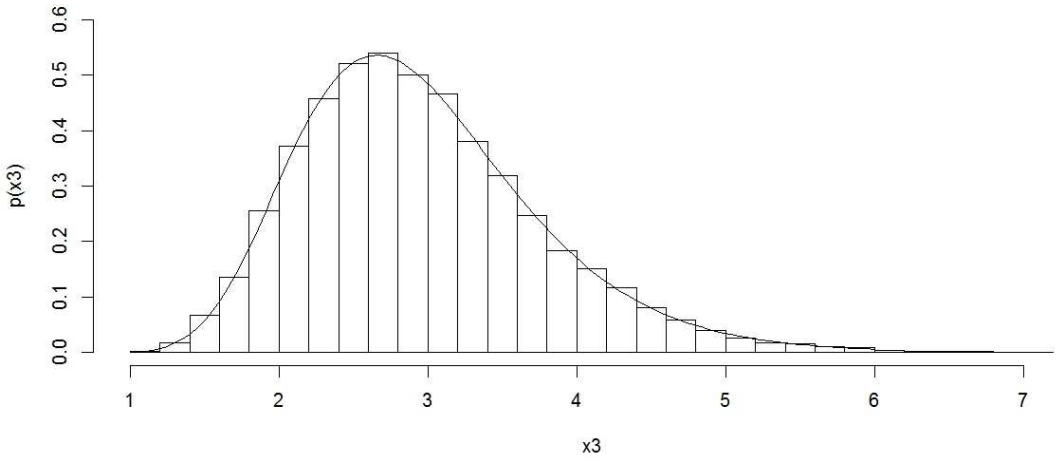


Figure 2.19: Histogram of the 3^{rd} input variable x_3 for the 1^{st} limit state

Now, the efficiency of one (randomly selected) Markov chain for each failure event and for every input variable is checked using *trace plots* to control the *mixing* of the algorithm. In such plots,

[7] If the first subset had chosen equal to the entire *x*-space domain, i.e. the first threshold level value equal to the minimum of the objective function, the histograms and the line of the PDF would have matched perfectly.

the index of the sample is plotted against the sample's value for each input variable see subsection 2.3.5. The label of the *y* axis refers to the input variable that is plotted in each plot. In figure 2.20, the trace plots of one (out of ten in total) Markov chain exploring the second intermediate failure region is shown. The plots corresponding to the rest failure regions, can be found in appendix A.2, figures A.4 to **??**.

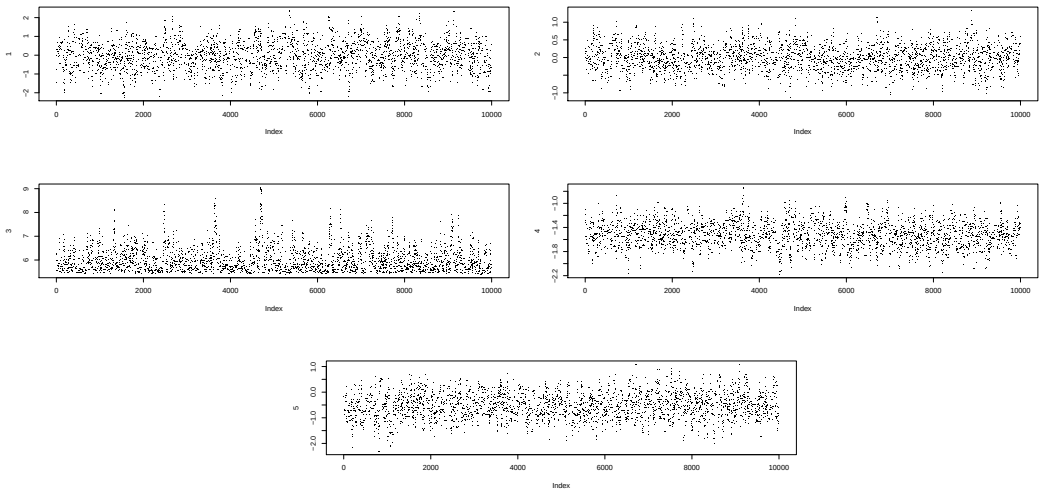


Figure 2.20: Trace plots of all input variables for the 2^{nd} limit state

Figure 2.20, proves that the chain mixes well (see subsection 2.3.5 for details about mixing) and that it reaches its limiting stationary distribution. It explores the space smoothly and tries to follow the ridge of the conditional probability density function $P(F_3|F_2)$. If the algorithm proposes a move along the ridge, the proposal is likely to be accepted. But if the algorithm proposes a move that takes us off the ridge, the proposal is likely to be rejected because the probability density value would be small and therefore the acceptance ratio would be small. But that is not happening here, the algorithm seems not to be stuck, so we surmise that it is proposing moves that are small compared to the widths of the ridges.

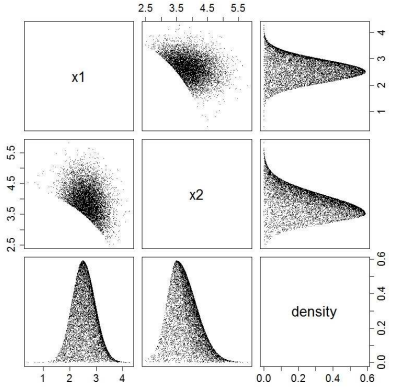


Figure 2.21: Pairs plot for the 1st limit state

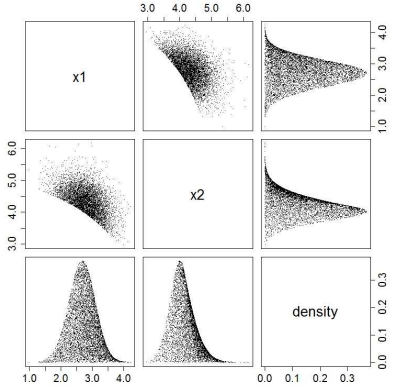


Figure 2.22: Pairs plot for the 2nd limit state

The reason why x_3 has a different trace plot from the other variables in figure 2.20, is that is the only one that follows a non-symmetric log-normal distribution. In addition to that, by the definition of the problem, x_3 takes larger values than the other variables and hence the objective function depends more to that variable than to the other ones. As a result, due to the gradually stronger conditioning,

variable x_3 is more and more strongly forced to take smaller values than in the lower conditional levels.

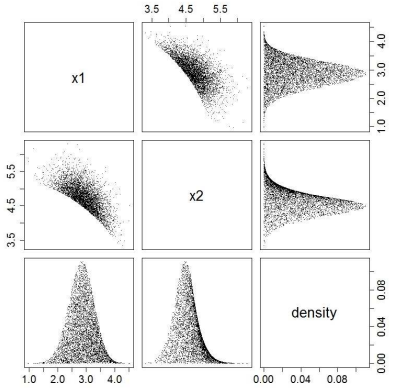


Figure 2.23: Pairs plot for the 3rd limit state

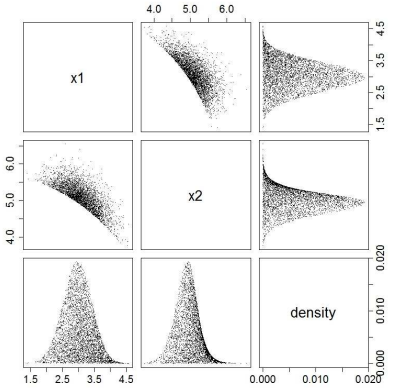


Figure 2.24: Pairs plot for the 4th limit state

Another intuitive way to check the efficiency and the convergence of the method, is to plot the samples of one variable against each one of the others. This kind of plot is called *pairs plot* and it is very useful because it provides a visual idea of the sampler's performance on the plain defined by the two variables plotted against each other. It is also possible to include the joint probability density of the samples in the pairs plot. As a result, the shape of the limiting stationary distribution of the corresponding Markov chain appears.

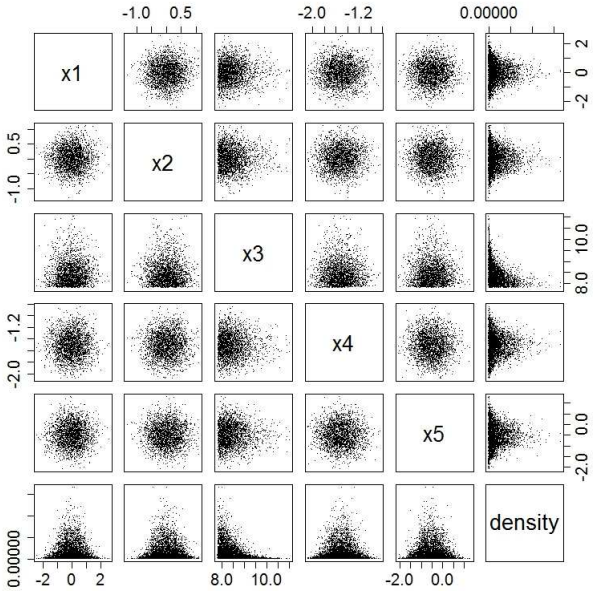


Figure 2.25: Pairs plot for the 5th limit state for the 5-dimensional case

In figures 2.21 to 2.24, the pairs plots of the first case (2-dimensional) for limit states 1 to 4 are shown, the corresponding figure for limit state 5 can be found in appendix A.3, figure 2.24. We can see that the sampler is performing well because the samples are sufficiently dispersed. The line observed on the pairs plots, is the threshold level value, under which, the algorithm is not allowed to sample. This conditioning on the limit states, produces an 'acquired' correlation between the samples. As moving to higher limit states, it can be observed, that more samples lie in areas with

lower density values because the conditioning on the subsets forces them to depart from their initial probability distributions and move to higher conditional levels. The higher the threshold level value, the lower the probability of a point to lie beyond that value. For more details about pairs plot, see [13].

In figure 2.25, pairs plot of the second case study (5-dimensional) for the 5^{th} conditional level are shown indicatively. Here the line of the threshold level value does not appear. The reason is that in this case, the functional value depends on 5 input variables and thus, it is not possible to have a full overview of the samples of a 5-dimensional problem in a 2-dimensional plane. The pairs plots of the other conditional levels can be found in appendix A.3 figures A.6 to A.9.

2.5.3 Comparison of Subset Sampling with standard MCS

In this subsection, the results of an efficiency test of the subset sampling method in comparison to the standard Monte Carlo simulation, are demonstrated. The two case studies introduced in subsection 2.5.2, were used to check the efficiency of the two methods in the calculation of the failure probability. The target limit state of the 2-dimensional case, was reduced from 50 to 43, so that the probability of failure is of the class of 10^{-5}.

For that purpose, ten runs for each algorithm for each case for different total number of iteration steps were performed in order to specify the range of the results and thus check the accuracy of the methods. All the runs were performed for a total number of 10^4, $5 \cdot 10^4$, 10^5, $5 \cdot 10^5$, 10^6, $5 \cdot 10^6$ and 10^7 iterations.

In figure 2.26, the calculated probability of failure is plotted against the number of iterations for the 2-dimensional case, while in figure 2.27 the range of the calculated P_F is denoted for the same plot.

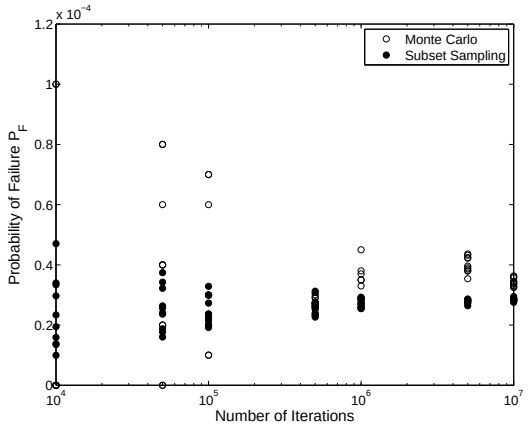


Figure 2.26: Scatter plot of P_F against the total number of iterations for the 2-dimensional case

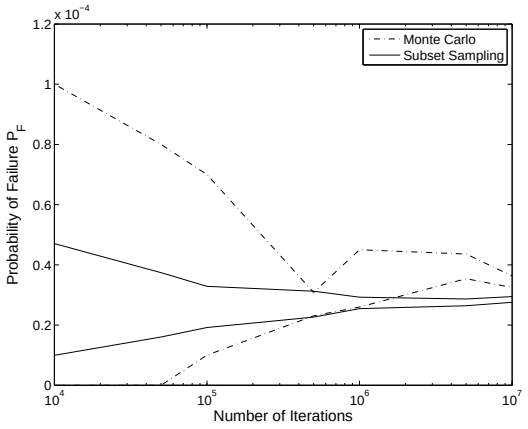


Figure 2.27: Range of P_F against the total number of iterations for the 2-dimensional case

It is obvious that the Monte Carlo method is incapable of yielding an approximation of the failure probability accurately for a number of iterations less than $5 \cdot 10^5$, because the failure probability we try to calculate is of the class of 10^{-5} and it needs more that 10^{-5} iterations to achieve acceptable accuracy, see table 2.1. On the other hand, the subset sampling method, yields significantly better estimation of P_F than MCS for all the number of total iterations, even for 10^4 samples, the estimation is pretty accurate. It should be noted, that the two algorithms converge to different values of P_F, due to imperfections on the complicated subset sampling algorithm. Even so, the values are very close to each other and the accuracy is still sufficient.

Figure 2.27, provides a more representative view of the difference in the accuracy of the two methods. It proves that the subset sampling method is more efficient than MCS for calculating small failure probabilities, because the range of the resulting P_F for subset sampling is fully contained in the range of MCS for all the total numbers of simulation runs.

In figures 2.29 and 2.28, the same plots stand for controlling the efficiency of the two methods in comparison, but for the 5-dimensional case this time.

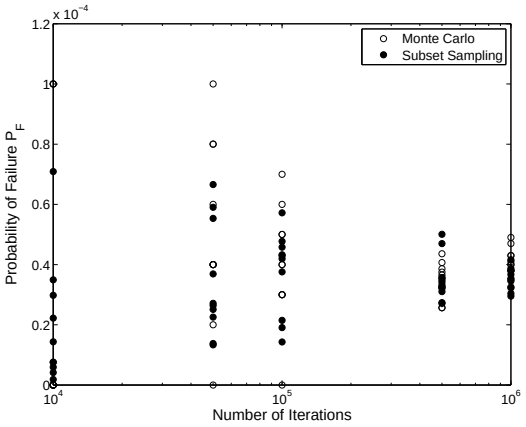


Figure 2.28: Scatter plot of P_F against the total number of iterations for the 5-dimensional case

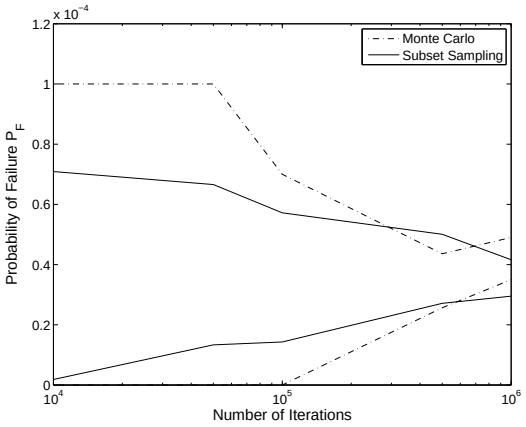


Figure 2.29: Range of P_F against the total number of iterations for the 5-dimensional case

It can be observed, that the efficiency of subset sampling decreases for increasing number of dimensions. The complexity of the subset sampling algorithm and the strong dependency of the method on the correlation between the input variables, makes it insufficient for very high dimensional problems. This is the only limitation of subset sampling method observed in the present project, at least in contrast to the standard MCS. In MCS, the correlation between the input variables, is not affecting the efficiency of the algorithm that much, rather influencing the statistical accuracy.

2.6 UNCERTAINTY ANALYSIS TOOL 'WINFUZ'

'Winfuz', is a multi component library, capable of performing data analysis, as well as polymorphic uncertainty analysis with emphasis to the latter. It provides the possibility of assigning fuzzy and random characteristics on the system's uncertainties and process the corresponding analysis, i.e. fuzzy analysis, stochastic analysis and combinations of them simultaneously. All the types of analyses, 'Winfuz' is capable of performing, are carried out with the aid of the deterministic fundamental solution. The deterministic fundamental solution, can be a simple function the so-called *objective function*, or a finite element model. In the case of a simple objective function, it can be represented as the difference of the applied loading (S) minus the available resistances (R) and it may result from a static or dynamic solution of the structural system.

The present project work, was engaged with parts of both fuzzy analysis and stochastic analysis components of 'Winfuz', and assigned some improvements. The fuzzy analysis component, is based on the α-*level optimization* technique, which requires a reliable multivariate, global optimization subroutine (see chapter 3). Concerning the stochastic analysis component, the drawback of Monte-Carlo simulation in calculating small failure probabilities was to be confronted. For that reason, the *subset sampling* method was applied

2.6.1 Fuzzy and fuzzy stochastic analysis

Fuzzy stochastic analysis, is an appropriate computational approach for processing uncertain data described by the uncertainty model fuzzy randomness (see subsection 1.4.3). The formal description of the model fuzzy randomness, is not suitable to treat uncertain parameters of usual structural problems. For that reason, a more suitable form for describing such uncertain parameters, the so-called α-*discretization*, was introduced by [17] and [18].

fuzzy analysis

The numerical simulation with consideration of fuzzy variables, can be formally described by the mapping

$$F_{FA}(d) : \underline{\tilde{X}} \mapsto \underline{\tilde{Z}}. \tag{2.19}$$

According to (2.19), the vector of fuzzy input variables (or the fuzzy input set) $\underline{\tilde{X}}$, in the space of fuzzy input values (x-space), is mapped onto the vector of fuzzy results $\underline{\tilde{Z}}$, in the space of fuzzy results (z-space), with the aid of the crisp analysis algorithm d. Any deterministic fundamental solution, such as static solution or finite element model, may be applied as algorithm d. The numerical simulation, is carried out with the aid of the so-called α-*level optimization* [18].

α-discretization

Fuzzy values may be discretized with the aid of α-level sets (see subsection 1.4.2). The α-level sets A_{i,α_k}, $i = 1, ..., n$ of the fuzzy input values $\tilde{A}_1, ..., \tilde{A}_i, ..., \tilde{A}_n$ form the n-dimensional crisp subspace $\underline{X}_{\alpha_k}$ of the x-space. For $\alpha_k = 0$, the crisp support subspace (see subsection 1.4.2) is obtained. Given two α values α_k and α_i and the corresponding crisp subspaces $\underline{X}_{\alpha_k}$ and $\underline{X}_{\alpha_i}$ with respect to the fuzzy input set $\underline{\tilde{X}}$, the following relationship holds.

$$\underline{X}_{\alpha_k} \subseteq \underline{X}_{\alpha_i} \qquad \forall \alpha_k, \alpha_i | \alpha_k, \alpha_i \in (0, 1],\ \alpha_k \geq \alpha_i \tag{2.20}$$

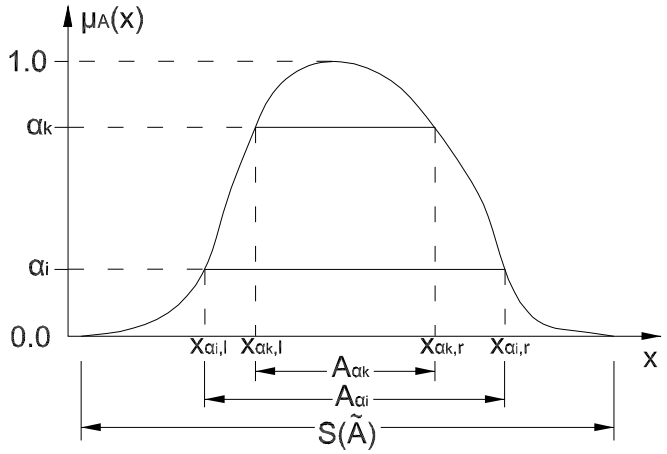


Figure 2.30: α-level discretization

As a result, all subspaces $\underline{X}_{\alpha_k}$ are contained in the support subspace $\underline{X}_{\alpha_k=0}$. If α_k takes all real values in the interval (0,1], the entirety $\underline{X}_{\alpha_k}$ then forms the fuzzy input set $\underline{\tilde{X}}$, where α_k is equal to the membership values of the subspaces $\underline{X}_{\alpha_k}$. If only selected values of $\alpha_k \in (0, 1]$ are assumed, the fuzzy input set $\tilde{X}$ is discretized (see figure 2.30).

α-level optimization

All fuzzy input variables are discretized using the same number of α-levels α_k, $k = 1, ..., r$. For each fuzzy input variable $\tilde{x}_i = \tilde{A}_i$ and for each α-level value α_k, the α-level set A_{i,α_k} is obtained. All the sets A_{i,α_k} form the crisp subspace $\underline{X}_{\alpha_k}$. With the aid of the mapping operator (deterministic fundamental solution) $z = f(\underline{x})$, it is possible to compute elements of the α-level set B_{α_k} of the fuzzy result value $\tilde{z} = \tilde{B}$ on the α-level α_k. The mapping of all elements of $\underline{X}_{\alpha_k}$ yields the crisp subspace Z_{α_k} of the z-space.

Once the largest element $z_{\alpha_k,r}$ and the smallest element $z_{\alpha_k,l}$ of the α-level set B_{α_k} have been found, two points of the membership function $\mu(z) = \mu_B(z)$ are known. If the fuzzy result value is convex, the $\mu(z)$ is completely described. The search of the largest $z_{\alpha_k,r}$ and the smallest $z_{\alpha_k,l}$ elements of the set B_{α_k}, may be formulated as an optimization problem. The objective functions

$$z_{\alpha_k,r} = f(x_1, ..., x_n) \Rightarrow max|(x_1, ..., x_n) \in \underline{X}_{\alpha_k} \tag{2.21}$$

$$z_{\alpha_k,l} = f(x_1, ..., x_n) \Rightarrow min|(x_1, ..., x_n) \in \underline{X}_{\alpha_k} \tag{2.22}$$

must be satisfied. The requirements $(x_1, ..., x_n) \in \underline{X}_{\alpha_k}$ represent the restrictions of the optimization problem.

The equations (2.21) and (2.22) are satisfied by the optimum point $\underline{x}_{opt}$. Precisely two optimum points in the crisp subspace $\underline{X}_{\alpha_k}$ are obtained for each α-level α_k which subsequently form the crisp subspace Z_{α_k} which defines the fuzzy result $\tilde{z}$ (see figure 2.31). The optimization task according to equations (2.21) and (2.22) for all α-levels α_k is referred to as α-level optimization. It should be noted that the α-level optimization is capable of computing more than one fuzzy results $\tilde{z}_j$, despite the fact that only the case of one fuzzy result was explained in the latter. The optimization procedure is described analytically in chapter 3.

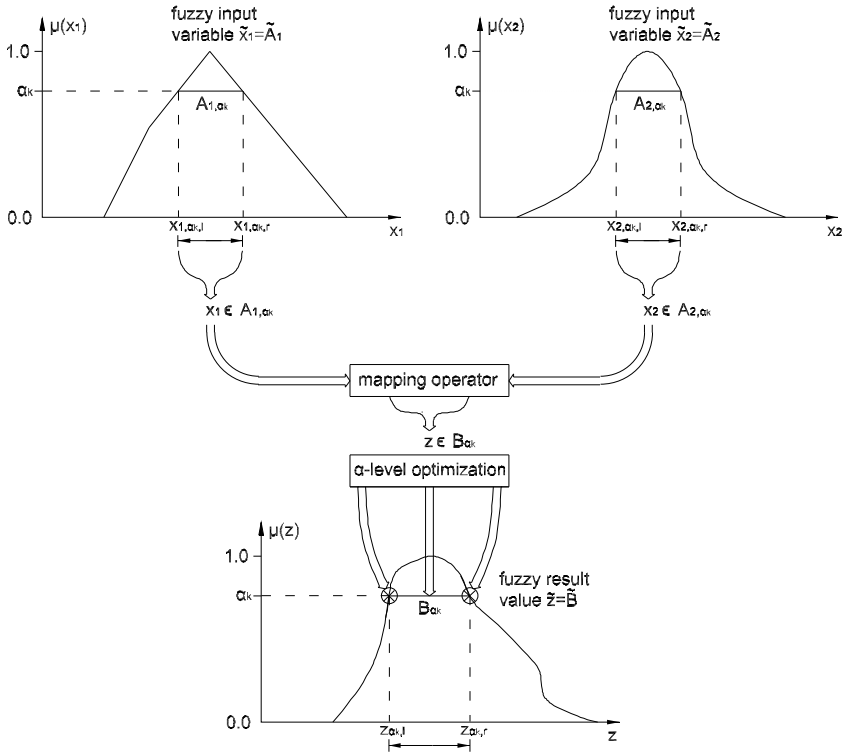


Figure 2.31: α-level optimization, mapping of the fuzzy input variables $\tilde{x}_1$ and $\tilde{x}_2$ onto the fuzzy result value $\tilde{z}$ by mapping all the α-level sets A_{i,α_k} onto the α-level set B_{α_k}

fuzzy stochastic analysis

The concept of the fuzzy stochastic analysis, is carried out by the combination of fuzzy analysis and stochastic analysis with respect to the deterministic fundamental solution as shown in figure 2.32. This is, for every α-level α_k, and for every fuzzy input variable $\tilde{x}_i$, the bounds $x_{i,\alpha_k,l}$ and $x_{i,\alpha_k,r}$ of the α-level sets A_{i,α_k} are obtained. These bounds, define the permissible domain $\underline{X}_{\alpha_k}$, the subspace of the x-space, in which the optimum points $z_{\alpha_k,l}$ and $z_{\alpha_k,r}$ are being searched. In the pure fuzzy analysis, the mapping of each sampled point[8] in the x-space, onto the z-space, is carried out with the aid of the deterministic fundamental solution directly. In the fuzzy stochastic analysis however, where fuzzy random variables[9] are used, the process is more complicated.

In the optimization procedure, each sampled point of the x-space, defines a different probability density function, according to which a stochastic simulation is carried out. The stochastic simulation, determines a specified result value e.g. probability of failure P_F, with the aid of the deterministic fundamental solution. As a result, the fuzzy analysis does not use the deterministic fundamental solution straightforward, but indirectly through the stochastic solution. The final result of the fuzzy stochastic analysis, is a fuzzy value, e.g. fuzzy probability of failure $\tilde{P}_F$. It should be noted, that for the stochastic solution, the Monte Carlo method or the subset sampling method introduced in the present chapter, may be used.

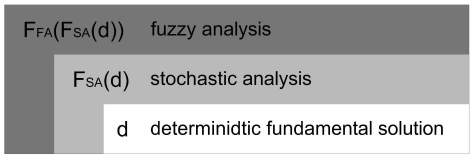


Figure 2.32: Fuzzy stochastic analysis (FSA)

2.6.2 Implementation in 'Winfuz'

The developed subset sampling algorithm was implemented into the uncertainty analysis tool 'Winfuz', in order to improve the stochastic solution which was carried out by the only option of Monte Carlo simulation (MCS). For the implementation, four new integer parameters were required in additon to the existing parameters of the subroutine 'MCS', which were kept unchanged.

These parameters were introduced as 'ipara(11)','ipara(12)','ipara(13)' and 'ipara(14)'. The first, 'ipara(11)', defines the type of dependency between subsequent samples. It thus specifies, the proposal density function and takes the values 1 and 2 for uniform and normal density function respectively. The second, 'ipara(12)', defines the proposal radius (see sections 2.3.4 and 2.4.3) of the proposal density function. If its value is 0, the algorithm uses the standard deviations of the distribution of each input variable for proposal radius. Else, the proposal radius is user specified and a set of new parameters of type real, is introduced. These real parameters are as many as the dimensions (input variables) of the problem and are denoted as 'rpara(nv)'. The next integer parameter, 'ipara(13)', is the number of subsets (nls)[10] and finally, 'ipara(14)', is the number of Markov chains per subset (nmc).

The definition of the parameters required by the new algorithm, was carried out using the standard data model of 'Winfuz', which is used to formally define parameters of the algorithm according to the specified data structure. The express-G representation of the data structure of 'Winfuz' was derived from [9] and is given in the appendix A.6.

[8] During the optimization procedure, points in the subspace $\underline{X}_{\alpha_k}$ are being sampled in search of an optimum.

[9] In an engineering problem, a fuzzy random variable is defined by its probability density function, the parameters of which, are fuzzy values.

[10] In the number of subsets, the final failure event has not taken into account as a subset. As a result, the algorithm is not effected but the real number of subsets is nls+1.

The subset sampling algorithm initializes when the subroutine 'MCMC' is called. The name 'MCMC' was assigned to the subset sampling subroutine because the method is based on the Markov chain Monte Carlo simulation method (MCMC). The subroutine is called through the subroutine 'SA' (Stochastic Analysis). The subroutine 'SA', contains the path for the assignment of values in the subset sampling algorithm parameters. The new parameters are user specified, but default values are assigned to them as standard settings through the Fortran module 'datamanagement'. The standard settings are shown in figure 2.33.

The analysis procedure with 'Winfuz', is carried out by the so-called *calculation points*. A calculation point is a set of definitions concerning the input variables, the result variables, the parameters and the preferences of the analysis, as well as the sequence of different analysis procedures (fuzzy analysis, stochastic analysis etc.) that may be required. For the definition of a calculation point, a GUI (Graphical User Interface) window is used. Within this window, the user can define all the required parameters of the analysis such as the α-cuts (α-levels α_k) of the fuzzy analysis, the input parameters of the optimization task (see chapter 3 and appendix B.1), the type of the stochastic analysis algorithm (MCS or MCMC) and its parameters, as well as the type of random numbers (see section 1.4.1) required.

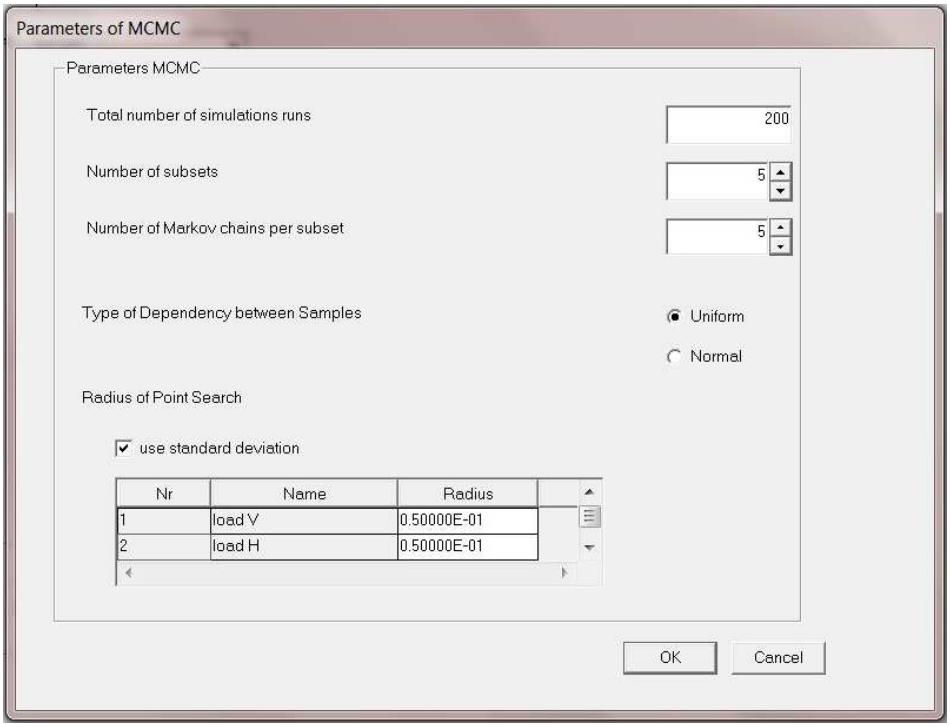


Figure 2.33: GUI dialog for the specification of the subset sampling algorithm parameters

For the implementation of the subset sampling algorithm (subroutine 'MCMC') into 'Winfuz', a new GUI dialog shown in figure 2.33 was designed in order to make the specification of the algorithm's parameters more convenient. The 'Total number of simulation runs', refers to the number of runs within the algorithm for all the subsets together with the initial MCS. The initial MCS and the simulation in each subset, runs *mc* times, hence the total number of runs is $(nls + 1) * mc$, where *nls* is the number of subsets, specified in the second line of the GUI dialog. Within each subset, several Markov chains develop. The number of the Markov chains (*nmc*) is also specified in that window. Moreover, it is possible to choose different types of dependency between subsequent samples, under the title 'type of dependency between samples' the options of uniform and normal proposal density functions are available. Finally, the proposal radius can be equal to the standard deviation of the input variables, or user specified under the title 'Radius of Point Search'.

As implied by the standard settings, by default, 'ipara(13)'='nls'=5, 'ipara(14)'='nmc'=5, the uniform

distribution is used as proposal density function, i.e. 'ipara(11)'=1 and the standard deviations as proposal radius, i.e. 'ipara(12)'=0. In that case ('ipara(12)'=0), the table below the 'use standard deviation' box is inactive. If the box is unchecked, then 'ipara(12)' becomes 1 and the user should specify the proposal radius by typing it in the corresponding position of this table. Then these values are transferred to the parameter vector 'rpara(nv)' which is now allocated.

A small numerical example of pure stochastic analysis using both algorithms was carried out. The resulting histograms of the samples, derived directly from 'Winfuz', are shown indicatively in figures 2.34 and 2.35. In subsection 2.6.3, a detailed numerical example of fuzzy stochastic analysis is being presented.

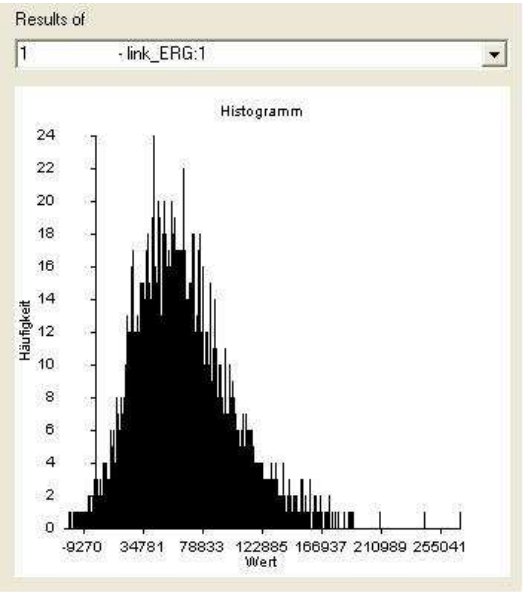


Figure 2.34: Histogram of MCS samples produced by 'Winfuz'

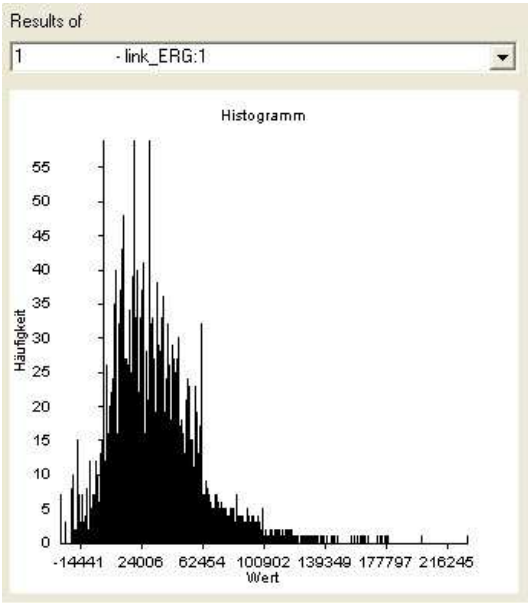


Figure 2.35: Histogram of subset sampling samples produced by 'Winfuz'

The difference in the two histograms is ascribed to the fact that in standard MCS the samples are independent and distributed according to their initial PDFs. In contrast, in the subset sampling method, the samples are gradually forced to depart from their initial PDFs in addition to their dependency between each other. This is the reason why the samples generated by the subset sampling have smaller values than the ones of the MCS, note that in this example, underflow of the limit value has been assumed as failure. In the case of overflow, the subset sampling samples would be larger than the ones of the MCS. Moreover, in the subset sampling histograms, it can be observed, that some values have extremely higher frequencies than others. This is explained by the fact that they were cases were the acceptance ratio was small for many sequential iterations, and thus many repeated values produced. This is a common phenomenon in MCMC methods, and if this repeatability is not extremely dense, the performance of the method is not affected.

2.6.3 Numerical example with 'Winfuz'

The shown truss (figure 2.6.3) is made of quadratic steel S235 JR truss members. The loads V and H, as well as the yield stress f_y of the steel are treated as fuzzy random variables and the calculation of the fuzzy probability of failure $\tilde{P}_F$ is requested.

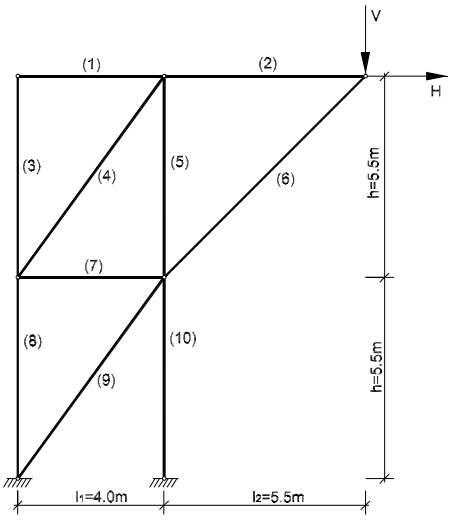
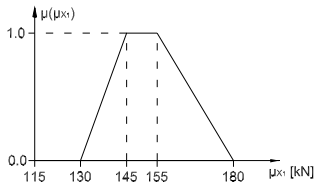
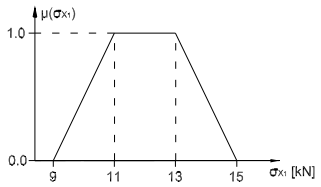


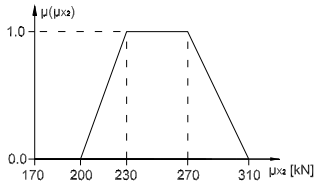
Figure 2.36: System and loads

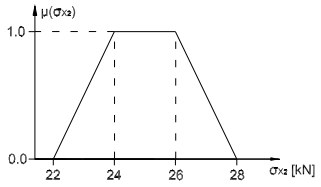
- **Loads**
  The load V is distributed according to the normal distribution $V \sim N(\tilde{\mu}_{X_1}, \tilde{\sigma}_{X_1})$ with the fuzzy parameters $\tilde{\mu}_{X_1}$ and $\tilde{\sigma}_{X_1}$ defined by the membership functions:



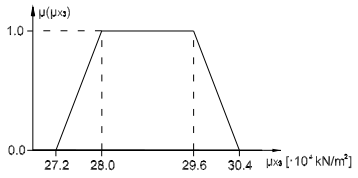


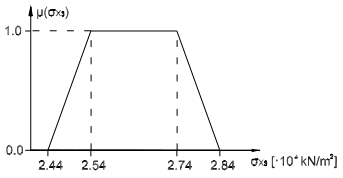
  The load H is distributed according to the normal distribution $H \sim N(\tilde{\mu}_{X_2}, \tilde{\sigma}_{X_2})$ with the fuzzy parameters $\tilde{\mu}_{X_2}$ and $\tilde{\sigma}_{X_2}$ defined by the membership functions:





- **Material**
  The yield stress of the steel f_y is distributed according to the log-normal distribution $f_y \sim LN(\tilde{\mu}_{X_3}, \tilde{\sigma}_{X_3}, x_{0,X_3})$ with the fuzzy parameters $\tilde{\mu}_{X_3}$ and $\tilde{\sigma}_{X_3}$ defined by the membership functions:





  and the location parameter x_{0,X_3} having the deterministic value of $x_{0,X_3} = 19.9 \cdot 10^4 kN/m^2$. The young modulus E has also deterministic value $E = 2.10 \cdot 10^8 kN/m^2$.

- **Cross sections**

truss member	$t[mm]$	$A[\cdot 10^{-3} \cdot m^2]$	$I[\cdot 10^{-5} \cdot m^4]$
(1),(2),(3),(5),(6),(7),(8),(9)	6.3	3.77	1.46
(4)	8.0	4.70	1.78
(10)	10.0	5.74	2.10

Table 2.2: cross section properties of truss members

Firstly the truss system was solved for the static loads V and H and the axial force F_i of every element i was determined. The objective function was constructed then, with the use of the calculated forces F_i, the yield stress of the steel f_y and the cross sectional areas A_i shown in table 2.6.3.

$$f_{obj}(V, H, f_y) = f_y - \max\left(\frac{F_i(V, H)}{A_i}\right) \qquad for \;\; i = 1, ..., 10 \tag{2.23}$$

The term f_y represents the resistance R of the system, while $\frac{F_i(V,H)}{A_i}$ which is the axial stress of the i-th member, represents the applied actions. Failure occurs if one member fails and this is the reason why only the member with the maximum axial stress participates in the objective function.

It is thus ensuing, that the failure probability, is being calculated with the aid of the condition that failure occurs if $f_{obj} < 0$.

For the fuzzy analysis, 3 α-cuts $\alpha_{1,2,3} = 0.0, 0.5, 1.0$ where used in the α-level optimization. For the stochastic analysis, both available algorithms (MCS, subset sampling) were used in two different cases for comparison reasons. The following parameters were used for the stochastic analysis.

- Monte Carlo simulation
  number of simulation runs = 40000

- Subset sampling using Markov chain Monte Carlo simulation
  total number of simulation runs = 40000
  number of subsets = 4
  number of Markov chains per subset = 1
  type of dependency between samples = Normal
  radius of point search = standard deviation

The fuzzy result, which is the fuzzy probability of failure $\tilde{P}_F$ is shown in figures 2.37 and 2.38.

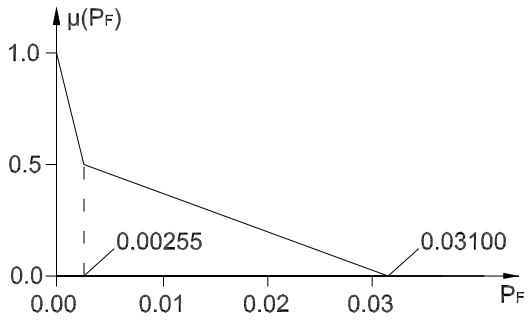


Figure 2.37: Membership function of the fuzzy probability of failure $\tilde{P}_F$ with MCS for the stochastic solution.

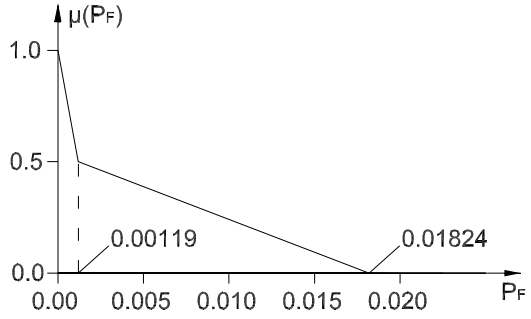


Figure 2.38: Membership function of the fuzzy probability of failure $\tilde{P}_F$ with subset sampling for the stochastic solution.

3 EVOLUTIONARY OPTIMIZATION

The term evolutionary optimization, stands for an optimization technique processed by an Evolutionary Algorithm (EA). Evolutionary algorithms, is a class of stochastic optimization methods that simulate the simplified human view of the natural evolution process. These methods are proven to be a powerful and robust search mechanism although they implement a generally simple scheme [27].

3.1 INTRODUCTION

The solution of an arbitrary optimization problem is generally described in terms of a *decision vector* $(x_1, x_2, ..., x_n)^T$ in the *decision space* $\mathbf{X}$ which is a subspace of $\mathbb{R}^n$ including the possibility of assuming specific bounds and constraints. A function $\boldsymbol{f} : \boldsymbol{X} \rightarrow \boldsymbol{Z}$, evaluates the quality of a specific solution by assigning it to an *objective value z* in the *objective space* $\mathbf{Z} \in \mathbb{R}$ [27]. Naturally, f, is often denoted as the *objective function*. The optimization problem can be described by equation (3.1).

$$\mathbf{z}_{opt} = f(x_1, x_2, ..., x_n) \Rightarrow Max|(x_1, x_2, ..., x_n)^T \in \mathbf{X} \tag{3.1}$$

Without loss of generality, maximization has assumed. Minimization is possible to be used instead.

In such an optimization problem, a solution $\mathbf{x}^{(1)} \in \mathbf{X}$ is better than another solution $\mathbf{x}^{(2)} \in \mathbf{X}$ if and only if $\mathbf{z}^{(1)} > \mathbf{z}^{(2)}$ where $\mathbf{z}^{(1)} = f(\mathbf{x}^{(1)})$ and $\mathbf{z}^{(2)} = f(\mathbf{x}^{(2)})$. Although several optimal solutions may exist in decision space, they are all mapped to the same objective value.

A popular optimization technique, is the *evolution strategy* which uses natural problem-dependent representations, and primarily mutation and selection, as search operators [6]. The operators are applied in a loop. An iteration of the loop is called a generation and the central point around which an improvement is being searched within the iteration, is referred to as the parent point. The sequence of generations is continued until a termination criterion is met. As far as real-valued search spaces are concerned, mutation is normally performed by adding a normally distributed random value to each vector component. Only if the mutant's (often referred to as offspring) fitness is at least as good as the parent one, it becomes the parent of the next generation. Otherwise the mutant is disregarded. This is a (1 + 1) evolution strategy. More generally, λ mutants can be generated and compete with the parent, called $(1 + \lambda)$ evolution strategy. In $(1 + \lambda)$-ES the best mutant becomes the parent of the next generation while the current parent is always disregarded.

In this chapter, the existing evolutionary optimization library 'evolu' is being analysed, with special attention on the offspring generation procedure, as well as on the behaviour of the algorithm along the boundaries of the permissible search domain. In addition, a modification on the local search procedure and specifically on the shape of the local search subspace, is proven to increase the efficiency of the optimization process.

3.2 THE 'EVOLU' SUBROUTINE

'Evolu' is a Fortran algorithm designed to perform global optimization in service of other uncertainty analysis tools such as 'Winfuz'. The algorithm is based on the idea of the *modified evolution strategy* introduced in [17] and enhanced with two additional capabilities. The first is the possibility to

consider constraints in the decision space X and the second, the capability of multi-processing, i.e. multiple runs are started within one optimization step.

The modified evolution strategy is a combination of evolution strategy, the gradient method and the Monte Carlo method [18], in order to service the demand of a robust optimization technique which is independent of the type and behaviour of the objective function and is capable of reliably finding global optima. The combination of directed and non-directed search techniques is advantageous compared with purely directed search methods regarding the determination of global optima. By making use of the existing information on the behaviour of the objective function, the number of the 'unnecessary' computations of objective functional values, which do not lead to an improvement, is reduced. If the amount of available information is insufficient, random-oriented methods are applied. Moreover, in contrast to the simple evolution strategy, the parent points are included in the optimum search procedure as well.

It should be noted here, that an important aspect in the functionality of 'evolu', is the use of the so-called *pointers* introduced in the new Fortran 90. There are two concepts of variables in Fortran, the static variable, and the dynamic pointer. Whereas a static variable is declared before the program is executed, a pointer can point to any variable during the execution of the program. A static variable is defined in the beginning of the program. Before executing the program a fixed address in the memory is allocated to contain the data of the variable. During the runtime this address does not change and all operations including this variable have to access the same address in the memory. When a pointer is associated to a static variable, it then contains the address in the memory that this variable is saved. This is very useful in complicated routines, because it gives detailed control over complex data structures.

3.2.1 Main features

As an evolutionary algorithm, 'evolu' has the features of *reproduction, mutation, recombination* and *selection*. More specifically, it *reproduces offspring points*, which are actually vectors $\mathbf{x} = (x_1, x_2, ..., x_n)^T \in \mathbf{X}_\alpha \subset \mathbf{X}$, using two possible procedures, where $\mathbf{X}_\alpha \subset \mathbf{X}$, is the pre-defined subspace referred as permissible domain (see α-level optimization in section 2.6). First, by *mutating* some *parent points* or second, by *recombining* different ancestor points from many generations back. This procedure is guided by a *selection* process in the sense that if the generated offspring points are not improvements of their parent points, they are being rejected and new offspring points are generated until improvement is reached. The improved offspring points become now parent points and the procedure continues until no further improvement is possible with respect to a specified level of convergence, or when the predefined maximum number of generations is reached.

The starting points of the optimization, can be user-specified, if for example there are available information about areas that the optimum point is more likely to lie on. In addition to that, the starting point can be determined randomly using Monte Carlo simulation or the Latin Hypercube Sampling method (LHS). If x_i^L and x_i^U are the lower and upper bounds of the permissible domain for dimension i, the case of random-specified starting points is described by equation (3.2).

$$x_i^{[0]} = x_i^L + u_i \cdot (x_i^U - x_i^L) \quad i = 1, 2, ..., n \tag{3.2}$$

Where u_i is a random number realized from $U[0, 1)$ and n is the number of dimensions. The starting points are set as the first parent points, from each of which, different optimization *chains* begin to search. From a parent point, a specified number of offspring points are generated (subroutine 'point_generation') using guided or random-oriented techniques. Subsequently, the value of the objective function for each of the offspring points is being evaluated and the best offspring point becomes the new parent. If none of the offspring points implements an improvement to the objective function, the range of the subspace decreases until the specified convergence tolerance is reached. This means that the current point is accepted as the optimum. The final optimum is chosen as the best of all the chains which started from different areas of the search domain.

It should be noted, that all the old points used in the procedure, are saved together with their functional values in two arrays ('xsave' and 'zsave' arrays) so that they can be reused, i.e. if a point is sampled again, the functional value is not evaluated because it already exists in one of these arrays ('zsave'). It is also possible to use already sampled points that are sufficiently close to a new sampled point and this is controlled by a pre-defined 'proximity radius'. This feature yields better efficiency of the algorithm.

A general flow chart of 'evolu' is demonstrated in figure 3.1 and the analytical explanation of some parts follows. Moreover, in appendix B.1, there is a description on the input, the output and the control variables of 'evolu'.

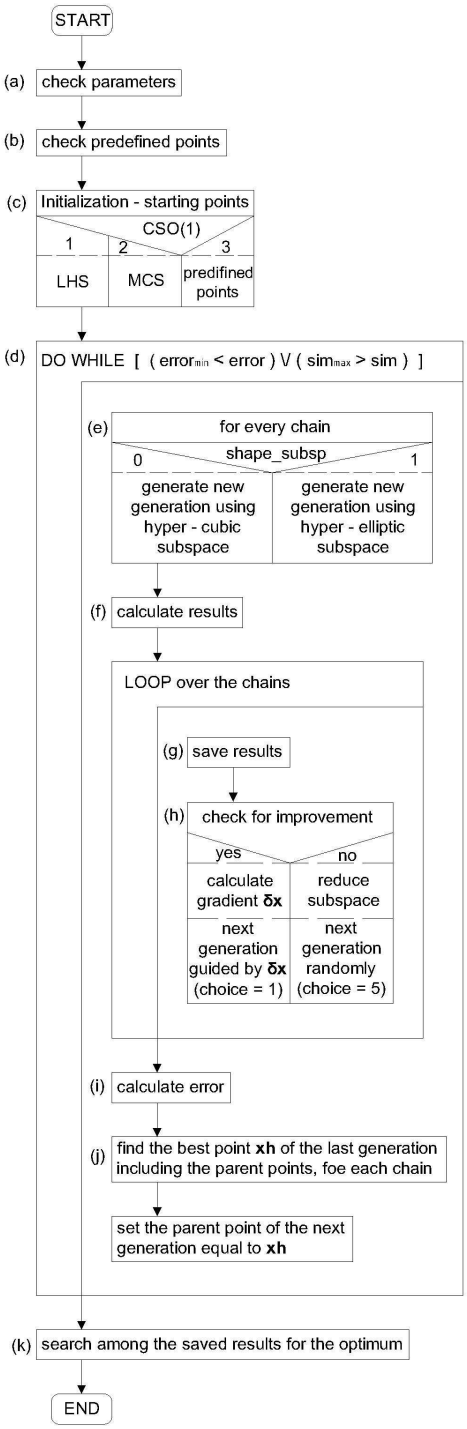


Figure 3.1: General flow chart of 'evolu'

(*a*) The subroutine 'input_editing' is called, which checks the input parameters for validity.

(*b*) The subroutine 'perm_pred_points' is called which extracts the permissible ones from the predefined points.

(*c*) Chooses the type of starting points determination. It can be done using LHS, MCS, predifined points, or some predifined points and the rest using LHS or MCS.

(*d*) Abortion criterion. The algorithm stops if the calculated error is lower than the predefined value $error_{min}$ or if the predefined maximum number of generations sim_{max} is reached. $error_{min}$ is represented in the source code by the variable 'err_def', while sim_{max} by 'maxsim'.

(*e*) The subroutine 'point_generation' is called, which produces one new generation.

Subroutine 'point_generation' is used to generate offspring points with respect to the given parent point. When it is called, it takes as input the parent point and the variable 'choice', and returns 'sapo' offspring points. If 'choice' equals to 1, it samples the first offspring directed by the gradient vector 'delta_x'. Then 'choice' becomes automatically 2. When 'choice' is 2, it samples one corner and automatically changes 'choice' to 3, which means that the opposite corner is being sampled and 'choice' becomes automatically 2 again in order to sample another corner. When all corners have been sampled, 'choice' becomes automatically 4, so the next offspring points are being determined randomly. Finally, if 'choice' is 5, all the offspring points are being determined randomly. This happens when no improvement of the parent has achieved in the previous step. It should be noted, that 'choice' can take only the values 1,2,5 when it is imported in 'point_generation', while within the subroutine it can also take the values 3 and 4 but only automatically. For more details see section 3.2.2.

(*f*) The subroutine 'evolu_det' is called.

Subroutine 'evolu_det' calculates the functional values of the *x* array values, which is used as its input and returns the corresponding *z* array.

(*g*) It saves the points and the corresponding functional values of the new generation in the 'xsave' and 'zsave' array respectively.

(*h*) It searches for the best improvement among the offspring points of the new (current) generation, including the parent point (according to the modified evolution strategy). As the algorithm loops over the offspring points, it keeps the best improvement so far in the variable 'z_compare'. If improvement has achieved, it calculates the gradient vector δx which equals to the variable 'delta_x', sizes it by the factor 'fac' in order to avoid very short gradient vectors and uses it to determine the first offspring point of the next generation ('choice' becomes 1). If no improvement has achieved in the current generation, it reduces the size of the local search domain, as well as the distance in which old points are re-used, and determines all the points of the next generation randomly ('choice' becomes 1).

(*i*) The subroutine 'opt_error' is called.

Subroutine 'opt_error' determines one error for every chain, using the equation (3.3).

$$error = \frac{|zmin_k - zmax_k|}{\frac{|\mathbf{Z}_{min}| + |\mathbf{Z}_{max}|}{2}} \tag{3.3}$$

Where '$zmin_k$' and '$zmax_k$' are obtained from the array 'zrun', and hence they are the minimum and the maximum functional values of the current local search for the current chain indicated by *k*. Thus, it is obvious that they are different from chain to chain. On the other hand, $\mathbf{Z}_{min}$ and $\mathbf{Z}_{max}$, are obtained as the minimum and the maximum value of the 'zsave' array, and hence they are global for all points, for all chains so far. Finally, it calculates the average error of all chains and saves it in the last position ('nchain'+1) of the 'error' array, in order to be used in the abortion criterion.

(*j*) The subroutine 'optimum' is called, in order to find the optimum among the points of the current generation including the parent points, separately for each chain.

Subroutine 'optimum' has an array of x values and their corresponding functional values z as input, and returns the same arrays but sorted from the most to the least optimum with respect to the z values. As a result, in the first place of these arrays is the optimum point and its functional value.

(k) The subroutine 'optimum' is called again, in order to find the optimum among all the saved points, which is the calculated global optimum. The optimum point and its functional value, are returned as 'xerg' and 'zerg' respectively.

3.2.2 Offspring generation

As mentioned before, the offspring points are generated in the proximity of the parent points by a mutation of their properties according to the random principle or guided using available information. This mutation is bounded by the maximum max_d_i and the minimum min_d_i distance for dimension i, which are proportional to the size of the permissible domain by means of the factors c_1 and c_2. As a result, the subspace of search, the so called *local search domain*, is defined and it holds

$$max\_d_i = c_1 \cdot (x_i^U - x_i^L) = max|x_i^{[q+1]} - x_i^{[q]}| \quad i = 1, 2, ..., n \tag{3.4}$$

$$min\_d_i = c_2 \cdot (x_i^U - x_i^L) = min|x_i^{[q+1]} - x_i^{[q]}| \quad i = 1, 2, ..., n \tag{3.5}$$

for the two quantities, which are user-specified as fractions of the permissible search domain. With a closer look in equations (3.4) and (3.5) it can be observed that the local search domain is the space between a n-dimensional hyper-cuboid defined by the vector **max_d** and another one defined by the vector **min_d**. In the common centre of the two hyper-cuboids is the parent point. In figure 3.2, the simple two dimensional case is demonstrated as zoomed visualization of the real optimization process in figure 3.3.

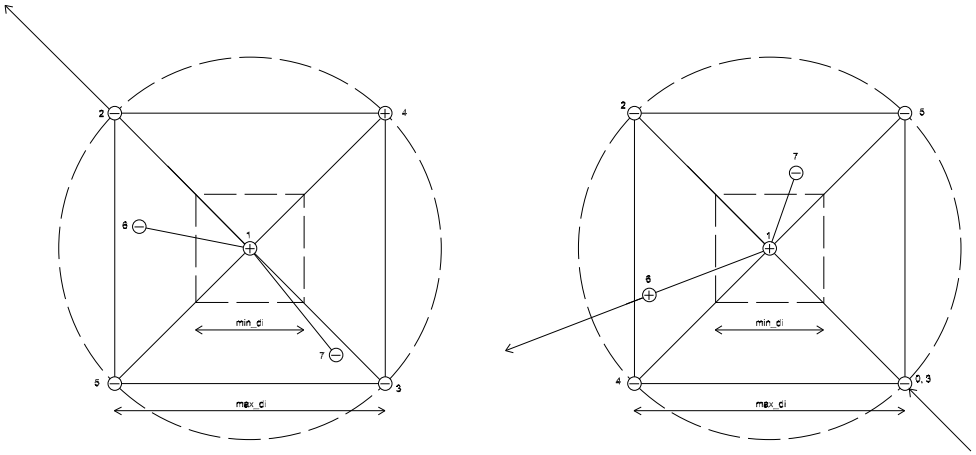


Figure 3.2: Subspace domain with the point generation procedure. **Left:** initial case, the parent point (1) is a starting point. **Right:** the parent point (1), found as an improvement of the previous parent point (0).

In the generation procedure a specified number of sample points (sapo) is being procreated before the evaluation of their functional values. With the initialization of the optimum search, the starting point is set as the first parent point. First the vertices of the hyper-cuboid are being sampled (see figure 3.2).

In a monotonically increasing objective function, improvements of the functional value are located on the vertices of the hyper-cuboid, except from the area around the global optimum, see figure 3.3. In the more general case of a non-monotonically increasing objective function, the vertices are

again being sampled first. This has the meaning that if the function is smooth, i.e. the distance between the local optima is sufficiently larger than max_d_i, improvements again lie on the vertices but they may lead the process to a local optimum. The general optimum search procedure of 'evolu', that tries to gradually improve the functional value of a point by exploring its proximity, can not avoid local optima. To compensate this drawback, different optimization chains explore the permissible domain from different starting points, because this increases the possibility that one of them will not fall to a local optimum.

After the *nc* (*nc* =number of corners) corners have been sampled, the next offspring points $\mathbf{x}^{[q+nc+j]}$ of $\mathbf{x}^{[q]}$ are generated randomly within the local search domain by using the *i*-dimensional vector of random numbers $\mathbf{u} \sim U[0, 1)$ in equation (3.6).

$$\mathbf{x}^{[q+nc+j]} = \mathbf{x}^{[q]} + 2 \cdot (\mathbf{u} - 0.5) \cdot \mathbf{max\_d} \quad 1 \leq j \leq sapo - nc \tag{3.6}$$

Where $\mathbf{max\_d} = c_1 \cdot (\mathbf{x}^U - \mathbf{x}^L)$. For at least one *i* the condition (3.7) should be fulfilled.

$$min\_d_i \leq d_i = |x_i^{[q+nc+j]} - x_i^{[q]}| \quad 1 \leq i \leq n, \ 1 \leq j \leq sapo - nc \tag{3.7}$$

The condition (3.7), ensures that the randomly sampled points, are not contained in the non-permissible inner hyper-cuboid defined by **min_d**. Points are being sampled randomly until the total number *sapo* of offspring points is reached. Thus it holds $nc + j \leq sapo$.

Subsequently, the functional values of all the sampled points including the parent point are evaluated and the best of them that leads to an improvement, is selected to be the next parent point. If an improvement has achieved, a new parent point has been obtained. The algorithm switches to the gradient method and uses the information gained by this improvement to specify the first offspring of the next generation. More specifically, the vector $\boldsymbol{\delta}\mathbf{x}$ in equation (3.8) is being calculated.

$$\boldsymbol{\delta}\mathbf{x} = \{x_i^q - x_{i(previous)}^q | i = 1, 2, ..., n\} \tag{3.8}$$

The first offspring point of the next generation is placed at the position shown in equation (3.9) (figure 3.2 right).

$$\mathbf{x}^{[q+1]} = \mathbf{x}^{[q]} + \boldsymbol{\delta}\mathbf{x} \tag{3.9}$$

Hence the algorithm is trying to follow the ridges of the objective function, a task that will definitely improve the efficiency. The next offspring points are being positioned on the corners as before, until all the corners (vertices of the hyper-cuboid) have been sampled. Then random points are being sampled using equations (3.6) and (3.7) until the total number *sapo* of offspring points is reached.

It should be noted, that the point evaluated by equation (3.9), will be sampled instead of one of the vertices of the hyper-cuboid, which defines the local search domain. As a result, in the case that an improvement found in the previous step, the algorithm samples $nc - 1$ vertices of the hyper-cuboid in the current step and not *nc* as in the initial case where it begins from a starting point. This is not clear in figure 3.2 right, because point 1, which is the best improvement of point 0, has fallen on the corner of the local search domain, and as a result, due to symmetry, point 2 sampled using the gradient vector $\boldsymbol{\delta}\mathbf{x}$, also falls on a corner. Hence, point 2, covers the corner that would not have been sampled in an another occasion.

After the total number *sapo* of offspring points has been sampled, their functional values are being evaluated and if an improvement has achieved, the gradient vector $\boldsymbol{\delta}\mathbf{x}$ is used in equation (3.9) to evaluate the first offspring point of the next generation and the procedure continues as described. If no improvement has found among the *sapo* offspring points, the algorithm uses as parent point the unimproved parent point of the previous step and samples *sapo* random points with respect to equations (3.10) and (3.11) which are a variation of the equations (3.6) and (3.7).

$$\mathbf{x}^{[q+j]} = \mathbf{x}^{[q]} + 2 \cdot (\mathbf{u} - 0.5) \cdot \mathbf{max\_d} \quad 1 \leq j \leq sapo \tag{3.10}$$

Where $\mathbf{max\_d} = c_1 \cdot (\mathbf{x}^U - \mathbf{x}^L)$. For at least one *i* the condition (3.11) should be fulfilled.

$$min\_d_i \leq d_i = |x_i^{[q+j]} - x_i^{[q]}| \quad 1 \leq i \leq n, \ 1 \leq j \leq sapo \tag{3.11}$$

In that case, the unimproved parent point of the previous step coincides with the parent point of the current step. This is a special characteristic of the modified evolution strategy where the parent point participates together with the offspring points in the optimum search.

All the steps are repeated many times until no further improvement is possible. It should be noted, that in one generation the offspring points which stand as candidates for the next parent point, may come from different sampling techniques. For example, in figure 3.2 right, point 0 is the previous parent point, point 1 is the current parent point and point 2 is specified by $\boldsymbol{\delta x}$ using (3.9). Points 3,4 and 5 are positioned on the corners and points 6 and 7 are specified randomly using (3.6) and (3.7). Finally, after the evaluation of the objective functional values, the best offspring was found to be point 6.

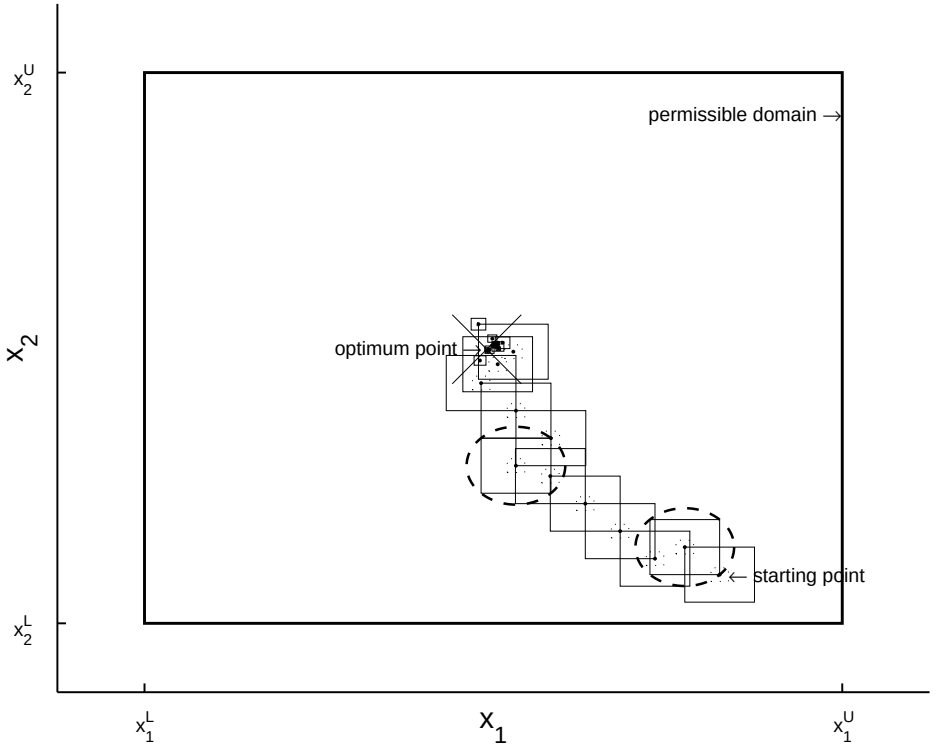


Figure 3.3: Real optimization procedure

A summary of the point generation procedure is shown in the following.

Repeat until the specified convergence or the maximum number of generations is reached

1. *sample one vertex and its opposite until all the vertices of the hyper-cuboid defined by* ***max_d*** *have been sampled.*
2. *sample random points until the total number of points sapo is reached.*
3. *calculate the functional values of all the sapo offspring points including the parent point and check if improvement has achieved. If no, keep the parent point, clear all the offspring points and go to 2.*
4. *Set the best improvement as the new parent point and calculate* $\boldsymbol{\delta x}$ *using* (3.8)*. Position the first offspring point of the new parent point using* $\boldsymbol{\delta x}$ *according to* (3.9)*.*

In the case that the random search (step 2.), does not yield any improvement, the bounds **max_d** and **min_d** of the subspace are being reduced by reducing the factors c_1 and c_2 and the process is

repeated until an optimum is achieved. The case that the offspring generation follows an improvement, is depicted in figure 3.2 right.

A real optimization procedure is shown in figure 3.3, the large x marker in the middle of the plot denotes the optimum point found. Moreover, a structogram of the subroutine 'point_generation', which is responsible for generating the offspring points of one generation when called, is shown in figure 3.4.

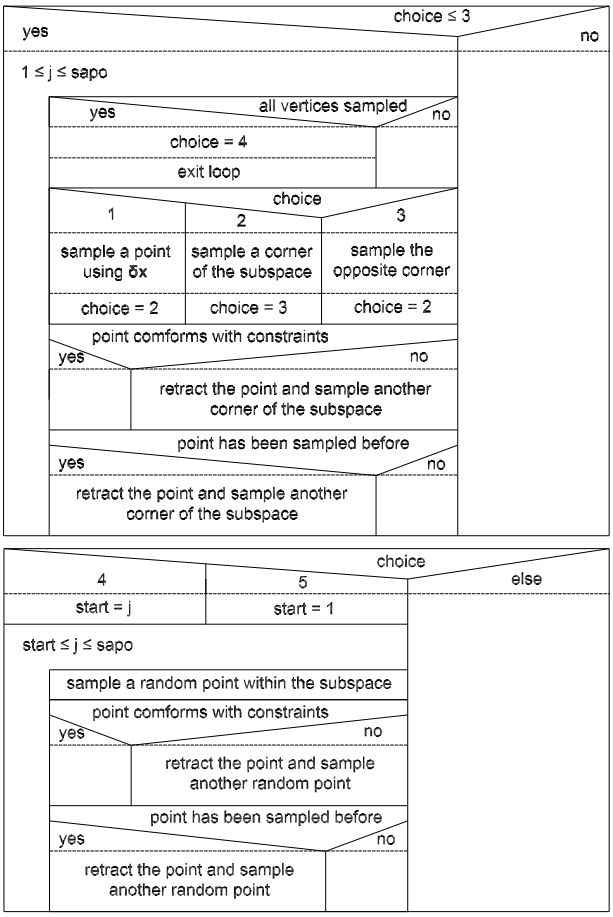


Figure 3.4: Structogram of the subroutine 'point_generation'

3.2.3 Behaviour at the boundaries

On the boundary of the permissible domain, the search of the algorithm distinguishes between random sampled points and directed specified points. If a randomly placed point $\mathbf{x}^{[q+1]}$, does not satisfy all the restrictions $(x_1, x_2, ..., x_n)^T \in \mathbf{X}_\alpha = [\mathbf{x}^L, \mathbf{x}^U]$, it is positioned on the boundary of the permissible domain by means of the equations (3.12) and (3.13).

$$x_{i\,(correct)}^{[q+1]} = x_i^L \quad for \;\; x_i^{[q+1]} < x_i^L \tag{3.12}$$

$$x_{i\,(correct)}^{[q+1]} = x_i^U \quad for \;\; x_i^{[q+1]} > x_i^U \tag{3.13}$$

If the corrected point $x_{i\,(correct)}^{[q+1]}$ does not comply with the distance min_d_i at least for one i, according to equation (3.7), this point is being rejected. The coordinate search is continued with the random determination of $\mathbf{x}^{[q+1]}$ (equations (3.6) and (3.7)).

If a directed placed point $\mathbf{x}^{[q+1]}$, does not completely fulfil the restrictions $(x_1, x_2, ..., x_n)^T \in \mathbf{X}_\alpha$, it is also placed on the boundary according to equations (3.12) and (3.13), see also figure 3.5. Subsequently, the least relative distance is calculated using equation (3.14) to determine if and which coordinate $x_i^{[q+1]}$ falls short of the minimum distance min_d_i.

$$d_{rel,min} = max(\frac{d_i}{min\_d_i}) \quad for \;\; i = 1, ..., n \tag{3.14}$$

Where d_i is the distance between parent and offspring point for dimension i. If $d_{rel,min} \geq 1$, no coordinate $x_i^{[q+1]}$ falls short of the distance bound min_d_i and the point $x_{i\,(correct)}^{[q+1]}$ has been found. If $d_{rel,min} < 1$, the d_i values that do not fulfil the min_d_i condition, are proportionally increased by the factor

$$c_3 = \frac{1}{2}(\frac{1}{d_{rel,min}} + \frac{1}{d_{rel,max}}) \tag{3.15}$$

where

$$d_{rel,max} = max(\frac{d_i}{max\_d_i}) \quad for \;\; i = 1, ..., n \tag{3.16}$$

is the minimum relative deficit below the maximum distance max_d_i. The point $\mathbf{x}_{(correct)}^{[q+1]}$ is determined from equation (3.17).

$$\mathbf{x}_{(correct)}^{[q+1]} = \mathbf{x}^{[q]} + c_3 \cdot (\mathbf{x}^{[q+1]} - \mathbf{x}^{[q]}) \tag{3.17}$$

The coordinates $x_i^{[q+1]}$, that are already boundary values of $\mathbf{X}_\alpha$, are retained and no longer considered in the equation (3.17). Only the coordinate $x_{i\,(correct)}^{[q+1]}$ which corresponds to $d_{rel,min}$ is recorrected, it is placed exactly between min_d_i and max_d_i (equations (3.15) and (3.17)), see also figure (3.5).

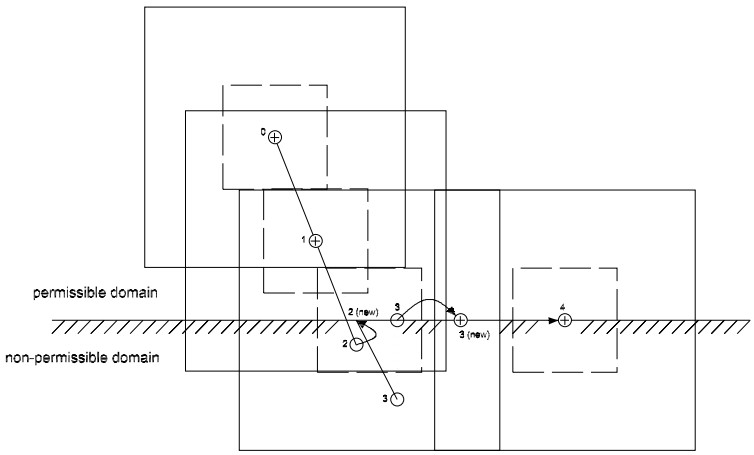


Figure 3.5: Behaviour of 'evolu' along the boundary of the permissible domain

After the correction, if the point $\mathbf{x}_{(correct)}^{[q+1]}$ lies in a corner of the permissible domain $\mathbf{X}_\alpha$, a check is no longer necessary for the distance bounds, and the optimization continues. Other wise the distance bounds min_d_i and max_d_i are rechecked and if all conditions are satisfied, $\mathbf{x}_{(correct)}^{[q+1]}$ is evaluated or the procedure is repeated again. If only the condition min_d_i is not fulfilled, only the steps in equations (3.14) through (3.17) are repeated.

The procedure that the algorithm follows if a new sampled point falls out of the bounds of the permissible domain, is described by the structogram in figure 3.6. The corresponding source code can be found in appendix B.2.

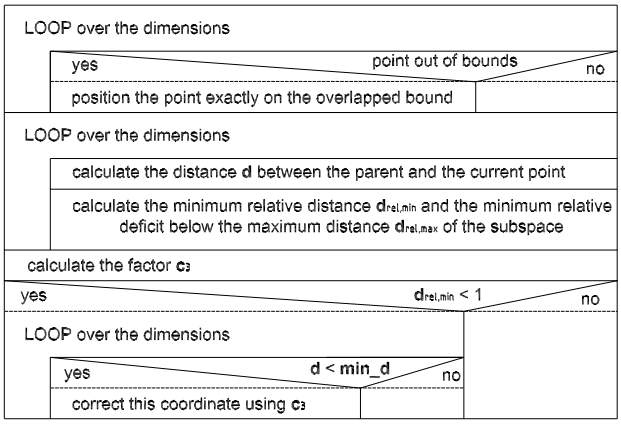


Figure 3.6: Structogram of the behaviour at the boundaries

3.3 COMPARISON BETWEEN HYPER-CUBIC AND HYPER-ELLIPTIC SUBSPACES

3.3.1 Proposed modification of subspace's shape

For the improvement in the efficiency of the existing optimization algorithm, a modification in the shape of the local search domain was proposed. The proposed hyper-elliptic shaped subspace domain was assigned, tested and compared to the hyper-rectangle case. It should be noted here that in the following, the term *n*-sphere, is used instead of the mathematically correct *n*-ball, which is a $n-1$-sphere, i.e. the common sphere in the 3-dimensional space, is a 3-ball or a 2-sphere. Moreover, *n*-ellipse refers to an ellipse in the *n*-dimensional space.

The reason that the *n*-ellipse was chosen as an alternative to the *n*-rectangle, is that the volume of an *n*-sphere which contains a unitary *n*-cube and osculates to its vertices, increases with increasing number of dimensions while the volume of the unitary[1] *n*-cube remains constant (see figure 3.7). On the other hand, the inner sphere, which is contained in the unitary *n*-cube and osculates to its sides, covers a volume that decreases with increasing number of dimensions. For that reason, the outer sphere was chosen instead of the inner because the volume of the inner sphere tends to zero for a large number of dimensions and this will affect negatively the efficiency of the optimization process.

[1] Unitary cube or more generally unitary *n*-cube, is the geometrical object in the *n*-dimensional space that has all its sides (*s*) linear and equal to 1. As a result its volume is also equal to 1 ($V = s^n = 1$)

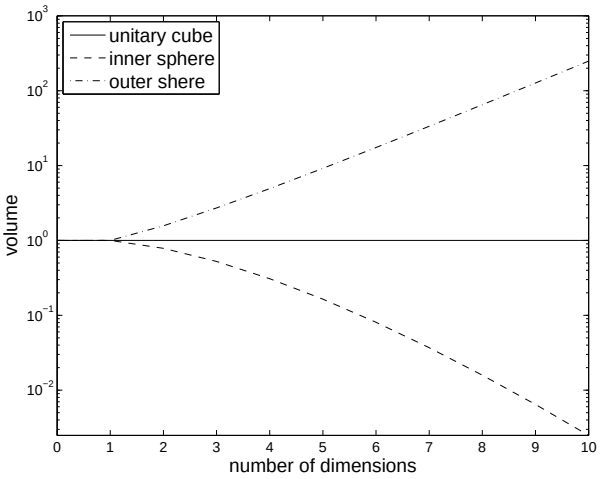


Figure 3.7: Volume of unitary *n*-cube, and its inner and outer *n*-spheres

In figure 3.7, the volume of a unitary (its edges equal to 1) *n*-cube, is plotted together with the volumes of its inner and outer *n*-spheres. The phenomenon of the rapid volume change in high dimensional spaces is one case of the more general phenomenon known as *curse of dimensionality*[2] and is responsible for various types of problems when the number of dimensions is high [3],[4]. In the case of the inner and outer *n*-spheres of a unitary *n*-cube, the curse of dimensionality is caused by the fact that the volume of an *n*-sphere, is proportional to R^n, where R is the radius of the sphere. The inner sphere has a fixed radius of 0.5 (half of the cube's edge), and thus, as $n \to \infty$, $R^n \to 0$. The outer sphere's radius, depends on the number of dimensions n, equals to $\frac{\sqrt{n}}{2}$ and thus, as $n \to \infty$, $R^n \to \infty$. The fact that the radius of the outer sphere depends on n, while the inner sphere's radius does not, is explained by the direction of the calculated radius relatively to the direction of the dimensions.

More specifically, the inner sphere osculates to the sides of the cube, each of which, is always perpendicular to one dimension and as result, the corresponding radius which is assumed orthogonal to the side of the cube, is then parallel to that dimension. On the other hand, the outer sphere osculates to the cube's vertices and thus, the corresponding radius is a diagonal of the cube. The diagonals of an *n*-cube, are never parallel to any of the dimensions and as a result, their length depends on the number of dimensions.

In the figures 3.8 and 3.9, the two cases of the inner and the outer 3-dimensional spheres of the unitary cube, are being illustrated. In figure 3.8, the radius of the inner sphere has a fixed value of 0.5 and is parallel to the dimensions. As a result the volume of that hyper-sphere decreases with increasing number of dimensions. In figure 3.9, the radius of the outer sphere this time, lies on the diagonal of the cube and thus, it can not be parallel to any dimension. As a result, the radius of that sphere depends on n and equals to $\frac{\sqrt{n}}{2}$, where n is the number of dimensions.

[2] The term curse of dimensionality was coined by Richard E. Bellman when considering problems in dynamic optimization.

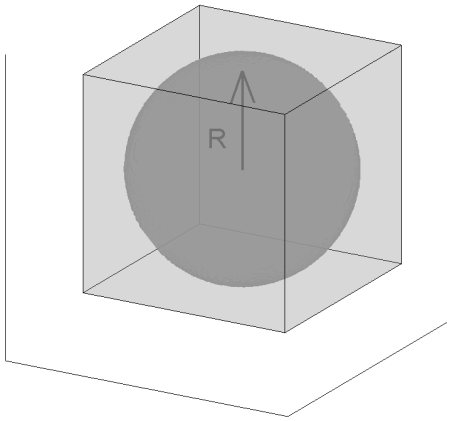


Figure 3.8: Inner Sphere $R = 0.5$

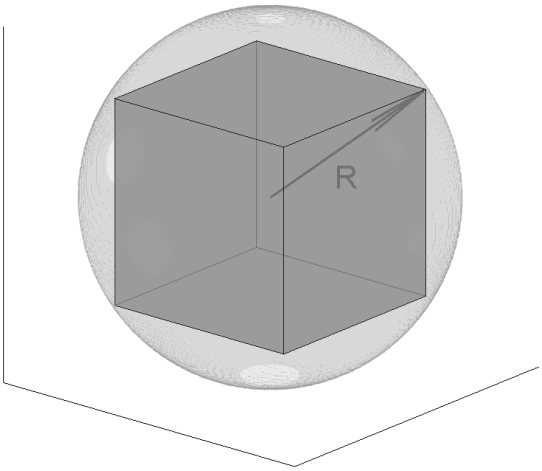


Figure 3.9: Outer Sphere $R = \frac{\sqrt{n}}{2}$

The calculation of the volume of a n sphere of radius R, the formula in (3.18), is used.

$$V_n = \frac{\pi^{n/2}}{\Gamma\left(\frac{n}{2}+1\right)} R^n \tag{3.18}$$

Where $\Gamma(\cdot)$ is the gamma function.

It is now possible to investigate the influence of R and n on the volume of an n-ball. For that reason, the volume was calculated for different values of the two influencing parameters.

Table 3.1: Values of the volume of an n-sphere contained with respect to n and R

$R \downarrow n \rightarrow$	1	2	3	4	5	10	50
0.1	2.00e-01	3.14e-02	4.19e-03	4.93e-04	5.26e-05	2.55e-10	1.73e-63
0.5	1.00e+00	7.85e-01	5.24e-01	3.08e-01	1.64e-01	2.49e-03	1.54e-28
1	2.00e+00	3.14e+00	4.19e+00	4.93e+00	5.26e+00	2.55e+00	1.73e-13
1.2	2.40e+00	4.52e+00	7.24e+00	1.02e+01	1.31e+01	1.58e+01	1.57e-09
1.5	3.00e+00	7.07e+00	1.41e+01	2.50e+01	4.00e+01	1.47e+02	1.10e-04
2	4.00e+00	1.26e+01	3.35e+01	7.90e+01	1.68e+02	2.61e+03	1.95e+02
5	1.00e+01	7.85e+01	5.24e+02	3.08e+03	1.64e+04	2.49e+07	1.54e+22
10	2.00e+01	3.14e+02	4.19e+03	4.93e+04	5.26e+05	2.55e+10	1.73e+37

As shown in table 3.1, the relationship between the radius, the volume and the number of dimensions n, of an n-ball, is complicated. This happens for the reason that if the radius is smaller than 1, the volume decreases, while if is larger than 1, the volume rapidly increases with increasing number of dimensions. In order to have a better view over this relationship, some plots for specific values of R are being shown in figures 3.10 to 3.13.

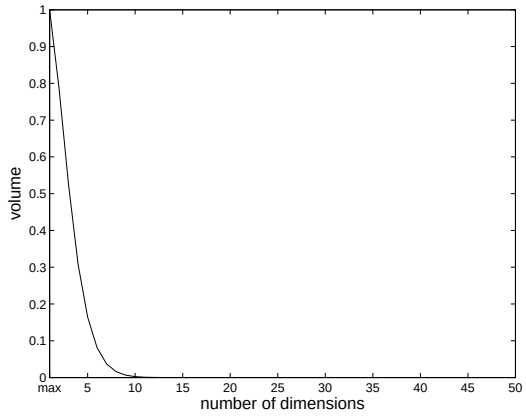


Figure 3.10: R=0.5

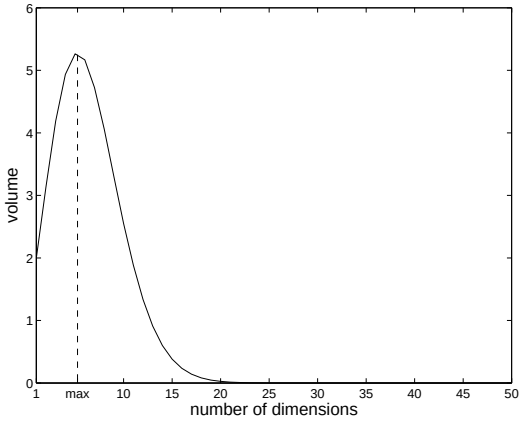


Figure 3.11: R=1

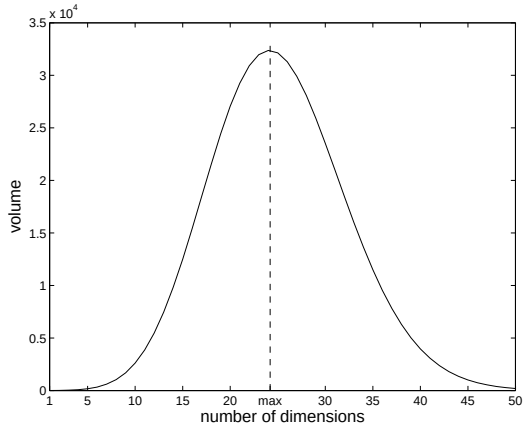


Figure 3.12: R=2

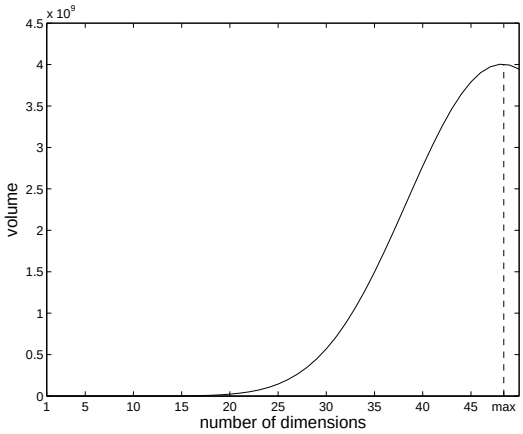


Figure 3.13: R=2.8

It is obvious that the volume of an *n*-sphere is highly dependent to the radius. This dependency causes problems when sampling in an *n*-spherical subspace because if the generated radius are within the interval $(0, 1]$, the volume approaches zero in high dimensions. As a result it is very inconvenient to sample in a zero volume subspace. On the other hand when the radius is high, as for example in the case of the outer sphere, where the radius depends on *n* and increases itself in high dimensions, problems may occur by the fact that the volume of the subspace is very high. Sampling in a very large volume subspace may also yield to inefficiency, because the local search domain is not limited properly.

Finally, the outer *n*-sphere is preferred than the inner one, because the inner sphere's volume is lower for all the numbers of dimensions and approaches zero when *n* is very high. Sampling in a zero-volume subspace would be inconvenient.

3.3.2 Assignment of the hyper-elliptic subspace

For the assignment of the option of sampling within a hyper-elliptic subspace, the new input parameter 'shape_subsp' was assigned first. This parameter should equal to 0 for sampling within a hyper-cubic subspace and to 1 for sampling within a hyper-elliptic subspace. The new parameters is assigned in the algorithm as part of the 'opt_set_i' set of parameters and specifically as

'opt_set_i(13)'. It is then necessary to increase the parameter 'gen_opt_i(1)' from 12 to 13 because this parameter implies the size of the 'opt_set_i(:)' array.

Concerning the computational approach of the hyper-elliptic subspace, the upper and lower bounds $\mathbf{x}^L$ and $\mathbf{x}^U$ where used to calculate the radius vector of the hyper-ellipsoid which contains the hyper-cuboid, that was used as local search domain, and osculates to its vertices. The generation of a random point within the area between the hyper-ellipsoid defined by $\mathbf{x}^U$ and the hyper-ellipsoid defined by $\mathbf{x}^L$, is different from the previous case of the hyper-cuboid, because it is impossible to sample a random mutation of the coordinates of the parent points directly. The algorithm firstly generates n random radius and $n-1$ random angles and then uses the transformation from hyper-ellipsoidal to Cartesian coordinates shown in (3.19). It should be noted, that in the following, the term 'hyper-cube' will be used for simplicity but it actually refers to 'hyper-rectangle' which is the mathematically correct term.

$$\begin{aligned} x_1 &= r_1 \cos(\phi_1) \\ x_2 &= r_2 \sin(\phi_1) \cos(\phi_2) \\ x_3 &= r_3 \sin(\phi_1) \sin(\phi_2) \cos(\phi_3) \\ &\cdot \\ &\cdot \\ &\cdot \\ x_{n-1} &= r_{n-1} \sin(\phi_1) \ldots \sin(\phi_{n-2}) \cos(\phi_{n-1}) \\ x_n &= r_n \sin(\phi_1) \ldots \sin(\phi_{n-2}) \sin(\phi_{n-1}) \end{aligned} \tag{3.19}$$

Where $\phi_1, \phi_2, \ldots, \phi_{n-2} \in [0, \pi]$ and $\phi_{n-1} \in [0, 2\pi)$

The generation of the random radius and the random angles, is described by the equations (3.20), after we have obtained a set of n random numbers u_i from $U[0, 1]$.

$$\begin{aligned} \mathbf{r} &= \mathbf{min\_d} \cdot \frac{\sqrt{2}}{2} + \mathbf{u} \cdot (\mathbf{max\_d} - \mathbf{min\_d}) \cdot \frac{\sqrt{2}}{2} \\ \phi_1 &= u_2 \cdot \pi \\ &\cdot \\ &\cdot \\ &\cdot \\ \phi_{n-2} &= u_{n-1} \cdot \pi \\ \phi_{n-1} &= u_n \cdot \pi \end{aligned} \tag{3.20}$$

It should be noted here, that the difference between the hyper-ellipsoid and the hyper-sphere, is that the hyper-ellipsoid extends differently from dimension to dimension and that is the reason it was used. It assigns n different radius which are required to calibrate the distance of the mutation from the parent point for every dimension separately.

Finally, we obtain the mutation vector **dx** by combining (3.19) and (3.20). The random offspring point is evaluated by adding the mutation vector to the parent point coordinates, using (3.21).

$$\mathbf{x}^{[q+1]} = \mathbf{x}^{[q]} + \mathbf{dx} \tag{3.21}$$

A structogram of the steps that the algorithm follows to calculate the coordinates of he offspring point randomly, within the *idim*-dimensional elliptical subspace, is shown in figure 3.14. The corresponding Fortran source code can be found in appendix B.3.

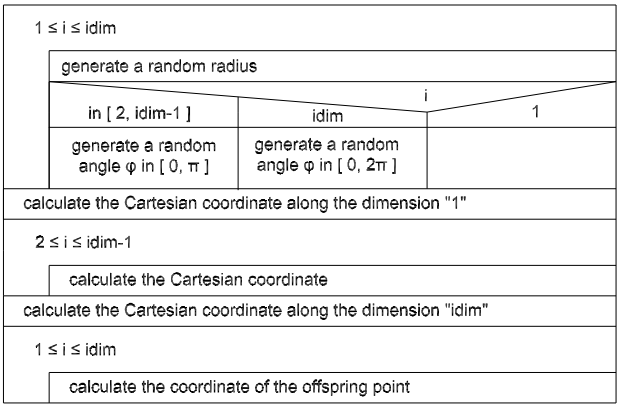


Figure 3.14: Structogram for the calculation of random points within the *idim*-dimensional subspace

In figures 3.15 to 3.18, the distribution of the absolute distances of the sampled points from the centre of the sphere, is shown for different number of dimensions. These distributions where derived after sampling 10000 random points within the domain which is upper and lower bounded by the two *n*-spheres defined by '*r max*' and '*r min*' respectively. The lower bound '*r min*', is derived using the *n*-dimensional vector '**min_d**' and equals to $\sqrt{n}$ in this particular case, while the upper bound '*r max*' by the *n*-dimensional vector '**max_d**' and equals to $4 \cdot \sqrt{n}$ in this case.

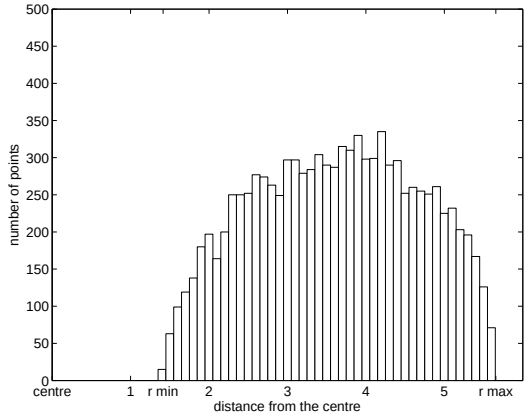


Figure 3.15: 2 dimensions

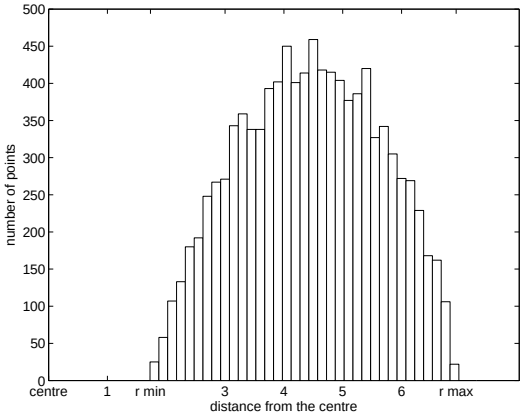


Figure 3.16: 3-dimensions

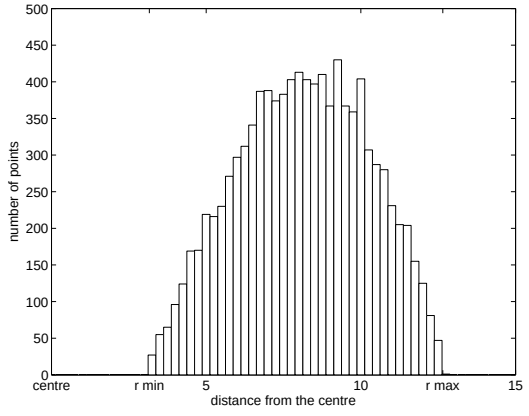


Figure 3.17: 10-dimensions

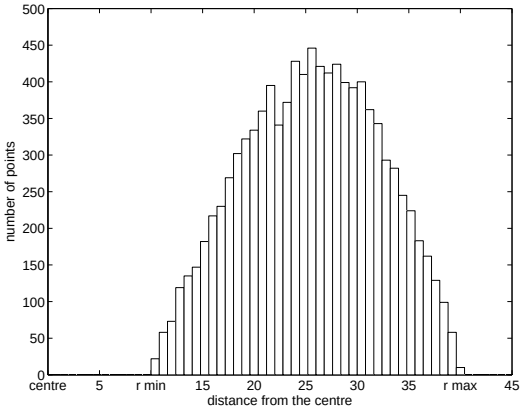


Figure 3.18: 100-dimensions

The distributions appear very similar to each other apart from the 2-dimensional case which is closer to a uniform distribution. In reality, the triangular shape of the distribution, of the distances of the sampled points from the centre, implies a lower possibility of sampling a point close to the bounds than the possibility of sampling it in the space in-between. This is credited to the fact that sampling a point close to the bounds requires a combination of small, or large, random numbers at the same time, which is less likely as the number of dimensions increases. It should be noted, that this distribution does not depend only on the random radius generation, but also in the random angle. As a result, understanding its shape is much more complicated and may include other influencing parameters that become effective in high dimensional spaces. The estimated distribution of the distances of the sampled points from the centre of the permissible domain, is good for the purposes of the present project, but a uniform distribution would be ideal.

Previously in the present subsection, only the case of the randomly selected offspring point was explained. This is because the rest of the optimization procedure was kept exactly the same as the hyper-cubic subspace case. Even the vertices of the hyper-cuboid are being sampled because they belong to the outer surface of the hyper-ellipsoid as well and it is a convenient way of sampling some first points all around the parent point.

In the following subsection, two case studies are being presented in order to compare the efficiency of the two possibilities for the shape of the local search domain.

3.3.3 Results

In the present subsection, two case studies, one 10-dimensional and one 5-dimensional, optimization problems have been accomplished in order to compare the efficiency of the algorithm using hyper-cubic subspace with the hyper-elliptic case. More specifically, the optimization problem, refers to the minimization of specific test functions shown in appendix B.4 and where chosen from [20].

The measure of the efficiency is an error expressing the absolute difference between the calculated result and the real optimum result,

$$error = z_{opt}^{(evolu)} - z_{opt}^{(real)}. \tag{3.22}$$

In the 10-dimensional case study, the relationship between the size of the subspace and the permissible number of generations was studied as well. The efficiency of the algorithm has been tested for different combinations of:

a) the objective function (2, 3, 6)

b) the shape of the local search domain (hyper-cuboid, hyper-ellipse)

c) the size of the local search domain (factor c_1) (0.05, 0.1, 0.2, 0.3, 0.4)

d) the number of generations (10, 20, 30, 40, 50, 60)

The results are shown as surface plots in figures 3.19 to 3.24.

The tables from which figures 3.19 to 3.24 were produced, can be found in appendix B.5, table B.1 to table B.6 respectively.

In general, it can be observed that the quality of the result varies very much with respect to these parameters. For all the cases, the optimal combinations found for the maximum number of generations (60), and for the subspace size of about 0.05 to 0.1 of the total permissible domain. Moreover, the error is much higher at the backside of the plots in figures 3.19 to 3.24 because the number

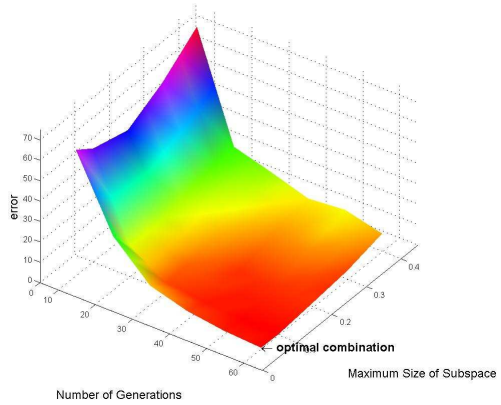


Figure 3.19: Surface plot, 10-dimensional case: axis parallel hyper-ellipsoid function - hyper-elliptical subspace

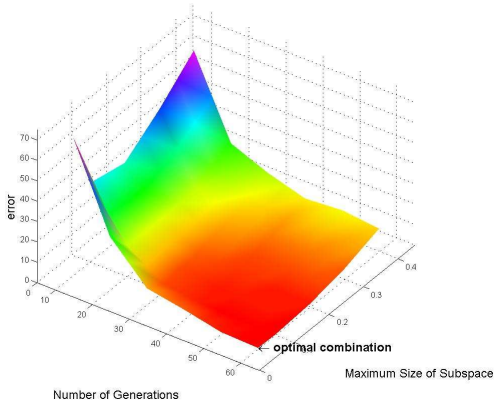


Figure 3.20: Surface plot, 10-dimensional case: axis parallel hyper-ellipsoid function - hyper-cubic subspace

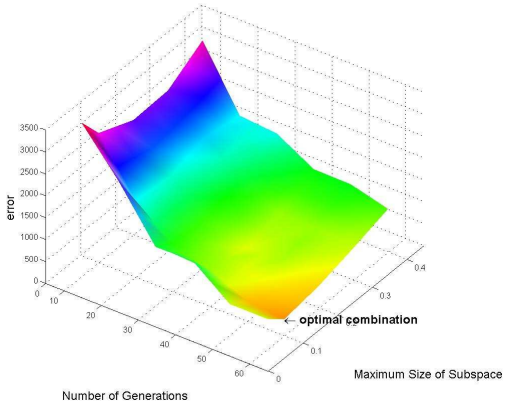


Figure 3.21: Surface plot, 10-dimensional case: rotated hyper-ellipsoid function - hyper-elliptical subspace

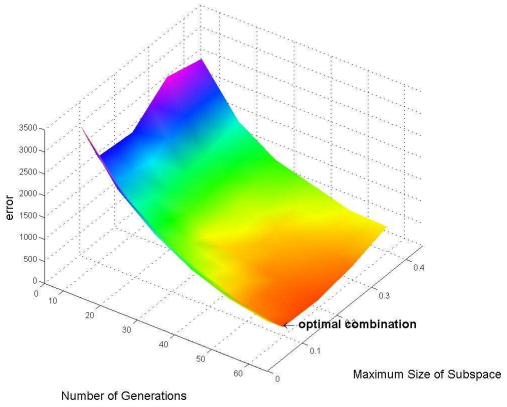


Figure 3.22: Surface plot, 10-dimensional case: rotated hyper-ellipsoid function - hyper-cubic subspace

of steps is not enough to reach an optimum. Indeed, the first results with a noteworthy degree of quality, come for a number of generations of about 30.

Clearly, there is a relationship between the size of the subspace and the number of generations. If the size of the subspace is small, more generations are needed to reach the optimum point (see figures 3.19 to 3.22), hence, the value 0.05 of the factor c_1 is not the best for small number of generations, because the subspace does not sufficiently cover this distance. On the other hand, if the subspace size is too big, the optimum point may lie only a few steps (generations) from the starting point, but the quality of the result decreases because the random sampling of the offspring points will rarely yield to improvements in such a large search domain.

For further investigation of the relationship between these parameters, as well as for more clear comparison between 'ellipsoid' and 'cuboid', a second 5-dimensional case study was accomplished. In this case study, the efficiency of the algorithm has been tested for a fixed number of generations (50) and for different combinations of:

a) the objective function (1, 2, 3, 4, 5, 6, 7, 8, 9)

b) the shape of the local search domain (hyper-cuboid, hyper-ellipse)

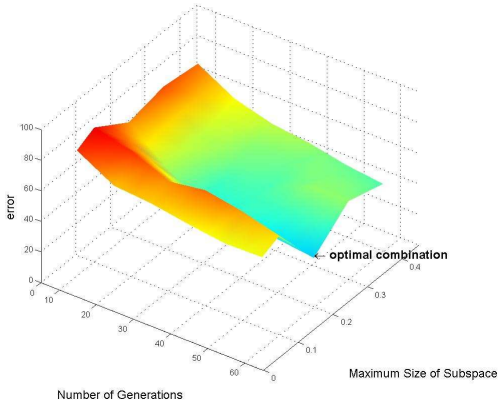


Figure 3.23: Surface plot, 10-dimensional case: Rastring's function - hyper-elliptical subspace

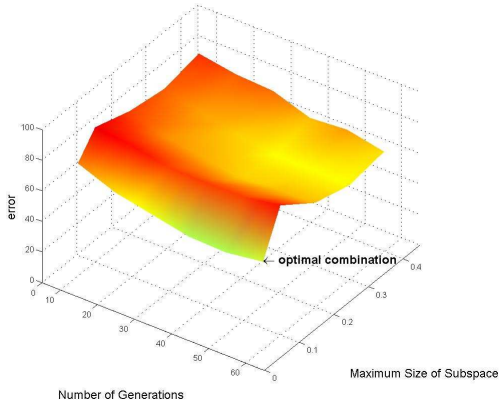


Figure 3.24: Surface plot, 10-dimensional case: Rastring's function - hyper-cubic subspace

c) the size of the local search domain (factor c_1) (0.05, 0.1, 0.2, 0.3, 0.4)

The results are shown as surface plots in figures 3.25 to 3.33.

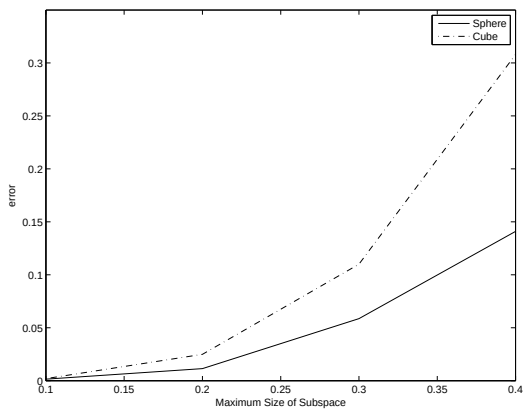


Figure 3.25: 5-dimensional case: De Jong's function, 50 generations

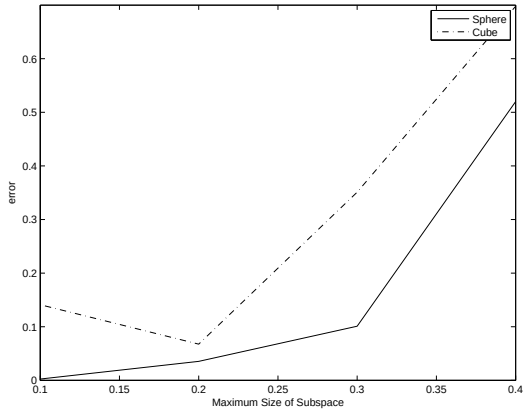


Figure 3.26: 5-dimensional case: axis parallel hyper-ellipsoid function, 50 generations

In figures 3.25, 3.26, 3.27, 3.28, 3.30 and 3.31, the ellipsoid appears to perform better for all the values of the factor c_1 which controls the shape of the local subspace of search. The test functions 1, 2, 3 and 4, are monotonically increasing smooth functions, a characteristic that explains the fact that the shapes of the two lines in the corresponding figures 3.25 to 3.28 are roughly the same. This happens, because the optimum search procedure, is dominated by points on the corners of the hyper-cuboid in monotonically increasing functions. Note that in the hyper-ellipse case, some of the corners of the internal hyper-cuboid are being sampled as well. The fact that the results provide decreased error for the hyper-ellipse case, is ascribed to the increased volume covered by the hyper-ellipse in contrast to the corresponding hyper-cuboid.

The results of the study of the test functions 6 and 7, were the most worthy for the new subspace shape proposal of the ellipsoid due to the fact that it produces better results in all cases. Moreover, these test functions, have a very complicated shape with numerous local optima and the fact that the proposed subspace shape provided an improvement to the efficiency of the method, is very encouraging.

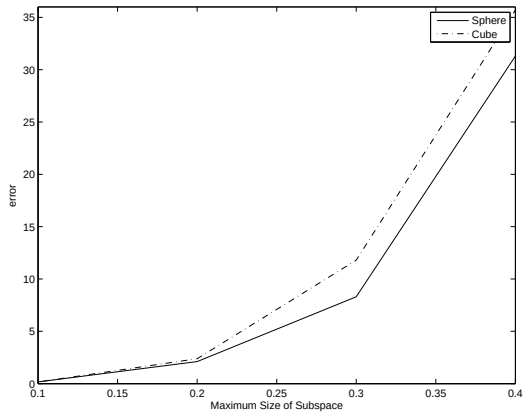


Figure 3.27: 5-dimensional case: rotated hyper-ellipsoid function (with respect to the coordinate axes), 50 generations

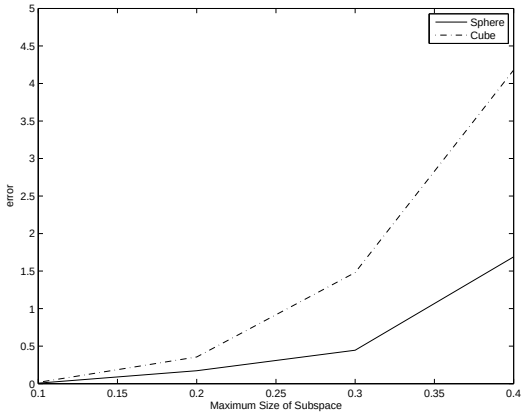


Figure 3.28: 5-dimensional case: moved axis parallel hyper-ellipsoid function, 50 generations

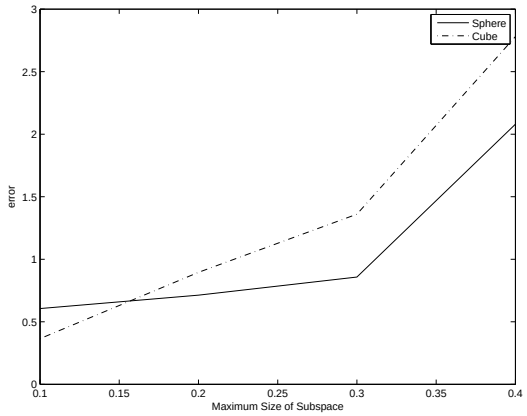


Figure 3.29: 5-dimensional case: Rosenbrock's valley function, 50 generations

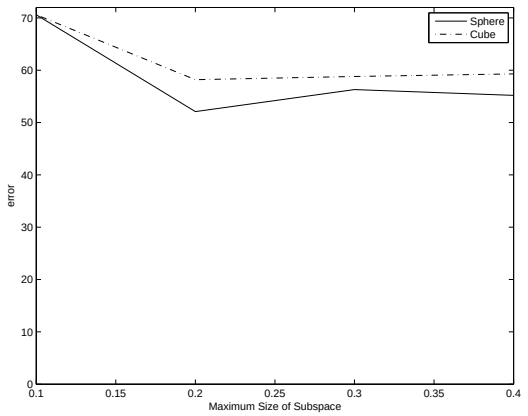


Figure 3.30: 5-dimensional case: Rastring's function, 50 generations

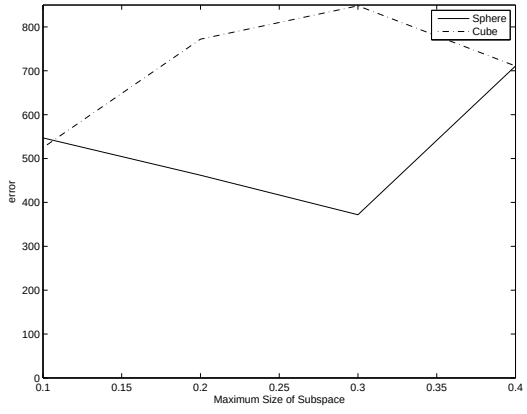


Figure 3.31: 5-dimensional case: Schwefel's function, 50 generations

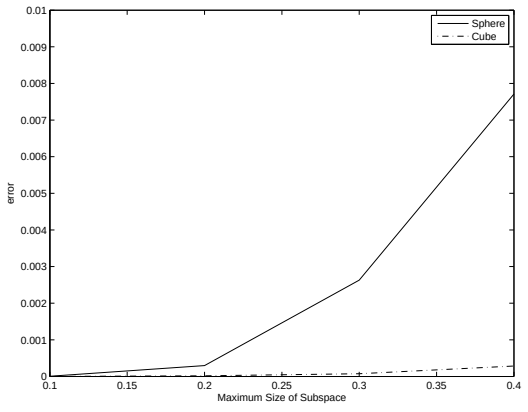


Figure 3.32: 5-dimensional case: sum of different power functions, 50 generations

On the other hand, in figures 3.32 and 3.33, the situation is different. The cuboid appears to perform much better than the ellipsoid, and the results are adjudged as bad. Generally, the test function 8, which corresponds to 3.32, is similar to the test function 6 but much more complicated by means of more local optima and the search of an optimum is difficult. On the other hand, the

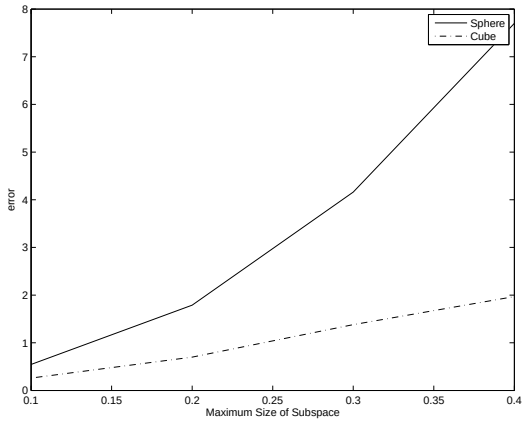


Figure 3.33: 5-dimensional case: Griewangk's function, 50 generations

decreased efficiency of the hyper-ellipse in contrast to the hyper-cube, is ascribed to the fact that the bounds of the permissible search domain are very small. As a result, the increased volume of the hyper-ellipse, assigns an unfavourable effect.

The value of the error in all the cases, strongly depends on the values of the bounds of the permissible search domain, which are very different for different test functions. As a result, very big differences of the error can be observed in figures 3.25 to 3.33 and this is because the error is absolute, not normalized, so it is not comparable from case to case.

Finally, to summarize, global optimization of arbitrary multivariate functions is a difficult task which is essential in almost all the computational fields. As a result, a lot of research dealt with it and it deserves much more, until a sufficiently reliable and robust method is found. The existing methods, rely enormously on the special characteristics of the optimization problem such as the size of the permissible domain and the shape of the objective function and none of them provides global applicability. Further improvement in efficiency of 'evolu', requires much more case studies with different test functions and parameter ranges. With a large number of available results, the relationship between the number of generations and the size of the subspace can be investigated deeply. Conclusively, a proposed combination between the two parameters is then possible to be defined and help the user of the software achieve more reliable results. Based on the results provided in the two case studies of the present chapter, it can be proposed that for a maximum number of generations ≥ 50, a value of the factor c_1 in the interval $[0.05, 0.2]$ can yield optimum results with a breadth range of applicability.

4 CONCLUSIONS

The present project, was engaged with the engineering treatment of uncertainty. Conclusively, the structural uncertainty analysis, is a recently developed field attempting to include subjective influencing parameters into the objective fundamental solution.

In chapter 2, the aim of decreasing the large computational effort of the direct Monte Carlo method was achieved by implementing the subset sampling method. Subset sampling, proved to calculate the failure probability efficiently, even for sample sizes much less than the Monte Carlo required sample size shown in table 2.1. In addition to that, subset sampling method overcomes the usual *burn-in* disadvantage of the MCMC simulation method by choosing the initial samples for each subset, within the previous subset's region. The results are summarized in figures 2.27 and 2.29, where the range of the calculated probability of failure using stochastic analysis, is presented for 10 runs of the algorithms. An important aspect of the comparison between the two methods, is that due to the complexity of the subset sampling algorithm, the computational effort within an iteration step is higher in contrast to the Monte Carlo method. Moreover, as a consequence of the reduced number of computations, an increase in the variance of the failure probability is observed. Even so, the total computational effort is much lower and subset sampling is capable of performing the efficient calculation of small failure probabilities in reliability problems including uncertainty.

The implementation of the subset sampling algorithm into the uncertainty analysis tool 'Winfuz', in addition to the introduced GUI dialog, improved the existing program by assigning this new option for the stochastic analysis. The subset sampling algorithm, may be used through 'Winfuz' for performing direct stochastic analysis or in combination with fuzzy analysis. That is, fuzzy stochastic analysis and even the more advanced method fuzzy stochastic fuzzy analysis, the explanation of which is beyond the scope of the present project. A numerical example of fuzzy stochastic analysis for the calculation of the probability of failure of a truss system was performed in chapter 2. In this example, the two stochastic analysis algorithms (MCS, subset sampling) were used separately, and the results were very close to each other, proving that the implementation of the new algorithm in 'Winfuz', as well as its performance were successful.

Concerning chapter 3 of the present project, the existing subroutine was found to provide sufficient efficiency for most of the test functions that have been optimized. Moreover, with the modification of the local search domain shape, improved efficiency was observed for most of the test functions, and indeed, for the most 'difficult' ones, in the sense of complexity and local optima. In addition to that, an investigation over the relationship between the size of the local search domain and the number of generations, in achieving accurate results with respect to the computational effort, yielded an optimum combination proposal of these parameters. The improved optimization library 'evolu', can be used for the α-level optimization technique, which it may thereafter improve the fuzzy analysis component of 'Winfuz'.

The subjective nature of some influencing parameters of am engineering problem, has always been a difficult problem for engineers when trying to describe natural phenomena within a mathematical framework with respect to realism, accuracy and simplicity. Uncertainty analysis, aims to quantify the difference between reality and engineering models. As a result, the dependency of the existing methods, on the knowledge and experience of the experts, is still high, and thus, the efforts of the researchers, are concentrated in decreasing that dependency. Another important aspect about uncertainty analysis that requires researchers attention, is the influence of the results e.g. probabilities of failure, in making decisions during design or maintenance of engineering systems.

Finally, due to the fact that uncertainty analysis is highly connected with the fields of statistics and

stochastic processes, deep knowledge of these fields is required for dealing with and improving existing methods. In addition to the continuous upgrading of computers together with the development of tools such as Markov Chain Monte Carlo simulation, provide great opportunities for developments in uncertainty analysis.

ACKNOWLEDGEMENTS

I would like to thank Dipl.-Ing. Marco Götz for his interest, help and useful recommendations, Prof. Dr.-Ing. Wolfgang Graf, Prof. Dr.-Ing. habil. Michael Kaliske, as well as my friend Evi for her support.

Appendices

A APPENDIX

A.1 LIMIT THEOREMS

Theorem A.1. ***Weak Law of Large Numbers - Bernouli*** *Let $\boldsymbol{X}_1, ..., \boldsymbol{X}_n$ be a sequence of independent and identically distributed random variables, each having a mean $\langle \boldsymbol{X}_i \rangle = \mu$ an a standard deviation σ. Define another variable*

$$\boldsymbol{X} \equiv \frac{\boldsymbol{X}_1 + ... + \boldsymbol{X}_n}{n}.$$

Then, as $n \to \infty$, the sample mean $\langle \boldsymbol{X} \rangle$ equals the population mean μ of each variable.

$$\boldsymbol{X} = \left\langle \frac{\boldsymbol{X}_1 + ... + \boldsymbol{X}_n}{n} \right\rangle = \frac{1}{n}(\langle \boldsymbol{X}_1 \rangle + ... + \langle \boldsymbol{X}_n \rangle) = \frac{n \cdot \mu}{n} = \mu.$$

In addition,

$$var(\boldsymbol{X}) = var\left(\frac{\boldsymbol{X}_1 + ... + \boldsymbol{X}_n}{n}\right) = var\left(\frac{\boldsymbol{X}_1}{n}\right) + ... + var\left(\frac{\boldsymbol{X}_n}{n}\right) = \frac{\sigma^2}{n^2} + ... + \frac{\sigma^2}{n^2} = \frac{\sigma^2}{n}.$$

Therefore, by the Chebyshev inequality, for all $\epsilon > 0$,

$$P(|\boldsymbol{X} - \mu| \geq \epsilon) \leq \frac{var(\boldsymbol{X})}{\epsilon^2} = \frac{\sigma^2}{n \cdot \epsilon^2}.$$

As $n \to \infty$, it then follows that

$$\lim_{n \to \infty} P(|\boldsymbol{X} - \mu| \geq \epsilon) = 0$$

[19],[8]

Theorem A.2. ***Strong Law of Large Numbers - Kolmogorov*** *Let $\boldsymbol{X}_1, ..., \boldsymbol{X}_n$ be a sequence of independent random variables, with corresponding means $\mu_i = \langle \boldsymbol{X}_i \rangle$ and variances $\sigma_i^2 = var(\boldsymbol{X}_i)$ $(i = 1, ..., n)$. This sequence, obeys the strong law of large numbers if, to every pair $\epsilon, \delta > 0$, corresponds an N such that there is probability $1 - \delta$ or better that for every $r > 0$, all $r+1$ inequalities*

$$\frac{|S_n - m_n|}{n} < \epsilon$$

for $n = N, N + 1, ..., N + r$ will be satisfied. Where

$$S_n \equiv \sum_{i=1}^{n} \boldsymbol{X}_i$$
$$m_n \equiv \langle S_n \rangle = \mu_1 + ... + \mu_n.$$

Kolmogorov established that the validity of the inequality

$$\sum_{i=1}^{\infty} \frac{\sigma_i^2}{n^2} < \infty,$$

is a sufficient condition for the strong law of large numbers to apply to the sequence of mutually independent random variables $\boldsymbol{X}_1, ..., \boldsymbol{X}_n$. [8]

Theorem A.3. ***Central Limit Theorem*** *Let $\boldsymbol{X}_1, ..., \boldsymbol{X}_n$ be a set of N independent random variates and each $\boldsymbol{X}_i$ has an arbitrary probability distribution $P(x_1, ..., x_N)$ with mean μ_i and a finite variance σ_i^2. Then the normal form variate*

$$\boldsymbol{X}_{norm} \equiv \frac{\sum_{i=1}^{N} x_i - \sum_{i=1}^{N} \mu_i}{\sqrt{\sum_{i=1}^{N} \sigma_i^2}}$$

has a limiting cumulative distribution function which approaches a normal distribution.

Under additional conditions on the distribution of the addend[1] *the probability density itself is also normal with mean $\mu = 0$ and variance $\sigma^2 = 1$. If conversion to normal form is not performed, then the variate*

$$\boldsymbol{X} \equiv \frac{1}{N} \sum_{i=1}^{N} x_i$$

is normally distributed with $\mu_X = \mu_x$ and $\sigma_X = \sigma_x/\sqrt{N}$. [8]

A.2 TRACE PLOTS

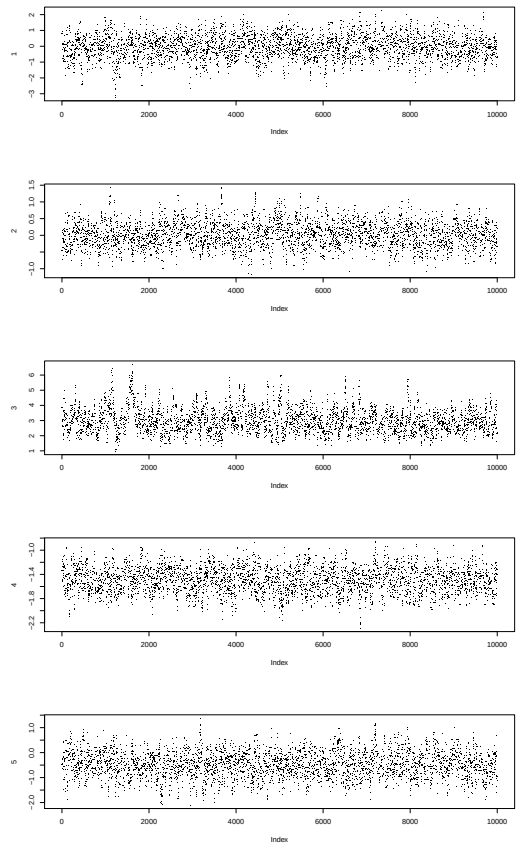


Figure A.1: Trace plots of all input variables for the 1^{st} limit state

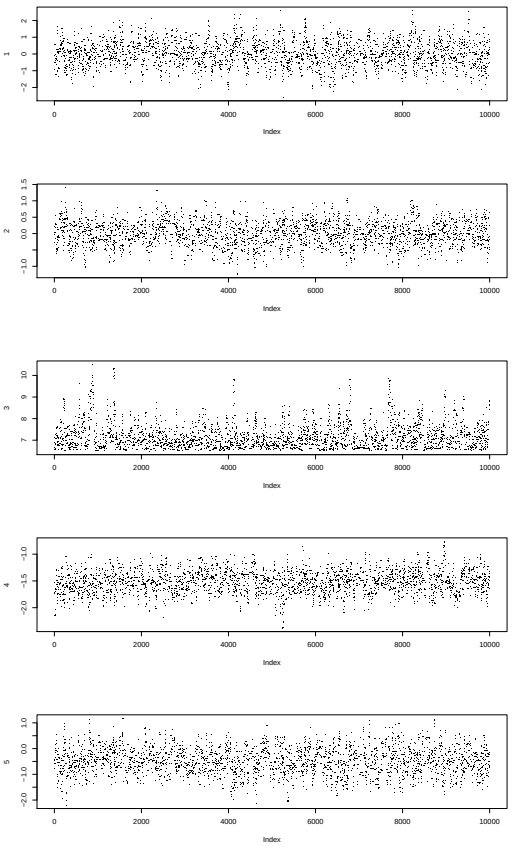


Figure A.2: Trace plots of all input variables for the 3^{rd} limit state

[1] Addend is a quantity to be added to another, also called a summand. For example, in the expression $a + b + c$, a, b and c are all addends. The first of several addends is sometimes called the augend.

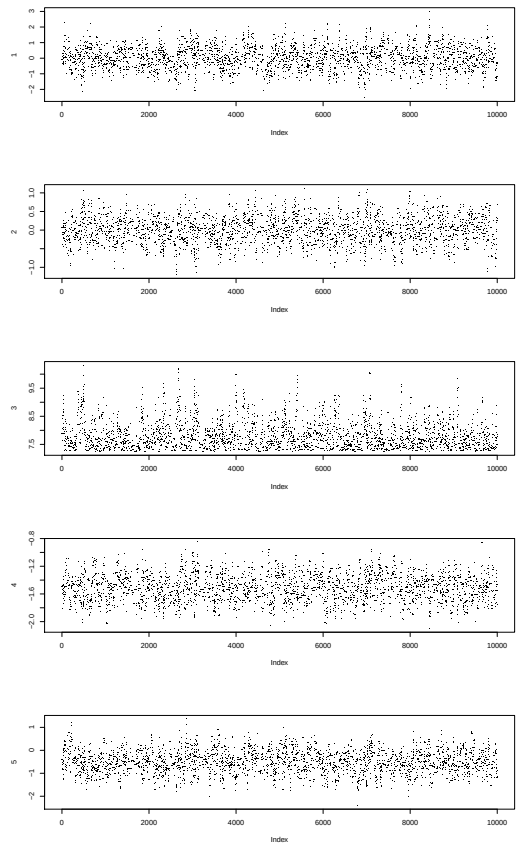


Figure A.3: Trace plots of all input variables for the 4^{th} limit state

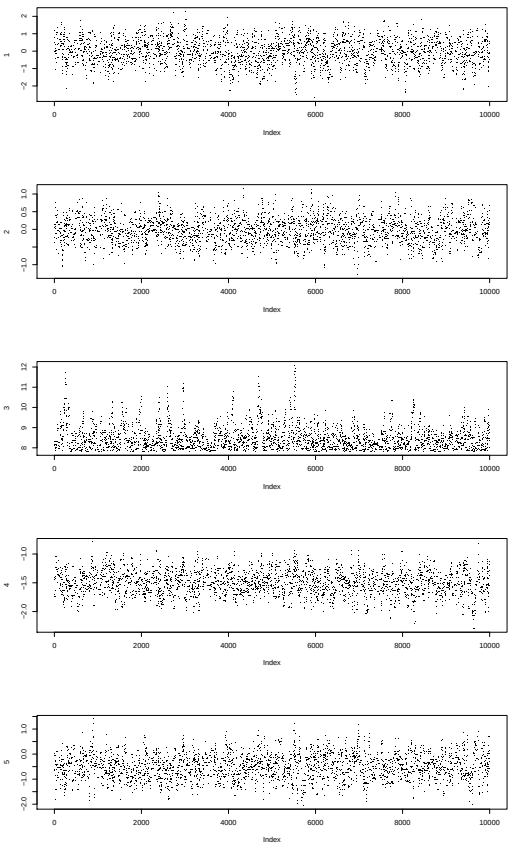


Figure A.4: Trace plots of all input variables for the 5^{th} limit state

A.3 PAIRS PLOTS

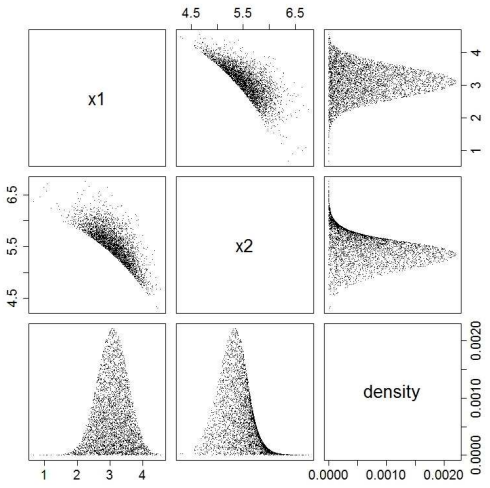


Figure A.5: Pairs plot for the 4th limit state

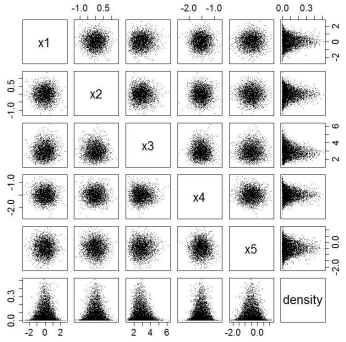


Figure A.6: Pairs plot for the 1st limit state for the 5-dimensional case

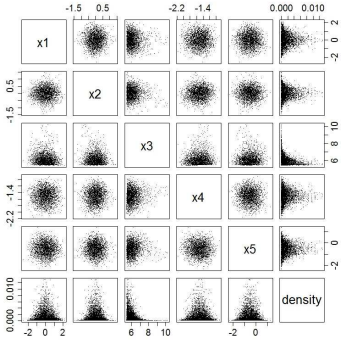


Figure A.7: Pairs plot for the 2nd limit state for the 5-dimensional case

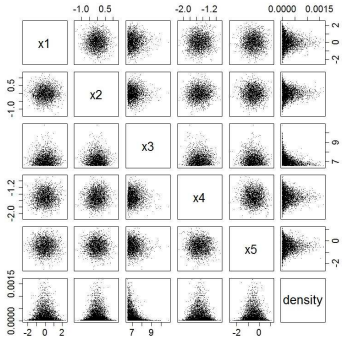


Figure A.8: Pairs plot for the 3rd limit state for the 5-dimensional case

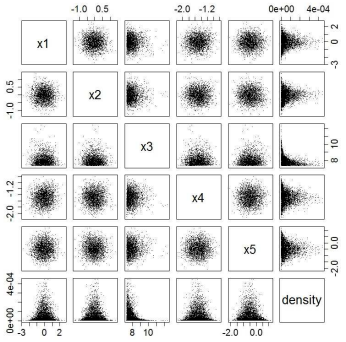


Figure A.9: Pairs plot for the 4th limit state for the 5-dimensional case

A.4 TRACE PLOTS FOR MONTE CARLO SIMULATION

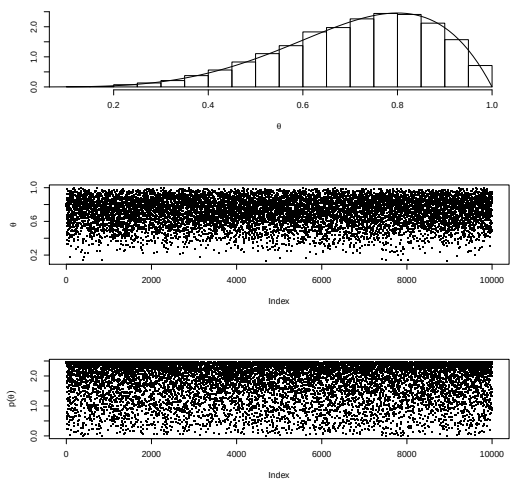


Figure A.10: 10,000 MC samples of the Be(5,2) density. **Top panel**: histogram of samples from the Monte Carlo simulation algorithm and the Be(5,2) density. **Middle panel**: θi plotted against i. **Bottom panel**: $p(\theta i)$ plotted against i

A.5 FORTRAN SOURCE CODE AND VARIABLES

The input variables of 'MCMC' are:

```
anz_ipara    = number of elements of the 'ipara' array, which contains the input
               parameters of integer type.
ipara(anz_ipara) -> 1 - anz_rpara - number of real parameters
                    2 - anz_ZG
                    3 - typ_ZZ  type of random number (equaly destributed,
                        quasi random, IHS-> Improved hyperqube sampling)
                    4 - neue_ZZ - not used in this subroutine
                    5 - typ_Realisierungen - Decomposition type,
                        Cholesky, Eigenvalue etc.
                    6 - anz_ix - size of the 'ix' array
                    7 - anz_rx - size of the 'rx' array
                    8 - anz_sim, total number of runs = mc*(nls+1)
                    9 - number of dimensions = nv
                    10 - zufallszahlen_init - not used in this subroutine
                    11 - Choose dependency between concecutive sample points
                         1/2 Uniform distr./Normal distr
                    12 - Proposal radius default/user specified - 0/else
                    13 - number of subsets = nls
                    14 - number of Markov chains per subset = nmc
rpara(ipara(9))        = proposal radius for every dimension.
                         Allocated only if ipara(12) not equal to 0
ix(ipara(9), 1)        = vtyp - types of distributions of all input variables
ix(ipara(9), 2)        = vstyp - possible choices for different sets of distribution
                         specification parameters, by moments, by parameters
ix(ipara(9), 3)        = anz_para - amount of parameters
rx(ipara(9), ipara(7)) = dp_para - values of distribution specification parameters
limit_value            = final limit value beyond which failure occurs
versag_unter_ueber     = yes / no - underflow / overflow of the
                         limit_value by the objective function
```

The control variables are:

```
Z(nv)             = array in which the samples of the initial MCS are saved
ZS(mc+1,nmc+1,nv) = array in which the samples of the subset sampling simulation
                    are temporarily saved
fe1               = threshold value of the 1st subset, latter equals to LS(1)
step              = distance between adjacent thresholds
LS(nls+1)         = threshold value of the indicated subset
rndMC(mc,nv)      = array of random numbers for the initial MCS
rnd(mc*nls,nv+1)  = array of random numbers for the subset sampling
                    points of the Markov chains of the next subset
X(nv)             = prob. density of the proposal state for he indicated dimension
Y(nv)             = prob. density of the previous state for he indicated dimension
A                 = joint probability density of the proposal state
B                 = joint probability density of the previous state
r                 = acceptance ratio of the M-H algorithm
p                 = random number which is being compared to r
zielfunk          = value of the objective function
x_stoch_det(nv)   = sample the functional value of which is going to be computed
rad(nv)           = proposal radius for the indicated dimension
Psing(1)          = probability of a point to lie in the first subset
Psing(2:nls+1)    = conditional probability of a point to lie in the indicated
                    subset, given that it lies in the previous one
mx                = mean value
sx                = standard deviation
ui(nmc)           = array of random numbers used for random assignment of starting
```

```
temp(mc*(nls+1)+1,nv) = array that temporary saves samples used for the assignment
                        of starting points of the Markov chains of the next subset
SPi(nls+1)        = number of samples that lie in both the current and the next subset
info              - not used in this subroutine
error             - not used in this subroutine

---- Parameters for connection with other parts of the program -----------------------
akt_MCS_run       = counter of simulation runs
akt_ERG
akt_ZP
akt_Start_SG
akt_ERG_SG
ZP
Ergebnis
nrERG
```

The output variables are:

```
ZZ                = samples of the 'MCS' subroutine, not used in this subroutine
Pf                = final probability of failure
```

```
subroutine MCMC(anz_ipara, ipara, rpara, ix, rx, ZZ)
! MCMC is a subroutine which performs the subset sampling method
! using the Markov chain Monte Carlo simulation method.
use probf90_r8
use analysedaten, only : akt_MCS_run,akt_ERG,akt_ZP,akt_Start_SG
use module_randomnumber

use datamanagement
use fuzzyeingabe, only : ZP
use fuzzyergebnis, only : Ergebnis

implicit none
    integer, intent(in) :: anz_ipara
    integer, intent(in) :: ipara(anz_ipara)
    real, intent(in)    :: rpara(ipara(1))
    integer, intent(in) :: ix(ipara(9), ipara(6))
    real, intent(in)    :: rx(ipara(9), ipara(7))
    double precision,intent(inout) :: ZZ(ipara(8),ipara(2)) ! not used in this subrtn
!
    integer                     :: i, j, anz_para, vtyp, vstyp
    double precision    :: dp_para(ipara(7))
    real*4              :: x_stoch_det(ipara(9))
    character(LEN=1024) :: info
    logical             :: error

    double precision, allocatable :: ZS(:,:,:), rndMC(:,:), rnd(:,:)
    double precision    :: Z(ipara(9)),ui(ipara(14)), A, B, Psing(ipara(13)+1), Pf
    double precision    :: Y(ipara(9)), X(ipara(9))
    double precision    :: mx, sx, temp(ipara(8)+1,ipara(9))
    real*4              :: limit_value, fe1, step, LS(ipara(13)+1)
    real*4              :: r, p, zielfunk, rad(ipara(9))
    integer             :: k, m, nv, mc, nls, nmc, ind, SPi(ipara(13)+1)
    integer(kind = 4)   :: nrERG, akt_ERG_SG
    logical             :: versag_unter_ueber

!-------------------------------------------------------------------------------
! getting data out of datapool
!-------------------------------------------------------------------------------
    if (Ergebnis(akt_ERG)%anz_SG .ne. 1) then
```

```
        ! error
        continue
    endif
    !
    nrERG = search_nr('ERG', Ergebnis(akt_ERG)%SG(akt_Start_SG)%link_ERG, link_ZP &
                      + = ZP(akt_ZP)%nr)
    akt_ERG_SG = search_nr('ERG_SG', ZP(akt_ZP)%ERG(nrERG)%link)
    !
    ! true for failure if underflow the limit value
    versag_unter_ueber = ERG_SG(akt_ERG_SG)%Wert(1)%PfDaten%versag_unter_ueber
    !
    ! the final limit value, if exceeded by the objective function, failure occurs
    limit_value        = Ergebnis(akt_ERG)%SG(akt_Start_SG)%Wert(1)%grenz_Wert
!-------------------------------------------------------------------------------
!                                                                  SIMULATIONSLAEUFE
!-------------------------------------------------------------------------------

    !a) Initialization
    mc  = int(ipara(8)/(ipara(13)+1)) - 1  ! total number of samples per subset
    nv  = ipara(9)                         ! number of dimensions

    nls = ipara(13)                        ! number of Subsets
    nmc = ipara(14)                        ! number of Markov Chains

    temp = 0.0d0

    ! alloacate needed arrays
    if (allocated(ZS)) deallocate(ZS)
    allocate(ZS(mc+1,ipara(14)+1,ipara(9)))

    !a)1) Specification of proposal radius
    if (ipara(12).EQ.1) then
        do j = 1, nv
            ! computing standard deviation for given parameters
            vtyp                   = ix(j,1)
            vstyp                  = ix(j,2)
            anz_para               = ix(j,3)
            dp_para(1:anz_para)    = dble(rx(j,1:anz_para))
            !
            call verteilungsparameter (vtyp, vstyp, anz_para, &
                                       + dp_para(1:anz_para), mx, sx)
            rad(j) =  sx  !standard deviations as proposal radius
        enddo
    elseif (ipara(12).EQ.0) then
        do j = 1, nv
            rad(j) =  rpara(j) !user defined proposal radius
        enddo
    endif

    akt_MCS_run = 1

    !a)2) Specification of the firts threshold level value
    do j = 1, nv
            ! computing the mean of every variable for given parameters
            vtyp                   = ix(j,1)
            vstyp                  = ix(j,2)
            anz_para               = ix(j,3)
            dp_para(1:anz_para)    = dble(rx(j,1:anz_para))
            !
            call verteilungsparameter (vtyp, vstyp, anz_para, &
                                       + dp_para(1:anz_para), mx, sx)
            x_stoch_det(j) = mx
```

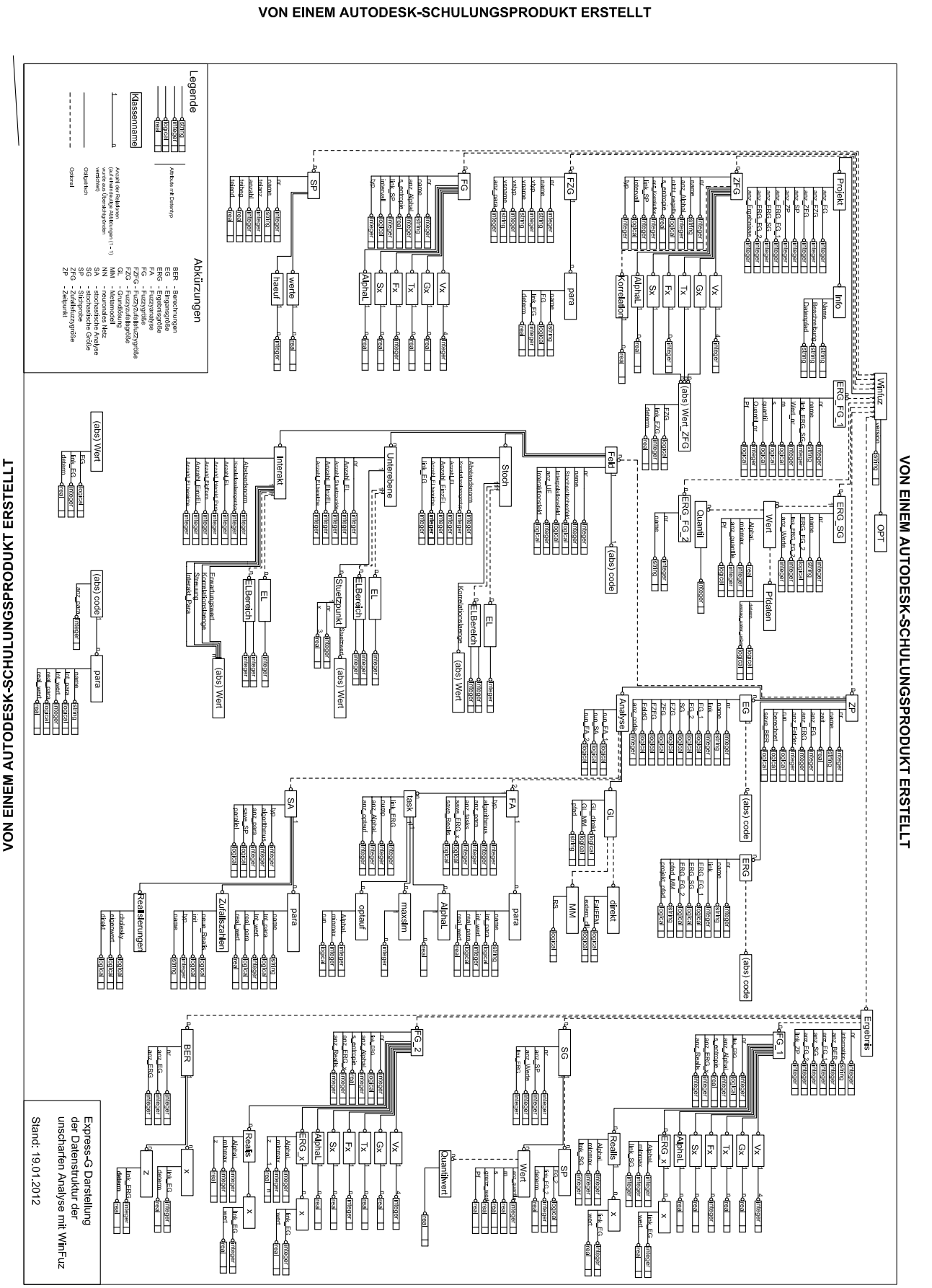
```
enddo
call stoch_det(x_stoch_det, nv)
! the value of the objective function for rx(:,1) (=the means)
! is set as the first failure event
fe1 =  Ergebnis(akt_ERG)%SG(akt_Start_SG)%SP(akt_MCS_run)%determ
akt_MCS_run = akt_MCS_run + 1 ! counter of simulation runs
step = (abs(fe1-limit_value))/nls

select case (versag_unter_ueber)
    case(.true.)          !underflow
        LS(1) = limit_value + nls*step
    case(.false.)         !overflow
        LS(1) = limit_value - nls*step
end select

! b) Direct MCS for determination of P(1)
if (allocated(rndMC)) deallocate(rndMC)
allocate(rndMC(mc,nv))
SPi(:) = 0
call randomnumbers(1, 1, mc, nv, rndMC)
do i = 1, mc
    do j = 1, nv
        call transform(rndMC(i,j), ix(j,1), ix(j,2), ix(j,3), &
                       + dble(rx(j,1:anz_para)), 1, Z(j), error, info)
        x_stoch_det(j) = Z(j)
    enddo
    call stoch_det(x_stoch_det, nv)
    zielfunk =  Ergebnis(akt_ERG)%SG(akt_Start_SG)%SP(akt_MCS_run)%determ
    akt_MCS_run = akt_MCS_run + 1 ! counter of current run

    select case (versag_unter_ueber)
        case(.true.)          !underflow
            if (zielfunk.LE.(LS(1))) then
                SPi(1) = SPi(1) + 1
                temp(SPi(1),:) = Z(:)
            endif
        case(.false.)         !overflow
            if (zielfunk.GT.(LS(1))) then
                SPi(1) = SPi(1) + 1
                temp(SPi(1),:) = Z(:)
            endif
    end select

enddo

Psing(1) = dble(SPi(1))/dble(mc)     ! Probability of being in the first subset

! Here randomly selected 'nmc' samples that lie in the first subset,
! are being set as the starting points ( ZS(1,m,:) ) of the 'nmc' Markov chains
! of the next subset .
call randomnumbers(1, 1, nmc, nv, ui)
do m = 1, nmc
    ind = int(real(ui(m))*real(SPi(1)))
    if (real(ui(m)).gt.0.99) ind = ind-1
    ZS(1,m,:) = temp(ind+1,:)
enddo

! calculation of the next threshold level values, defining the next subsets
select case (versag_unter_ueber)
    case(.true.)         !underflow
        do k = 2, nls+1
            LS(k) = LS(1) - (k-1)*step
```

```
        enddo
    case(.false.)        !overflow
        do k = 2, nls+1
            LS(k) = LS(1) + (k-1)*step
        enddo
end select

if (allocated(rndMC)) deallocate(rndMC)

!c) Calculation of conditional probabilities for indermidiate failure events
if (allocated(rnd)) deallocate(rnd)
allocate(rnd(nls*mc,nv+1))

call randomnumbers(1, 1, nls*mc, nv+1, rnd)

do k = 2, nls + 1
    do i = 2, int(mc/nmc)
        do m = 1, nmc
            ! step 1. of M-H algorithm
            call proposal(rnd((k-1)*(i-1)*m,1:nv), anz_ipara, ipara, rad, &
                          + ZS(i-1,m,:), ZS(i,m,:))
            A=1.0d0
            B=1.0d0
            do j = 1, nv
                ! calculation of the density values of
                ! ZS(i,m,j) -> X(j) and ZS(i-1,m,j) -> Y(j)
                !
                call transform (ZS(i,m,j), ix(j,1), ix(j,2), ix(j,3), &
                                + dble(rx(j,1:ix(j,3))), 3,  X(j), error, info)
                ! X(j) is the prob. density value of ZZ(i) for dimension j
                !
                call transform (ZS(i-1,m,j), ix(j,1), ix(j,2), ix(j,3), &
                                + dble(rx(j,1:ix(j,3))), 3, Y(j), error, info)
                ! Y(j) is the prob. density value of ZZ(i-1) for dimension j
                !
                A=A*X(j) ! A is the joint probability density
                         ! p(X(1),X(2),X(3),...) of ZS(i,m,:)
                B=B*Y(j) ! B is the joint probability density
                         ! p(Y(1),Y(2),Y(3),...) of ZS(i-1,m,:)
            enddo
            r = min(1.0, real(A/B))           ! acceptance ratio
            p = real(rnd((k-1)*(i-1)*m,nv+1)) ! random number to be compared with r
            if (p.GT.r) then     ! this is a 1-1 binomial experiment,
                                 ! it gives 1 with probability r, else gives 0
                ZS(i,m,:) = ZS(i-1,m,:)
                ! if it comes 1, the proposal is being accepted,
                ! so ZS(i,m,:) remains, else ZS(i,m,:) = ZS(i-1,m,:)
            endif

            do j = 1, nv
                x_stoch_det(j) = real(ZS(i,m,j))
            enddo
            call stoch_det(x_stoch_det, nv)
            zielfunk =  Ergebnis(akt_ERG)%SG(akt_Start_SG) &
                        + %SP(akt_MCS_run)%determ
            akt_MCS_run = akt_MCS_run + 1    ! counter of simulation runs

            ! step 2. of M-H algorithm
            select case (versag_unter_ueber)
                case(.true.)              !underflow
                    if (zielfunk.GT.LS(k-1)) then
                        ZS(i,m,:) = ZS(i-1,m,:)
```

```
                        else
                            if (zielfunk.LE.LS(k)) then

                               SPi(k) = SPi(k) + 1
                               temp(SPi(k),:) = ZS(i,m,:)

                            endif
                        endif
                    case(.false.)             !overflow
                        if (zielfunk.LE.LS(k-1)) then
                            ZS(i,m,:) = ZS(i-1,m,:)
                        else
                            if (zielfunk.GT.LS(k)) then

                               SPi(k) = SPi(k) + 1
                               temp(SPi(k),:) = ZS(i,m,:)
                            endif
                        endif
                end select

            enddo
        enddo

        Psing(k) = dble(SPi(k))/dble(mc)  ! Conditional probability of a point
                                          ! to lie on the k-th subset,
                                          ! given that it lies on the (k-1)-th subset.

        ! Here randomly selected 'nmc' samples that lie in the k-th subset,
        ! are being set as the starting points ( ZS(1,m,:) ) of the 'nmc' Markov chains
        ! of the next ((k+1)-th) subset.
        call randomnumbers(1, 1, nmc, nv, ui)
        do m = 1, nmc
            ind = int(real(ui(m))*real(SPi(k)))
            if (real(ui(m)).gt.0.99) ind = ind-1
            ZS(1,m,:) = temp(ind+1,:)
        enddo
    enddo
    Pf = 1.0d0
    do k = 1, nls + 1
        Pf = Pf*Psing(k)
    enddo

    Ergebnis(akt_ERG)%SG(akt_Start_SG)%Wert(1)%m = 0.0
    Ergebnis(akt_ERG)%SG(akt_Start_SG)%Wert(1)%s = 0.0
    Ergebnis(akt_ERG)%SG(akt_Start_SG)%Wert(1)%Pf = Pf  ! Output of Pf

    if (allocated(ZS)) deallocate(ZS)
    if (allocated(rnd)) deallocate(rnd)
!-------------------------------------------------------------------------------
!                                                          ENDE SIMULATIONSLAEUFE
!-------------------------------------------------------------------------------
    endsubroutine MCMC

! ==============================================================================

    subroutine proposal(ra, anz_ipara, ipara, rad, prev, prop)
     ! This subroutine is used to produce a proposal sample point (prop),
     ! from the previous one (prev).
    use probf90_r8
    use module_randomnumber

    implicit none
```

```
integer, intent(in) :: anz_ipara
integer, intent(in) :: ipara(anz_ipara)
real, intent(in)    :: rad(ipara(9))
double precision    :: c(2), ra(ipara(9))
double precision    :: prev(ipara(9)), prop(ipara(9)), par(2)
real*4              :: p
integer(kind = 4)   :: j
character(LEN=1024) :: info
logical             :: error

do j = 1, ipara(9)
    if (ipara(11).EQ.1) then
        prop(j) = prev(j) + (ra(j)-0.5)*2.0*dble(rad(j))
    elseif (ipara(11).EQ.2) then
        ! We obtain a random value (prop(j)) from the normal distribution
        ! with m=prev(j) and sigma=rad(j) using the random number ra(j).
        par(1) = prev(j)
        par(2) = dble(rad(j))
        call transform(ra(j), 2, 1, 2, par, 1, prop(j), error, info)
    endif
enddo

endsubroutine proposal
```

A.6 EXPRESS-G REPRESENTATION OF THE DATA STRUCTURE OF 'WINFUZ'



B APPENDIX

B.1 VARIABLES OF 'EVOLU'

The input and output variables of 'evolu' are:

```
integer (in)  q_gen_opt_i              = quantity of general parameters
integer (in)  gen_opt_i(q_gen_opt_i)   = general parameters
integer (in)  opt_set_i(gen_opt_i(1))  = settings for optimization (integer)
real*4  (in)  opt_set_r(gen_opt_i(2))  = settings for optimization (real)
-------------------------------------------------------------------------------
gen_opt_i( 1)  = 13      quantity of integer parameter

opt_set_i( 1) = idim    dimension of input quantity x
opt_set_i( 2) = odim    dimension of output quantity z
opt_set_i( 3) = dec     1 / 2  == min / max
opt_set_i( 4) = nchain  number of parallel optimzation starts - mulitchain optimization
opt_set_i( 5) = sapo    number of SAmple POints per chain
opt_set_i( 6) = maxsim  maximal number of simulation runs
                        (maxsim*nchain*sapo = all simulation)
opt_set_i( 7) = cso(1)  Chosse Start Option - 1/2/3 = 1-LHS 2-MCS 3-predefined points
opt_set_i( 8) = cso(2)  Chosse Start Option - 1/2   = 1-choose nchain best startpoints
                                                      2-1st point of every chain
opt_set_i( 9) = npred   number of predefined points
opt_set_i(10) = nstart  number of user-specified points, which have to be evaluated
opt_set_i(11) = constr  constraint no/yes - 0 = no, >0 = yes
opt_set_i(12) = z_opt   number of result that should be optimized (zopt in flow chart)
opt_set_i(13) = shape_subsp  NEW PARAMETER subspace cubic/elliptic => 0/1
-------------------------------------------------------------------------------
gen_opt_i( 2) = 5 quantity of real parameter

opt_set_r( 1) = sisusp(1)  maximal SIze of SUbSPace (0...1), factor $c_1$
opt_set_r( 2) = sisusp(2)  minimal SIze of SUbSPace (0...1), factor $c_2$
opt_set_r( 3) = equalsusp  distance, in which old points are re-used (0...1)
opt_set_r( 4) = err_def    abortion criterion for optimization
opt_set_r( 5) = converge   convergence factor [0..1] - 1.0 means fast convergence
                                                       0.0 means slow confergence
-------------------------------------------------------------------------------
gen_opt_i( 3) =            save documentation-files(0/1)
-------------------------------------------------------------------------------
real*4 (in ) x(idim,2)           = lower and upper bounds of search area per dimension
real*4 (in ) xpred(npred,idim)   = predefined x-points (by user) - for starting points
real*4 (in ) zpred(npred,odim)   = predefined z-points (by user) - for starting points
real*4 (in ) xstart(nstart,idim) = user-specified points, which have to be evaluated,
                                   xpred has a functional value while xstart not
-------------------------------------------------------------------------------
integer (out) sim                               = number of (real) simulation runs

real*4  (out) xerg(nchain,idim)                 = resulting optimal x-vector per chain
real*4  (out) zerg(nchain,odim)                 = resulting optimal z-vector per chain
real*4  (out) xsave(maxsim*nchain*sapo,idim)  = all determined x-points in optimization
```

```
real*4  (out) zsave(maxsim*nchain*sapo,odim)  = all determined z-points in optimization
real*4  (out) error(nchain+1)   = error, abort the optimization - compared to "err_def"
```

The control variables are:

```
integer  ncpu                = nchain*sapo
integer  choice(5)           = controls the point generation type (random,guided..)
integer  num_ihs             = ncpu - nstart if ncpu >= nstart, else 0
integer  x_ihs(idim,num_ihs) = input for LHS
integer  lauf                = counter used when saving results to file
integer  num_z_err           = number of calculated errors per chain, to be saved
integer  lauf_opt            = counter used in different parts of 'evolu'

real*4   xrun(nchain,sapo+1,idim) = temporary save x values in every step
real*4   zrun(nchain,sapo+1,odim) = temporary save z values in every step
real*4   x_lhs(num_ihs,num_ihs)   = output samples from LHS
real*4   delta_x(nchain,idim)     = gradient vector to guide point generation
real*4   grad_vor(idim)      = factor used for point generation
real*4   fac                 = factor to size delta_x and avoid very short values
real*4   fac_h    = it is calculated in every step and the smallest is saved as fac
real*4   xh(idim) = temporary save x values in different parts of 'evolu'
real*4   zh(odim) = temporary save z values in different parts of 'evolu'
real*4   error_min(nchain) = to be compared with error
real*4   z_compare         = used to save best point for comparison with the next ones
real*4   z_err(nchain,num_z_error,odim) = $z$ values from which it calculates the error
```

B.2 SOURCE CODE FOR THE BEHAVIOUR AT THE BOUNDARIES

The behaviour of 'evolu' along the boundary of the permissible domain in implemented by the following Fortran code snippet.

```
!   Check for compliance with the bounds of the permissible search domain
!   and placement of the point on the bounds x(i,1) and x(i,2) if required
        do i=1, n
            if (xrun(k,j+1,i).lt.x(i,1))  xrun(k,j+1,i)=x(i,1)
            if (xrun(k,j+1,i).gt.x(i,1))  xrun(k,j+1,i)=x(i,2)
        enddo
!
!  Calculation of the minimum relative distance
        drelmin=0.0
        drelmax=0.0
        do i=1, n
            d(i)=abs(xrun(k,j+1,i)-xrun(k,1,i))
            if ((d(i)/min_d(i)).gt.drelmin)  drelmin=d(i)/min_d(i)
            if ((d(i)/max_d(i)).gt.drelmax)  drelmax=d(i)/max_d(i)
        enddo
!
!  Calculation of the factor c3
        c3=1/2*(1/drelmin+1/drelmax)
!
!  Check for compliance with min_d(i), if drelmin is greater or equal than 1.0,
!  there is no coordinate that lies within the non-permissible minimum domain
        if (drelmin.lt.1.0)
            do i=1, n
                !
                !  Check which coordinates fall short of min_d(i)
                !  and correct them using the factor c3
```

```
                if (d(i)/min_d(i).lt.1.0)
                    xrun(k,j+1,i)=x(k,1,i)+c3*(xrun(k,j+1,i)-xrun(k,1,i))
                endif
            enddo
        endif
```

B.3 SOURCE CODE FOR THE CALCULATION OF POINTS WITHIN THE HYPER-ELLIPSE

```
!   Generation of idim radius and (idim-1) angles phi
        do i=1, n
            call random_number(rand)
            radius(i)=min_d(i)*sqrt(real(n))/2.0 + rand*(max_d(i)-min_d(i)) &
                                                 + *sqrt(real(n))/2.0
            if ((i.ne.1).and.(i.ne.n)) then
                phi(i-1)=rand*pi
            elseif (i.eq.n) then
                phi(i-1)=rand*2.0*pi
            endif
         enddo
!
!  Calculation of the hyper-elliptical coordinates dx(i)
         dx(1)=radius(1)*cos(phi(1))
         do i=2, n-1
             dx(i)=dx(i-1)/( radius(i-1)*cos(phi(i-1)) ) &
             * ( radius(i)*sin(phi(i-1))*cos(phi(i)) )
         enddo
         dx(n)=dx(n-1)/( radius(n-1)*cos(phi(n-1)) ) * ( radius(n)*sin(phi(n-1)) )
!
!   Addition of the hyper-elliptical coordinates dx(i) to the
!   parent point coordinates for generation of the offspring point
         do i=1,n
             xrun(k,j+1,i)=xrun(k,1,i) + dx(i)
         enddo
```

B.4 TEST FUNCTIONS

The test functions used for the case studies of the present project, can be found in [20] and they are:

1. De Jong's function

$$f(\mathbf{x}) = \sum_{i=1}^{n} {x_i}^2 \quad for \quad -5.12 \leq x_i \leq 5.12 \quad global\ min : z_{opt} = 0.0 \tag{B.1}$$

2. axis parallel hyper-ellipsoid function

$$f(\mathbf{x}) = \sum_{i=1}^{n} i \cdot {x_i}^2 \quad for \quad -5.12 \leq x_i \leq 5.12 \quad global\ min : z_{opt} = 0.0 \tag{B.2}$$

3. rotated hyper-ellipsoid function

$$f(\mathbf{x}) = \sum_{i=1}^{n} (\sum_{j=1}^{i} x_j)^2 \quad for \quad -65.536 \leq x_i \leq 65.536 \quad global\ min : z_{opt} = 0.0 \tag{B.3}$$

4. moved axis parallel hyper-ellipsoid function

$$f(\mathbf{x}) = \sum_{i=1}^{n} 5i \cdot {x_i}^2 \quad for \quad -5.12 \leq x_i \leq 5.12 \quad global\ min : z_{opt} = 0.0 \tag{B.4}$$

5. Rosenbrock's function

$$f(\mathbf{x}) = \sum_{i=1}^{n-1} 100 \cdot (x_{i+1} - {x_i}^2)^2 + (1 - x_i)^2 \quad for \quad -2.048 \leq x_i \leq 2.048 \quad global\ min : z_{opt} = 0.0 \tag{B.5}$$

6. Rastring's function

$$f(\mathbf{x}) = 10 \cdot n + \sum_{i=1}^{n} ({x_i}^2 - 10 \cdot \cos(2 \cdot \pi \cdot x_i)) \quad for \quad -5.12 \leq x_i \leq 5.12 \quad global\ min : z_{opt} = 0.0 \tag{B.6}$$

7. Schwefel's function

$$f(\mathbf{x}) = \sum_{i=1}^{n} -x_i \cdot \sin(\sqrt{|x_i|}) \quad for \quad -500 \leq x_i \leq 500 \quad global\ min : z_{opt} = -n \cdot 418.9829 \tag{B.7}$$

8. sum of different power functions

$$f(\mathbf{x}) = \sum_{i=1}^{n} |x_i|^{(i+1)} \quad for \quad -1 \leq x_i \leq 1 \quad global\ min : z_{opt} = 0.0 \tag{B.8}$$

9. Griewangk's function

$$f(\mathbf{x}) = \sum_{i=1}^{n} \frac{{x_i}^2}{4000} - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) \quad for \quad -600 \leq x_i \leq 600 \quad global\ min : z_{opt} = 0.0 \tag{B.9}$$

B.5 TABLES

Table B.1: corresponds to the figure 3.19

	10	15	20	25	30	35	40	45	50	55	60
0.05	61.92	44.45	26.97	18.34	9.72	7.10	4.48	3.18	1.88	1.30	0.71
0.10	55.78	35.41	15.04	9.59	4.14	3.06	1.99	1.45	0.91	0.70	0.48
0.15	53.43	34.32	15.21	10.41	5.60	4.17	2.73	2.31	1.90	1.66	1.43
0.20	51.09	33.23	15.38	11.22	7.07	5.27	3.47	3.18	2.88	2.63	2.37
0.25	55.89	36.67	17.45	13.79	10.14	7.82	5.50	5.41	5.31	4.37	3.43
0.30	60.69	40.10	19.52	16.36	13.21	10.37	7.53	7.63	7.74	6.11	4.48
0.35	67.62	44.39	21.16	18.29	15.41	12.54	9.67	10.13	10.58	8.64	6.70
0.40	74.55	48.68	22.80	20.21	17.61	14.71	11.82	12.62	13.42	11.17	8.92

Table B.2: corresponds to the figure 3.20

	10	15	20	25	30	35	40	45	50	55	60
0.05	68.48	47.53	26.58	17.46	8.33	6.72	5.10	3.18	1.25	0.93	0.61
0.10	38.89	27.10	15.31	9.44	3.56	2.50	1.44	1.06	0.68	0.53	0.37
0.15	36.96	26.09	15.22	9.91	4.60	3.35	2.10	1.78	1.45	1.26	1.06
0.20	35.04	25.08	15.12	10.38	5.63	4.20	2.77	2.49	2.21	1.98	1.75
0.25	41.35	28.42	15.50	11.64	7.78	6.22	4.67	4.57	4.47	4.02	3.56
0.30	47.65	31.76	15.88	12.90	9.93	8.25	6.57	6.65	6.74	6.05	5.36
0.35	55.26	37.73	20.20	16.87	13.54	11.37	9.19	9.58	9.97	9.17	8.37
0.40	62.87	43.69	24.52	20.83	17.14	14.48	11.82	12.51	13.20	12.29	11.38

Table B.3: corresponds to the figure 3.21

	10	15	20	25	30	35	40	45	50	55	60
0.05	3523.3	3022.8	2522.3	1936.1	1349.9	1359.9	1369.8	1042.5	715.3	713.7	712.1
0.10	2966.4	2363.0	1759.6	1428.4	1097.3	964.3	831.2	658.1	485.0	433.2	381.4
0.15	2795.1	2254.8	1714.6	1399.7	1084.9	926.3	767.6	703.1	638.6	542.0	445.4
0.20	2623.8	2146.7	1669.5	1371.0	1072.6	888.3	704.0	748.1	792.1	650.8	509.4
0.25	2643.6	2217.2	1790.8	1496.3	1201.9	990.8	779.6	804.2	828.7	726.1	623.5
0.30	2663.5	2287.7	1912.0	1621.6	1331.1	1093.2	855.3	860.3	865.2	801.4	737.6
0.35	2917.1	2382.9	1848.6	1685.1	1521.6	1281.9	1042.3	1042.0	1041.6	953.6	865.6
0.40	3170.8	2478.0	1785.2	1748.6	1712.0	1470.6	1229.2	1223.6	1218.1	1105.9	993.7

Table B.4: corresponds to the figure 3.22

	10	15	20	25	30	35	40	45	50	55	60
0.05	3452.3	2892.7	2333.1	1989.0	1644.9	1407.1	1169.3	1006.7	844.2	757.3	670.5
0.10	2421.0	2018.8	1616.6	1317.5	1018.5	849.4	680.4	510.3	340.1	273.9	207.7
0.15	2381.9	1938.0	1494.1	1158.8	823.5	693.1	562.7	439.7	316.7	255.1	193.5
0.20	2342.8	1857.2	1371.6	1000.1	628.6	536.8	445.0	369.2	293.3	236.3	179.3
0.25	2655.3	2038.3	1421.2	1069.1	717.1	604.5	491.9	420.5	349.0	300.6	252.1
0.30	2967.8	2219.3	1470.7	1138.1	805.5	672.2	538.8	471.8	404.8	364.9	325.0
0.35	2857.7	2209.2	1560.8	1258.4	955.9	829.8	703.7	609.1	514.4	485.1	455.7
0.40	2747.5	2199.2	1650.9	1378.6	1106.3	987.5	868.7	746.3	624.0	605.2	586.5

Table B.5: corresponds to the figure 3.23

	10	15	20	25	30	35	40	45	50	55	60
0.05	82.20	75.44	68.68	67.71	66.73	64.74	62.75	61.16	59.57	59.86	60.14
0.10	88.02	85.27	82.52	77.30	72.09	73.95	75.80	73.65	71.50	68.27	65.05
0.15	80.65	74.47	68.29	62.17	56.06	57.23	58.40	55.91	53.42	51.09	48.75
0.20	73.27	63.66	54.06	47.05	40.03	40.51	40.99	38.17	35.35	33.90	32.46
0.25	75.79	65.42	55.06	49.34	43.63	43.03	42.43	42.77	43.10	42.50	41.90
0.30	78.31	67.18	56.06	51.64	47.22	45.55	43.87	47.36	50.86	51.10	51.35
0.35	76.75	67.84	58.93	55.03	51.14	49.58	48.02	48.55	49.07	48.47	47.87
0.40	75.19	68.49	61.80	58.43	55.06	53.62	52.18	49.73	47.28	45.83	44.38

Table B.6: corresponds to the figure 3.24

	10	15	20	25	30	35	40	45	50	55	60
0.05	73.74	69.05	64.37	61.90	59.43	56.95	54.48	54.07	53.65	55.43	57.21
0.10	88.27	86.93	85.60	83.34	81.08	80.91	80.74	80.89	81.03	82.98	84.93
0.15	84.45	81.88	79.31	77.23	75.14	74.20	73.27	73.76	74.26	75.46	76.65
0.20	80.63	76.82	73.02	71.11	69.20	67.50	65.79	66.64	67.49	67.93	68.37
0.25	78.81	75.49	72.17	70.04	67.91	65.99	64.07	64.50	64.92	64.99	65.05
0.30	76.98	74.15	71.31	68.96	66.61	64.48	62.35	62.36	62.36	62.05	61.74
0.35	79.42	77.00	74.58	73.08	71.59	68.48	65.38	65.86	66.34	64.98	63.61
0.40	81.85	79.85	77.84	77.20	76.56	72.49	68.42	69.37	70.31	67.90	65.49

BIBLIOGRAPHY

[1] Siu-Kiu Au. *On the Solution of the First Excurtion Problems by Simulation with Applicationto Probabilistic Seismic Performance*. California Institute of Technology, Pasadena, California, 2001.

[2] Bilal M. Ayyub and Richard H. McCuen. *Probability, Statistics and Reliability for Engineers and Scientists*. Chapman and Hall/CRC, 2002.

[3] Richard Ernest Bellman. *Dynamic programming*. Princeton University Press, 1957.

[4] Richard Ernest Bellman. *Adaptive control processes: a guided tour*. Princeton University Press, 1961.

[5] Deborah J. Bennett. *Randomness*. Harvard University Press, 1998.

[6] H.-G. Beyer and H. P. Schwefel. Evolution strategies - a comprehensive introduction. *Natural Computing*, 1:3–52, 2002.

[7] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of Markov Chain Monte Carlo*. Chapman and Hall/CRC, 2011.

[8] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume Vol. 1. Wiley, 3rd edition, 1968.

[9] Marco Götz. *Entwicklung effizienter numerischer Lösungsverfahren zur fuzzy-stochastischen Optimierung*. TU Dresden, Insitute for structural analysis. Diplomarbeit, 2011.

[10] John Hassal. *The Old Nursery Stories and Rhymes*. Blackie and Son Limited, London, 1909.

[11] Hastings W. K. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57:97–109, 1970.

[12] Doob Joseph L. *Stochastic processes*. New York : Wiley, 1953.

[13] Michael Lavine. *Introduction to Statistical Thought*. 2010.

[14] Gregory F. Lawrel. *Introduction to Stochastic Processes*. Probability Series. Chapman and Hall/CRC, 1995.

[15] Martin Liebscher, Stephan Pannier, Jan-Uwe Sickert, and Wilfgang Graf. Efficiency improvement of stochastic simulation by means of subset sampling. *5. LS-DYNA Anwenderforum, Ulm 2006*, (K - I):27–40, 2006.

[16] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, and A. H. Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[17] Bernd Möller and Michael Beer. *Fuzzy Randomness - Uncertainty in Civil Engineering and Computational Mechanics*. Springer, 2004.

[18] Bernd Möller, Wolfgang Graf, and Michael Beer. Fuzzy structural analysis using α level optimization. *Computational Mechanics*, 26:547–565, 2000.

[19] Athanasios Papoulis. *Probability, Random Variables and Stochastic Processes*. Series in Electrical and Computer Engineering. McGraw-Hill, 4th edition, 2002.

[20] Hartmut Pohlheim. *Examples of Objective Functions*. Genetic and Evolutionary Algorithm Toolbox for Matlab, geatbx version 3.8 edition, 2006.

[21] M.L. Puri and D. Ralescu. Fuzzy random variables. *Journal of Mathematical Analysis and Applications*, (114):409–422, 1986.

[22] B. Russell. *Human Knowledge*. Simon and Schuster, 1948.

[23] Fishman George S. *Monte Carlo : concepts, algorithms, and applications*. New York ; Berlin ; Heidelberg : Springer, 1996.

[24] Frank Steinigen, Jan-Uwe Sickert, Wolfgang Graf, and Michael Kaliske. Fuzzy and fuzzy stochastic methods for the numerical analysis of reinforced concrete structures under dynamical loading. *Computational Methods in Stochastic Dynamics*, (K - I):27–40, 2013.

[25] Australian National University. *ANU Quantum Random Numbers Server*. http://qrng.anu.edu.au.

[26] John Walker. *HotBits: Genuine random numbers, generated by radioactive decay*. http://www.fourmilab.ch/hotbits/, May 1996.

[27] Eckart Zitzler, Marco Laumanns, and Stefan Bleuler. A tutorial on evolutionary multiobjective optimization. *Swiss Federal Institute of Technology (ETH) Zurich, Computer Engineering and Networks Laboratory (TIK)*.