# Algorithms for Structural Reliability

## Computational tools and Machine learning Methods

A dissertation submitted in partial fulfillment of the requirements for the degree of MSc in Analysis and Design of Earthquake Resistant Structures


| | |
|---|---|
| Chair: | Laboratory for Earthquake Engineering |
| Advisor: | Dr. Michalis Fragiadakis, Associate Professor NTUA |
| Submitted by: | Ioannis Prentzas |
| Submission Date: | 02.03.2022 |

# Abstract

Structural reliability analysis provides a useful tool for safety assessment of engineering structures and enables performance of more rational risk evaluations. It is an alternative approach to traditional deterministic structural design, which takes into account the uncertain parameters characterizing the physical state of structure and its environment. Generally, structural reliability analysis is convenient and straightforward when the limit state function is formulated with an explicit function.

However, in practical engineering, the limit state function is generally expressed as implicit function. The implicit limit state function presents great difficulties in structural reliability analysis when the most common methods are used, such as the first-order reliability method (FORM). Typically, when the implicit limit state function is evaluated implicitly using a numerical code, such as the finite element method. Although reliability analysis can be performed using the Monte Carlo simulation or Subset Simulation, a large number of FEM executions for structural analysis is time consuming, especially for large and complex structures with high reliability.

Various regression models in combination with reliability methods have been used to solve reliability analysis problems involving the implicit limit state function. Gaussian process regression, and Support Vector machine are Machine Learning algorithms, which have been applied to approximate the limit state function, shortening the computational time, and the failure probability was predicted using reliability methods, such as Monte Carlo.

The structural reliability methods have been applied to three structural analysis problems for calculating the probability of failure. The limit state equation is explicit in the first example, while the equation of the other examples includes the finite element method. Moreover, these equations have been approximated using the two machine learning regressions models.

# Acknowledgments

First and foremost, I would like to express my sincere gratitude to my supervisor, Associate Prof. Michalis Fragiadakis, for the continuous support that he offered, his patience, motivation, enthusiasm and immense knowledge. I would like to thank him for trusting me to work with him, for his advice, ideas and guidance throughout my master thesis. I could not have imagined having a better supervisor for my thesis.

Besides my supervisor, I would like to thank Zeinep Achmet, PhD student at Laboratory for Earthquake Engineering of NTUA, for her contribution on this thesis.

Finally, I must express my very profound gratitude to my parents, and to my brother for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis.

# Contents

# List of Figures

# List of Tables

# 1. Introduction

## 1.1 Overview

Many sources of uncertainty are inherent in structural design. Despite what we often think, the parameters of the loading and the load-carrying capacities of structural members are not deterministic quantities. They are random variables, and thus absolute safety (or zero probability of failure) cannot be achieved. Consequently, structures must be designed to serve their function with a finite probability of failure.

To illustrate the distinction between deterministic and random quantities, consider the loads imposed on a bridge by car and the truck traffic. The load on the bridge and the weights of the vehicles. As we all know from daily experience, cars and trucks come in many shapes and sizes. Furthermore, the number of vehicles that pass over a bridge fluctuates, depending on the time of day. Since we do not know the specific details about each vehicle that passes over the bridge or the number of vehicles on the bridge at any time, there is some uncertainty about the total load on the bridge. Hence the load is a random variable.

Society expects building and bridges to be designed with a reasonable safety level. In practice, these expectations are achieved by following code requirements specifying design values for minimum strength, maximum allowable deflection, and so on. Code requirements have evolved to include design criteria that take into account some of the sources of uncertainty in design. Such criteria are often referred to as reliability-based design criteria. The objective of this thesis is to provide computational tools to quantify the reliability of structures.

The reliability of a structure is its ability to fulfill its design purpose for some specified lifetime. Reliability is often understood to equal the probability that a structure will not fail to perform its intended function. The term 'failure' does not necessarily mean catastrophic failure but is used to indicate that the structure does not perform as desired.

## 1.2 Chapter layout

**Chapter 2** describes the definition of basic terminologies, such as failure, limit state function, probability of failure, variables and reliability index, of structural reliability analysis. Then, in later section some techniques for structural reliability analysis are represented. These are methods for calculating the probability of failure, which are second-moment and transformations methods, such as First-order second moment reliability method, and Rackwitz-Fiessler procedure and simulation techniques, such as crude Monte Carlo simulation and Subset simulation.

**Chapter 3** investigates the efficiency and the accuracy of the reliability methods, which are presented in chapter 2. For this reason, three structural reliability examples are presented. The first example is a three-span continuous beam with 3 random variable, in which the limit state function is explicit. The second example is a 23-bar 2d truss with 13 random variables, while the performance function includes a finite element model. The last example describes a frame with 11 random variables, while the limit state function involves the finite element method with beam elements.

**Chapter 4** introduces two machine learning algorithms and examines how they will be connected with Structural Reliability Analysis. Two regression models are performed with aim to replace the finite element model in limit state equation. The models are the Gaussian process regression, and Support vector machine. The general purpose is to minimize computational cost and produce compatible results with classical reliability methods.

**Chapter 5** exhibits the connection between Structural Reliability analysis and Machine Learning algorithms for the calculation of the probability of failure. For this purpose, the examples in chapter 3, in which the limit state function involves FEM model, are performed using the regression models. It is crucial to examine the efficiency and the accuracy of this combination and to quantify the shortening of the computational time.

**Chapter 6** summarizes the conclusions that concern the different methodologies used for the reliability analysis as well as the combination between machine learning and structural reliability methods for calculating the probability of failure.

# 2. Structural Reliability Analysis

## 2.1 Introduction

The term structural reliability should be considered as having two meanings, a general one and a mathematical one. In the most general sense, the reliability of a structure is its ability to fulfill its design purpose for some specified time. In a narrow sense, it is the probability that a system will not attain each specified limit state (ultimate or serviceability) during specified reference period. The study of structural reliability is concerned with the calculation and prediction of the probability of limit state violation for an engineered structural system at any stage during its life. In particular, the study of structural safety is concerned with the violation of the ultimate or safety limit states for the structure. More generally, the study of structural reliability is concerned with the violation of performance measures, (of which ultimate or safety limit states are a subset). These include safety of the structure against collapse, limitations on damage, or on deflections or other criteria.

The fundamental variables that define and characterize the behaviour and safety of a structure may be termed the 'basic' variables. In probabilistic assessments any uncertainty about a variable (expressed, in terms of its probability density function) is taken into account explicitly. Typical examples are dimensions, densities or unit weights, materials, loads, material strengths. This is not the case in traditional ways of measuring safety, such as the 'factor of safety' or 'load factor'. These are 'deterministic' measures, since the variables describing the structure, its strength and the applied loads are assumed to take on known (if conservative) values about which there is assumed to be no uncertainty. In dealing with real world problems, uncertainties are unavoidable. The effects of uncertainties on the design and planning of an engineering system are important.

Over the past few decades, many different methods for solving the engineering reliability problem have been developed [4]. In general, the proposed reliability methods can be classified into three categories, namely :

(i) Analytical methods based on Taylor-series expansion of the performance function, such as the First-Order Reliability Method (FORM) and the Second- Order Reliability Method (SORM)

(ii) Monte Carlo simulation methods, such as crude Monte Carlo, Importance Sampling and Subset simulation.

(iii) Surrogate methods are based on a functional surrogate of the performance function, which are presented in chapter 4.

### 2.1.1 Limit State Functions (Performance Functions)

Before we begin with structural reliability analysis, we must first determine how we define the limit state function. The concept of a limit state is used to help define failure in the context of structural reliability analyses [1]. A limit state is a boundary between desired and undesired performance of a structure. The boundary is often represented mathematically by a limit state function or performance function. For example, in bridge structures, failure could be defined as the inability to carry traffic. This undesired performance can occur by many modes of failure: cracking, corrosion, excessive deformations, exceeding load-carrying capacity for shear or bending moment, or local overall buckling. Some members may fail in brittle manner, whereas others may fail in a ductile fashion. In the traditional approach, each mode of failure is considered separately, and each mode can be defined using the concept of a limit state.

A traditional notion of the 'safety margin' is associated with the ultimate limit states. For example, a mode of beam failure could be when the moment due to loads exceeds the moment-carrying capacity. Let $R$ represent the resistance (moment-carrying capacity) and $S$ represent the load effect (total moment applied to the considered beam). It sometimes helpful to think of $R$ as the 'capacity' and $S$ as the 'demand'. A performance functrion, or limit state function, can be defined for this mode of failure as

$$g(R, Q) = R - S \tag{2.1}$$

The limit state, corresponding to the boundary between desired and undesired performance, would be when $g = 0$. If $g \geq 0$, the structure is safe (desired performance); if $g < 0$, the structure is not safe (undesired performance). The probability of failure $p_f$ is equal to the probability that the undesired performance will occur. Mathematically, this can be expressed in terms of the performance function as

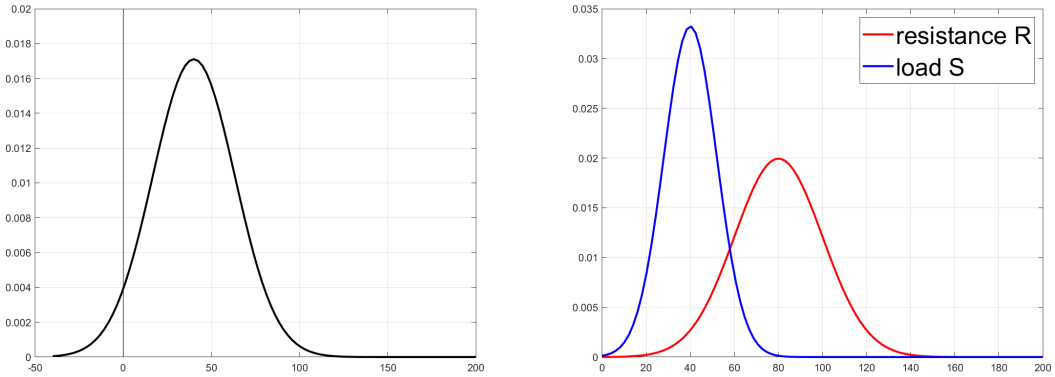
$$p_f = P(R - S < 0) = P(g < 0) \tag{2.2}$$

If both $R$ and $S$ are continuous random variables, then each has a probability density function (PDF) in Fig.1. Furthermore, the quantity $R - S$ is also a random variable with its own PDF. This also shown in Fig.1. The probability of failure corresponds to the shared area in Fig.1.

All realizations of a structure can be put into one of two categories:

- Safe (load effect $\leq$ resistance)
- Failure (load effect $>$ resistance)

The state of structure can be described using various parameters $X_1, X_2, ..., X_n$,




(a) Probability of failure, safety margin S-R

(b) R, resistance and S, load effect

Figure 1: PDF's of load, resistance and safety margin

which are load and resistance parameters such as dead load, live load, length, depth, compressive strength, yield strength, and moment of inertia. A limit state function, or performance function, is a function $g(X_1, X_2, ..., X_n)$ of these parameters such that

- $g(X_1, X_2, , X_n) > 0$ for a safe structure
- $g(X_1, X_2, , X_n) = 0$ border or boundary between safe and unsafe
- $g(X_1, X_2, , X_n) < 0$ for failure

In general, the performance function (limit state function) can be a function of many variables: load components, influence factors, resistance parameters, material properties, dimensions, analysis factor, and so on. A direct calculation of $p_f$ using Eq.2.2 is often very difficult, if not impossible. Therefore, it is convenient to measure structural safety in terms of reliability index.

### 2.1.2 Probability of Failure

Initially, we should understand that the term failure does not necessarily mean catastrophic failure but is used to indicate that the structure does not performed as desired. Next, we examine how to determine the probability of failure for the relatively simple performance function given earlier by

$$g(R, Q) = R - S. \tag{2.3}$$

The probability of failure, $p_f$, can be derived by considering the PDFs of the random variables $R$ and $Q$. Traditional structural analysis process generally starts with the establishment of reasonable mechanical models, and then obtains the responses based on the deterministic structural parameters and certain external loads [1],[2]. However, a number of uncertainties of involved parameters are unneglectable in practical engineering, which may lead the results of reliability analysis to considerable deviations. To assess these uncertainties, the probability-based reliability analysis methods spring up and then are maturely applied to a number of sectors and societies. The main work in structural reliability analysis is to obtain the failure probability by solving the following multi-dimensional integration:

$$p_f = P(g(x) < 0) = \int_{g(x)<0} f_X dX, \tag{2.4}$$

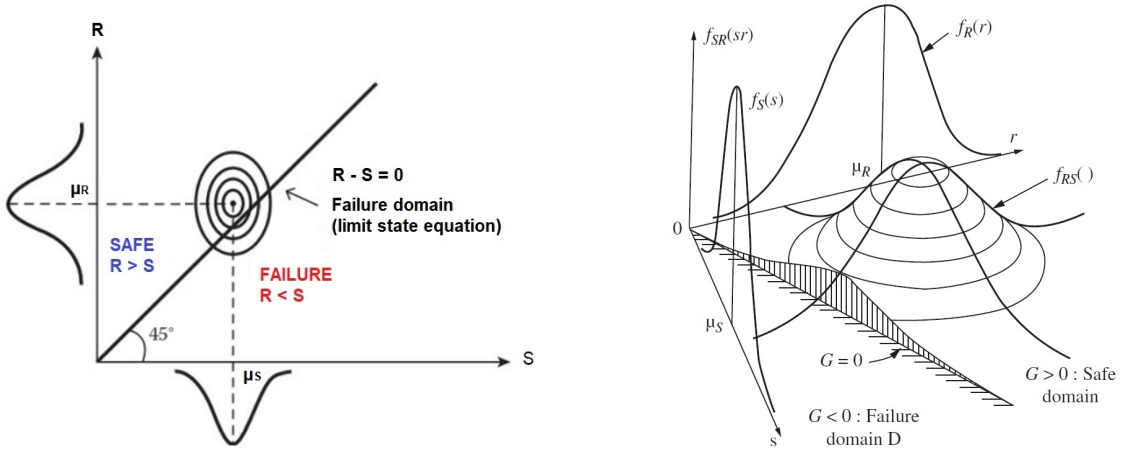
where $P(\cdot)$ is the probability, $X = (X_1, X_2, ..., X_n)$ are the basic random variables, $g(x)$ is the limit state function (LMS) and $f_X(X)$ the is the joint probability density function (PDF) of $X$.

### 2.1.3 Space of State Variables

To begin our analysis, we need to define the state variables of the problem. The state variables are the basic load and resistance parameters used to formulate the performance function. For n state variables, the limit state function is a function of n parameters.

If all loads (or load effects) are represented by the variable S ant the total resistance (or capacity) by R, then the space of state variable is a two-dimensional space as shown in Fig.2a. Within this space, we can seperate the 'safe domain' from the failure domain; the boundary between the two domains is described by the limit state function $g(R, S) = 0$.

Since both R and Q are random variables, we can define a joint density function $f_{R,S}(r, s)$. A general joint density is plotted in Fig.2b. Again, the limit state function separates the safe and the failure domains. The probability of failure is calculated by integration of the joint density function over the failure domain. As noted earlier, this probability is often very difficult to evaluate, so the concept of a reliability index is used to quantify structural reliability.




(a) Safe domain and failure domain in a two-dimensional state space.

(b) Three-dimensional representation of a possible joint density function $f_{RS}$.

Figure 2: Two- and three- dimensional representation of the state space

### 2.1.4 Definition of Reliability Index

The reliability is defined as the shortest distance from the origin of reduced variables to the line $g(Z_R, Z_S) = 0$. This definition, which was introduced by Hasofer and Lind (1974) [5], is illustrated in Fig.3.

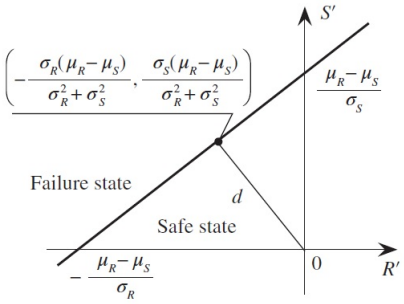



Figure 3: Reliability index defined as the shortest distance in the space of reduced variables

.

Using geometry, we can calculate the reliability index (shortest distance) from the following formula:

$$\beta = \frac{\mu_R - \mu_S}{\sqrt{\sigma_R^2 + \sigma_S^2}} \tag{2.5}$$

where $\beta$ is the inverse of the coefficient of variation of the function $g(R, S) = R - S$ when R and S are uncorrelated. For normally distributed random variables R and S, it can be shown that the reliability index is related to the probability of failure by

$$\beta = -\Phi(p_f) \quad p_f = \Phi(-\beta) \tag{2.6}$$

The definition for a two-variable case can be generalized for n variables as follows. Consider a limit state function $g(X_1, X_2, ..., X_n)$ where the $X_i$ variables are all uncorrelated.

The Hasofer-Lind reliability index is defined as follows:

1. Define the set of reduced variables $\{Z_1, Z_2, ..., Z_n\}$

$$Z_i = \frac{X_i - \mu_{X_i}}{\sigma_{X_i}} \tag{2.7}$$

2. Redefine the limit state function by expressing it in terms of the reduced variables $\{Z_1, Z_2, ..., Z_n\}$.

3. The reliability index is the shortest distance from the origin in the n-dimensional space of reduced variables to the curve described $g(Z_1, Z_2, ..., Z_n) = 0$.

## 2.2 Second-Moment and Transformations Methods

### 2.2.1 First-Order Second-Moment Reliability Index

The above ideas are readily extended in the case where the limit state function is a random function consisting of more than two basic random variables. Consider a **linear** limit state function of the form

$$g(X_1, X_2, ..., X_n) = \alpha_0 + \alpha_1 X_1 + ... + \alpha_n X_n = \alpha_0 + \sum_{i=1}^{n} \alpha_i X_i \tag{2.8}$$

where the $\alpha_i$ terms $(i = 0, 1, 2, ..., n)$ are constants and the $X_i$ terms are uncorrelated random variables. If we apply the three-step procedure outlined above for determining the Hasofer-Lind reliability index, we would obtain the following expression for $\beta$:

$$\beta = \frac{\alpha_0 + \sum_{i=1}^{n} \alpha_i \mu_{X_i}}{\sqrt{\sum_{i=1}^{n} \alpha_i {\sigma_{X_i}}^2}} \tag{2.9}$$

Observe that the reliability index, $\beta$ in Eq.2.9 depends only on the means and standard deviations of the random variables. Therefore, this $\beta$ is called a second-moment measure of structural safety because only the first two moments (mean and variance) are calculated $\beta$.

Now consider the case of a **nonlinear** state function. When the function is nonlinear, we can obtain an approximate answer by linearizing the nonlinear function using a Taylor series expansion. The result is

$$g(X_1, X_2, ..., X_n) = g(x_1^*, x_2^*, ..., x_n^*) + \sum_{i=1}^{n} (X_i - x_i^*) \frac{\partial g}{\partial X_i}\bigg|_{(x_1^*, x_2^*, ..., x_n^*)} \tag{2.10}$$

where $(x_1^*, x_2^*, ..., x_n^*)$ is the point about which the expansion is performed. One choice for this linearization point is the point corresponding to the mean values of the random variables. Thus equation 2.10 becomes

$$g(X_1, X_2, ..., X_n) \approx g(\mu_{x_1}, \mu_{x_2}, ..., \mu_{x_n}) + \sum_{i=1}^{n} (X_i - \mu_{X_i}) \frac{\partial g}{\partial X_i}\bigg|_{\text{evaluated at mean values}} \tag{2.11}$$

Since equation 2.11 is a linear function of the $X_i$ variables, it can be rewritten to look exactly like Eq.2.8. Thus equation Eq.2.9. can be used as an approximate solution for the reliability index $\beta$. After some algebraic manipulations, the following expressions

for $\beta$ results:

$$\beta = \frac{g(\mu_{x_1}, \mu_{x_2}, ..., \mu_{x_n})}{\sqrt{\sum_{i=1}^{n} \alpha_i {\sigma_{X_i}}^2}} \quad \text{where} \quad \alpha_i = \left.\frac{\partial g}{\partial X_i}\right|_{\text{evaluated at mean values}} \tag{2.12}$$

The reliability index defined in equation Eq.2.12 is called a first-order-moment mean value reliability index.

**First order**: first-order terms in the Taylor series expansion.

**Second moment**: only means and variances are needed.

**Mean value**: the Taylor series expansion is about the mean values.

### 2.2.2 Sensitivity Measures

The symbol $a_i$ is called direction cosines, and it is given by Eqn. 2.12, while it represents the sensitivity of the standardized limit state function $g$ at $X^*$ to changes in $X$. This sensitivity has an important practical implication. Thus if the sensitivity $a_i$ to $X_i$, say, is low there is little need to be very accurate about the determination of $X_i$. Also it would signal that, if necessary, $X_i$ might well be treated as a deterministic rather than a random variable. More precisely, knowing how an input variable of the reliability problem may influence the probability of failure can give the analyst valuable information. This variable can either be a distribution parameter of a random variable or a parameter of the limit state function. This reduces the dimensionality of the space of random variables. Within the field of structural reliability, the sensitivity of the reliability or the probability of failure of a structure or a system is of great interest.

A measure in the context of reliability structural analysis is denoted as local when the sensitivity of the probability of failure is measured around a point of interest, such as **Importance Factors**. The Importance Factors are a byproduct of the FORM algorithm. Consider the design point $U^*$ and the reliability index $\beta$. The design point is defined as the point on the limit state surface closest to the origin, measured in the standard normal space. Further, the reliability index is the distance between the design point $U^*$ and the origin. Using these two definitions, the important direction $\alpha$ can be defined as:

$$U^* = \beta\alpha \tag{2.13}$$

As already outlined in previous section, a direct formulation of $\alpha$ is given as the normalized gradient of the limit state function evaluated at the design point.

$$\alpha = -\frac{\nabla G_{|u^*}}{\|\nabla G_{|u^*}\|} \tag{2.14}$$

Eventually, the Importance Factors $IF_i$ are readily available from the unit important

vector:

$$IF_i = \alpha_i^2 \tag{2.15}$$

The Importance Factors have long been the standard in reliability-sensitivity analysis. They offer an indication of the important parameters (once the problem is cast in the standard normal space) at no cost.

### 2.2.3 First-Order Reliability Method (FORM)- Hasofer-Lind Reliability Index

In 1974, Hasofer and Lind [5] proposed a modified reliability index, which is a useful (but not essential) first step is to transform all variables to their standardized form. The 'correction' is to evaluate the limit state function at a point known as thew 'design point' instead of mean values. The design point is a point on the failure surface 'g=0'. Since the design point is generally not known a priori, an iteration technique must be used to solve the reliability index.

Consider a limit state function $g(X_1, X_2, ..., X_n)$, where the random variable $X_i$ are all uncorrelated. The limit state function is rewritten in terms of the standard form of the variables (reduced variables), $N(0, 1)$, using

$$Z_i = \frac{X_i - \mu_{x_i}}{\sigma_i} \tag{2.16}$$

As before, the Hasofer-Lind reliability index is defined as the shortest distance from the origin of the reduced variable space to the limit state function $g = 0$. First-order

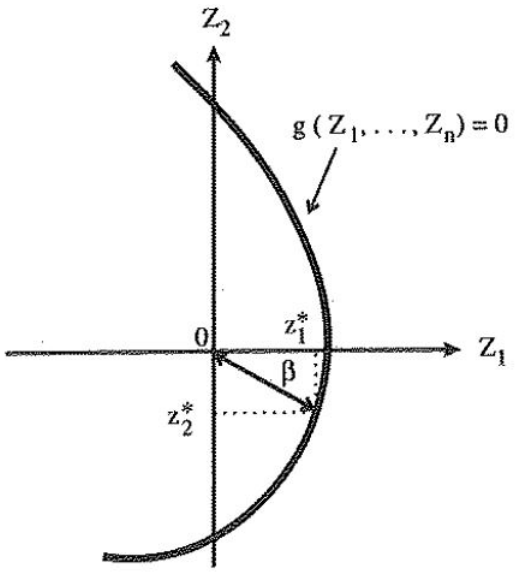



Figure 4: Design point and reliability index for a highly nonlinear limit state function.

reliability method (FORM) is considered to be a reliable computational method for

structural reliability. Its accuracy is generally dependent on three parameters, i.e. the curvature radius at the design point, the number of random variables and the first-order reliability index.

FORM is an analytical approximation in which the reliability index is interpreted as the mini-mum distance from the origin to the limit state surface in standardized normal space (u-space). Because the performance function is approximated by a linear function in u-space at the design point, accuracy problems occur when the performance function is strongly nonlinear.

The matrix procedure [1] consists of the following steps:

1. Formulate the limit state function and appropriate parameters for all random variables $X_i (i = 1, 2, ..., n)$ involved.

2. Obtain an initial design point $x_i^*$ by assuming values for $n - 1$ of the random variables $X_i$ (Mean values are often a reasonable initial choice). Solve the limit state equation $g = 0$ for the remaining random variable. This ensure that the design point is on the failure boundary.

3. Determine the reduced variables $z_i^*$ corresponding to design point $\{x_i^*\}$ using

$$z_i^* = \frac{x_i^* - _i}{\sigma_{x_i}} \tag{2.17}$$

4. Determine the partial derivatives of the limit state function with respect to the reduced variables. Define a column vector $\{G\}$ as the vector whose elements are these partial derivatives multiplied by -1:

$$\{G\} = \begin{Bmatrix} G_1 \\ G_2 \\ \vdots \\ G_n \end{Bmatrix} \quad \text{where} \quad G_i = \frac{\partial g}{\partial Z_i} \tag{2.18}$$

5. Calculate an estimate of $\beta$ using the folowing formula:

$$\beta = \frac{\{G\}^T \{z^*\}}{\sqrt{\{G\}^T \{G\}}} \quad \text{where} \quad \{z^*\} = \begin{Bmatrix} z_1^* \\ z_2^* \\ \vdots \\ z_n^* \end{Bmatrix} \tag{2.19}$$

6. Calculate a column vector containing the sensitivity factors using

$$\alpha = \frac{\{G\}}{\sqrt{\{G\}^T\{G\}}} \tag{2.20}$$

7. Determine a new design point in reduced variables for $n-1$ of the variables using

$$z_i^* = a_i\beta \tag{2.21}$$

8. Determine the corresponding design point values in original coordinates for the $n-1$ values in Step 7 using

$$x_i^* = \mu_{x_i} + z_i^*\sigma_{x_i} \tag{2.22}$$

9. Determine the value of the remaining random variable (i.e., the one not found in Steps 7 and 8) by solving the limit state function $g = 0$.

10. Repeat Steps 3 to 9 until $\beta$ and the design point $\{x_i^*\}$ converge.

#### 2.2.3.1 Example: RC section - Hasofer Lind

Assume an RC section with tension reinforcement $A_s$ subjected to moment $M_{ED}$. The moment capacity of the section is:

$$M_{RD} = A_s f_y (d - 0.59 \frac{A_s f_y}{f_c b}) \tag{2.23}$$

where $A_s$ is the area of steel, $f_y$ is the yield strength of the steel, $f_c$ is the compressive strength of the concrete, b=25cm is the width of the section, and d is the depth (d=28cm) of the section.

We want to examine the limit state of ecxeeding the beam capacity in bending. The limit state function would be

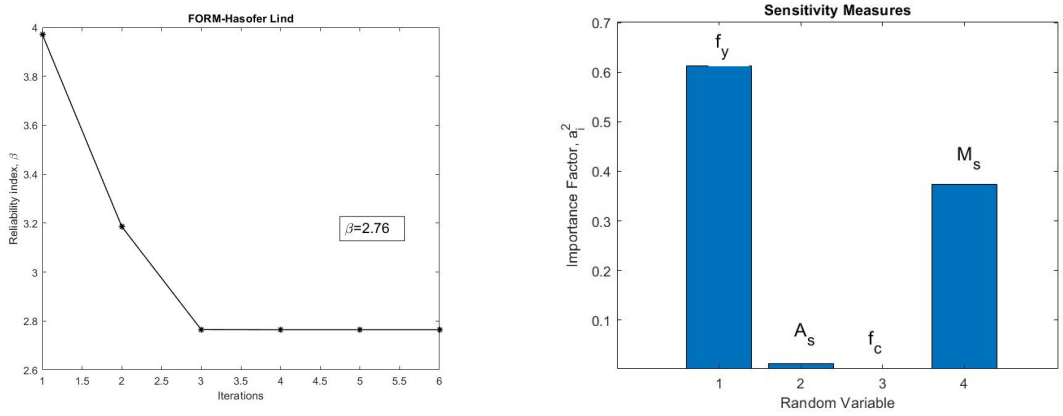
$$g(A_s, f_y, f_c, Q) = A_s f_y (d - 0.59 \frac{A_s f_y}{f_c b}) - M_{ED} \tag{2.24}$$

where $M_{ED}$ is the moment (load effect) due to the applied load. The random variables in the problem are $M_{ED}, f_y, f_c$, and $A_s$. The distribution parameters and design parameters are given in the table 2.

Table 1: Distribution parameters of basic variables

| Random variable | Mean | c.o.v. | Standard deviation |
|---|---|---|---|
| $f_y$ | 300 MPa | 10.5% | 31.5 MPa |
| $A_s$ | 0.0026 $m^2$ | 2% | 0.000052 $m^2$ |
| $f_c$ | 20 MPa | 14% | 2.8 MPa |
| $M_s$ | 230 kNm | 12% | 27.6 kNm |

The failure probability ($p_f = 8.8 \times 10^{-3}$) have been calculated using **FORM-HL**. The convergence plot is shown in Fig.5a. Moreover, the importance factor for every random variable is defined in Fig.5b.




(a) Convergence plot

(b) Importance Factors

Figure 5: Iteration process and Sensitivity measures.

#### 2.2.3.2 MATLAB Script - FORM-HL

A MATLAB function (FORM_HLRF), which solves example 2.2.3.1 using FORM-HLRF method, exists in the appendix.

```
clear; clc; close all;
1. m_fy=300*10^-3; m_As=0.0026;
2. m_fc=20*10^-3; m_Med=230;
3. s_fy=31.5*10^-3; s_As=0.000052;
4. s_fc=2.8*10^-3; s_Med=27.6;
5. Properties=[m_fy s_fy 1
6.             m_As s_As 2
7.             m_fc s_fc 3
8.             m_Med s_Med 4];
9.% Create distribution objects
10.fy=makedist('Normal','mu',m_fy,'sigma',s_fy);
11.As=makedist('Normal','mu',m_As,'sigma',s_As);
12.fc=makedist('Normal','mu',m_fc,'sigma',s_fc);
13.Med=makedist('Normal','mu',m_Med,'sigma',s_Med);
14.
15.dist=[fy As fc Med];
16.% Zero value in the RV_list declares
17.% that the variable is considered as deterministic.
18.RV_list=[1;2;3;4];
19.% InitialRV is the r.v., which is calculated from the limit state eqn.
20.initialRV=4;
21.FORM_HLRF(RV_list,Properties,initialRV,dist)
```

### 2.2.4 Rackwitz-Fiessler Procedure

We have now covered methods for calculating reliability indexes using information on the means and standard deviations of the random variables. Detailed information on the type of distribution for each random variable was not needed. In this section, we introduce the Rackwitz-Fiessler procedure for calculating reliability indexes. The method requires the knowledge of the probability distributions for all the variables involved. Rackwitz–Fiessler method is an efficient way to solve the non-normal reliability problems by transforming original non-normal variables into equivalent normal variables based on the equivalent normal conditions. The basic idea behind the procedure begins with the calculation of 'equivalent normal' values of the mean and standard deviation for each non-normal random variable.

We use these equivalent normal parameters in our analysis. Suppose that a particular random variable $X$ with mean $\mu_x$ and $\sigma_x$ is described by a cumulative distribution function $F_X(x)$ and a probability density function $f_X(x)$. To obtain the equivalent normal mean $\mu_X^e$ and standard deviation $\sigma_X^e$, we require that the CDF and PDF of the actual function be equal to the normal CDF and normal PDF at the value of the variable $x^*$ on the failure boundary described by $g = 0$. Mathematically, these requirements are expressed as

$$F_X(x^*) = \Phi(\frac{x^* - \mu_X^e}{\sigma_X^e}) \tag{2.25}$$

$$f_X(x^*) = \frac{1}{\sigma_X^e}\phi(\frac{x^* - \mu_X^e}{\sigma_X^e}) \tag{2.26}$$

where $\Phi$ is the CDF for the standard normal distribution $\phi$ is the PDF for the standard normal distribution.

By manipulating equations 2.25, 2.26, we can obtain expressions for $\mu_X^e$ and $\sigma_X^e$ as follows:

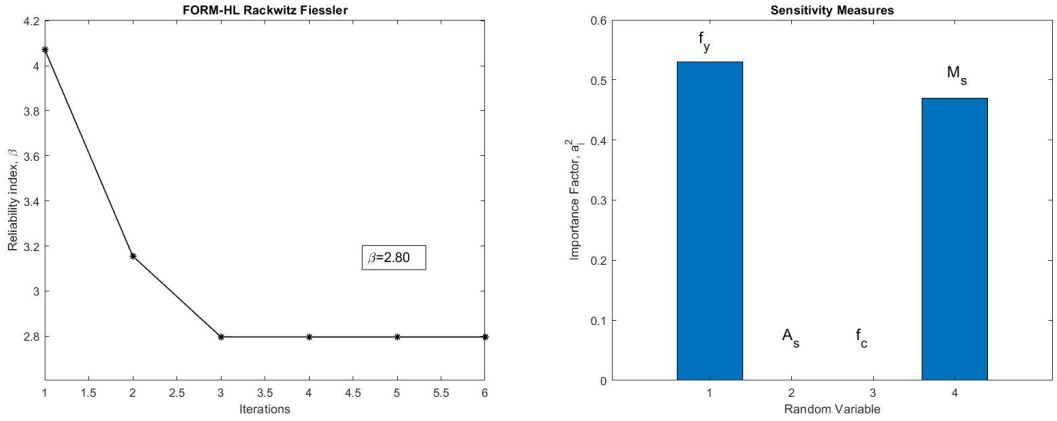
$$\mu_X^e = x^* - \sigma_X^e[\Phi^{-1}(F_X(x^*))] \tag{2.27}$$

$$\sigma_X^e = \frac{1}{f_X(x^*)}\phi[\Phi^{-1}(F_X(x^*)] \tag{2.28}$$

#### 2.2.4.1 Example: RC section - Rackwitz Fiessler

Assume the same problem as in 2.2.3.1, but now $f_y$, $A_s$, and $f_c$ is a lognormal variable and $M_{ED}$ follows extreme value distribution.

The probability of failure is calculated using FORM and Rackwitz and Fiessler transformation to define the equivalent normal's distribution parameters. The convergence plot is similar to FORM-HL, while the reliability index is 2.80.




(a) Convergence plot

(b) Importance Factors

Figure 6: Iteration process and Sensitivity measures.

#### 2.2.4.2 MATLAB Script - FORM-HLRF

A MATLAB function (FORM_HLRF), which solves example 2.2.4.1 using FORM-HLRF method, exists in the appendix.

```
clear; clc; close all;
m_fy=300*10^-3; m_As=0.0026;
m_fc=20*10^-3; m_Med=230;
s_fy=31.5*10^-3; s_As=0.000052;
s_fc=2.8*10^-3; s_Med=27.6;
Properties=[m_fy s_fy 1
            m_As s_As 2
            m_fc s_fc 3
            m_Med s_Med 4];
% Create distribution objects
fy_log_std = sqrt(log(1 + s_fy^2));
fy_log_mean = log(m_fy) - 0.5*(fy_log_std^2);
fy=makedist('Lognormal','mu',fy_log_mean,'sigma',fy_log_std);
As_log_std = sqrt(log(1 + s_As^2));
As_log_mean = log(m_As) - 0.5*(As_log_std^2);
As=makedist('Lognormal','mu',As_log_mean,'sigma',As_log_std);
fc_log_std = sqrt(log(1 + s_fc^2));
fc_log_mean = log(m_fc) - 0.5*(fc_log_std^2);
fc=makedist('Lognormal','mu',fc_log_mean,'sigma',fc_log_std);
Med=makedist('Normal','mu',m_Med,'sigma',s_Med);

dist=[fy As fc Med];
% Zero value in the RV_list declares
% that the variable is considered as deterministic.
RV_list=[1;2;3;4];
% InitialRV is the r.v., which is calculated from the limit state eqn.
initialRV=4;
FORM_HLRF(RV_list,Properties,initialRV,dist)
```

## 2.3 Integration and Simulation Techniques

The previous sections have given an overview of the basic principles of structural reliability theory and it has been presented analytical methods (FORM) for solving the reliability integral. In this section numerical integration methods, such as Monte Carlo and Subset Simulation, will be explored.

### 2.3.1 Crude Monte Carlo Simulation

As the name implies, Monte Carlo simulation techniques involve 'sampling' at 'random' to simulate artificially a large number of experiments and to observe the result. The Monte Carlo method is a special technique that we can use to generate some results numerically without actually doing any physical testing [2]. This method perfectly fits to complex problems for which closed-form solutions are either not possible or extremely difficult. Probabilistic problems involving complicated nonlinear finite element models can be solved by Monte Carlo simulation with the requirement of the necessary computing power is available. Moreover, this simulation technique can solve the original problem without simplifying assumptions, thus the obtained results are more realistic, as the dimensions of the problems do not affect the method. It could also be used to verify the results of other solution methods.

The Monte Carlo Simulation (MCS), is a method to calculate numerically the reliability integral that defines the failure probability $p_f$. In the case of analysis for structural reliability, this means, in the simplest approach, sampling each random variable $X_i$ randomly to give a sample value $\hat{x}_i$. The limit state function $g(x) = 0$ is then checked using the sample set of values $\hat{x}_i$. If the limit state function is violated ($g(\hat{x}_i) \leq 0$), the structure or structural element has 'failed'. The experiment is repeated many times, each time with a randomly chosen vector $\hat{x}$ of $\hat{x}_i$ values.

From a mathematical point of view, crude MC allows to estimate the expected value of a quantity of interest. More specically, suppose the goal is to evaluate $E_\pi[h(x)]$, that is an expectation of a function $h : X \to R$ with respect to PDF $\pi(x)$,

$$E_\pi[h(x)] = \int_X h(x)\pi(x)\, dx. \tag{2.29}$$

The idea behind crude MC is a straightforward application of the law of large numbers that states that if $x^{(1)}, x^{(2)}$,... are i.i.d. (independent and identically distributed) from the PDF $\pi(x)$, then the empirical average $\frac{1}{N}\sum_{i=1}^{N} h(x^{(i)})$ converges to the true value $E_\pi[h(x)]$ as N goes to $+\infty$. Therefore, if the number of samples N is large enough, then $E_\pi[h(x)]$ can be accurately estimated by the corresponding empirical

average:

$$E_\pi[h(x)] = \frac{1}{N}\sum_{i=1}^{N} h(x^{(i)}) \tag{2.30}$$

The relevance of crude MC to the reliability problem follows from a simple observation that the failure probability $p_F$ can be written as an expectation of the indicator function, namely:

$$p_F = \int_F \pi(x)\, dx = \int_X I_F \pi(x)\, dx = E_\pi[I_F(x)], \tag{2.31}$$

where X denotes the entire input x-space. Therefore, the failure probability can be estimated using the crude MC method as follows:

$$p_F \approx \widehat{p_F}^{MC} = \frac{1}{N}\sum_{i=1}^{N} I_F(x^{(i)}), \tag{2.32}$$

where $x^1, ..., x^N$ are samples from $\pi(x)$ and $I_F(x)$ stands the indicator function,

$$I_F(x) = \begin{cases} 1, & \text{if } x \in F. \\ 0, & \text{if } x \notin F. \end{cases} \tag{2.33}$$

The crude MC estimate of $p_F$ is thus just the ratio of the total number of failure samples $\sum_{i=1}^{N} I_F(x^{(i)})$, i.e., samples that produce system failure according to the system model, to the total number of samples, N. Note that ${p_F}^{MC}$ is an unbiased random estimate of the failure probability, that is, on average, ${p_F}^{MC}$ equals to $p_F$ . Mathematically, this means that $E[{p_F}^{MC}] = p_F$. Indeed, using the fact that $x^i \sim \pi(x)$ and Eq.2.32,

$$\begin{aligned} E[{p_F}^{MC}] &= E[\sum_{i=1}^{N} I_F(x^{(i)})] \\ &= \sum_{i=1}^{N} I_F(x^{(i)}) \\ &= \sum_{i=1}^{N} I_F(x) = p_F \end{aligned} \tag{2.34}$$

The main advantage of MC is that the accuracy does not depend on the dimension d of the input space. In reliability analysis, the standard measure accuracy of an unbiased estimate $\widehat{p_F}$ of the failure probability is its coefficient of variation $\delta(\widehat{p_F})$, which is defined as the ratio of the standard deviation to the expected value of $\widehat{p_F}$. The smaller of the c.o.v. $\delta(\widehat{p_F})$, the more accurate the estimate $\widehat{p_F}$ is. The c.o.v. of the

crude MC is estimated by:

$$\begin{aligned} V[{p_F}^{MC}] &= V[\sum_{i=1}^{N} I_F(x^{(i)})] \\ &= \sum_{i=1}^{N} V[I_F(x^{(i)})] \\ &= \sum_{i=1}^{N} (E[I_F(x^{(i)})^2] - E[I_F(x^{(i)})]^2) \\ &= \sum_{i=1}^{N} (p_F - {p_F}^2) = \frac{p_F(1-p_F)}{N} \end{aligned} \tag{2.35}$$

Here, the identity $I_F(x)^2 = I_F(x)$. Using equations 2.35 and 2.34, the c.o.v. of crude MC estimate can be calculated :

$$\delta(\widehat{p_F}) = \frac{\sqrt{V[{p_F}^{MC}]}}{E[{p_F}^{MC}]} = \sqrt{\frac{1-p_F}{Np_F}} \tag{2.36}$$

As it obvious from the above equation, the $\delta(\widehat{p_F})$ depends only on the failure probability $p_F$ and the number of samples and does not depend of the dimensions, d, of the input space.

Nevertheless, crude MC has a significant disadvantage, which is the inefficiency of calculating small failure probabilities. For typical engineering reliability problem, the failure probability $p_F$ is very small, $p_F \ll 1$. In the reliability literature, $p_F \sim 10^{-2}$–$10^{-9}$ have been considered. If $p_F$ is very small, then it follows that

$$\delta(\widehat{p_F}) \approx \frac{1}{\sqrt{Np_F}}. \tag{2.37}$$

#### 2.3.1.1 Example-MC

Assume the same problem as in 2.2.3.1, but now we calculate the probability of failure using Monte Carlo simulation. The coefficient of variance is considered as 10%, thus the number of simulations is 100.000. The probability of failure is $p_F = 9 \times 10^{-3}$, while in Fig.7 is represented the histogram of the failure function.

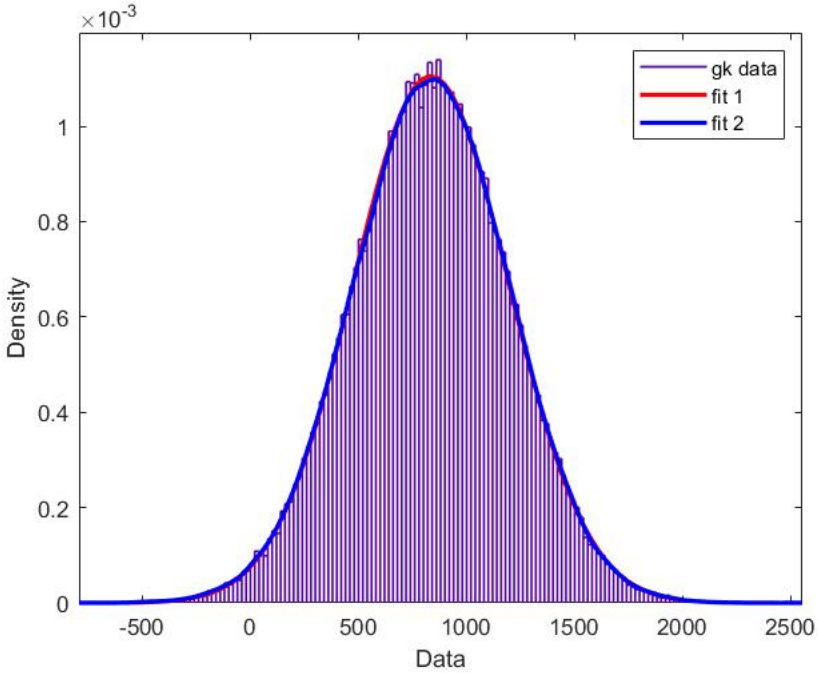



Figure 7: Histogram of $g_{failure}$ data
fit 1: Normal
fit 2: Non-parametric

#### 2.3.1.2 MATLAB Script - MC

A MATLAB function (MC), which solve example 2.3.1.1 using MC simulation, exists in the appendix.

```
clear; clc; close all;
1. m_fy=300*10^-3; m_As=0.0026;
2. m_fc=20*10^-3; m_Med=230;
3. s_fy=31.5*10^-3; s_As=0.000052;
4. s_fc=2.8*10^-3; s_Med=27.6;
5. Properties=[m_fy s_fy 1
6.             m_As s_As 2
7.             m_fc s_fc 3
8.             m_Med s_Med 4];
9.% If s_=0 the variable is considered as deterministic.
10.% Create distribution objects
11.fy=makedist('Normal','mu',m_fy,'sigma',s_fy);
12.As=makedist('Normal','mu',m_As,'sigma',s_As);
13.fc=makedist('Normal','mu',m_fc,'sigma',s_fc);
14.Med=makedist('Normal','mu',m_Med,'sigma',s_Med);
15.
16.dist=[fy As fc Med];
17.N_MC_Simulations=100.000;
18.MC(Properties,dist,N_MC_Simulations)
```

### 2.3.2 Subset Simulation method

**Subset simulation** is a method used in reliability engineering to compute small failure probabilities encountered in engineering systems, while it overcomes difficulties, which are invoke in Monte Carlo simulation. This is a relatively new method, which is proposed by Au and Beck (2001) [3]. It is an efficient and elegant method to compute small failure probabilities encountered in reliability analysis of engineering systems. The basic idea is to express the failure probability as a product of larger conditional failure probabilities by introducing intermediate failure events. With a proper choice of the conditional events, the conditional failure probabilities can be made sufficiently large so that they can be estimated by means of simulation with a small number of samples. The original problem of calculating a small failure probability, which is computationally demanding, is reduced to calculating a sequence of conditional probabilities, which can be readily and efficiently estimated by means of simulation. The conditional probabilities cannot be estimated efficiently by a standard Monte Carlo procedure, however, and so a Markov chain Monte Carlo simulation (MCS) technique based on the Metropolis algorithm is presented for their estimation. This powerful method is robust to the number of uncertain parameters and efficient in computing small probabilities.

Subset Simulation is essentially based on two different ideas: conceptual and technical. The conceptual idea is to decompose the rare event $F$ into a sequence of progressively $'less - rare'$ nested events,

$$F = F_m \subset F_{m-1} \subset \ldots \subset F_1 \tag{2.38}$$

where $F_1$ is a relatively frequent event. For example suppose that $F$ represents the event of getting exactly $m$ heads when flipping a fair coin $m$ times. If $m$ is large, then $F$ is a rare event. To decompose $F$ into a sequence (2.38), let us define $F_k$ to be the event of getting exactly $k$ heads in the first $k$, the less rare the corresponding event $F_k$; and $F_1$ getting heads in the first flip-is relatively frequent.

Given a sequence of subsets (2.38), the small probability $\mathbb{P}(F)$ of the rare event $F$ can then be presented as a product of larger probabilities as follows:

$$\begin{aligned} \mathbb{P}(F) &= \mathbb{P}(F_m) \\ &= \mathbb{P}(F_1)\frac{\mathbb{P}(F_2)\mathbb{P}(F_3)}{\mathbb{P}(F_1)\mathbb{P}(F_2)} \cdots \frac{\mathbb{P}(F_{m-1})\mathbb{P}(F_m)}{\mathbb{P}(F_{m-2})\mathbb{P}(F_{m-1})} \\ &= \mathbb{P}(F_1)\mathbb{P}(F_2|F_1)\ldots\mathbb{P}(F_m|F_{m-1}) \end{aligned} \tag{2.39}$$

where $\mathbb{P}(F_k|F_{k-1}) = \mathbb{P}(F_k)\mathbb{P}(F_{k-1})$ denotes the conditional probability of event $F_k$ given the occurrence of event $F_{k-1}$, for $k = 2, \ldots, m$.

In real applications, it is often not obvious how to decompose the rare event into a sequence (2.38) and how to compute all conditional probabilities in (2.40). In Subset Simulation, the 'sequencing' of the rare events is done adaptively as the algorithm proceeds. This is achieved by employing Markov Chain Monte Carlo, an advanced simulation technique, which constitutes the second-technical-idea behind SS. Finally, all conditional probabilities are automatically obtained as by-product of the adaptive sequencing.

Unlike crude Monte Carlo, where all computational resources are directly spent on sampling the input space, $x^{(1)}, \ldots, x^{(N)} \sim \pi(\cdot)$, and computing the values of the performance function $g(x^{(1)}), \cdots, g(x^{(N)})$, Subset Simulation first 'probes' the input space X by generating a relatively small number of i.i.d samples ${x_0}^{(1)}, \ldots, {x_0}^{(n)} \sim \pi(x), n < N$, and computing the corresponding system responses ${y_0}^{(1)} = g({x_0}^{(1)}), \ldots, \geq g({x_0}^{(n)})$. Here, the subscript 0 indicates the $0^{th}$ stage of the algorithm. Since F is a rare event and n is relatively small, it is very likely that none of the samples ${x_0}^{(1)}, \ldots, {x_0}^{(n)}$ belongs to F, that ${y_0}^{(i)} < y^*$ for all $i = 1, \ldots, n$. Nevertheless, these Monte Carlo samples contain some useful information about the failure domain that can be utilized. To keep the notation simple, assume that ${y_0}^{(1)}, \cdots, {y_0}^{(n)}$ are arranged in the decreasing order, i.e. ${y_0}^{(1)} \geq \cdots \geq {y_0}^{(n)}$ (it is always possible to achieve this by renumbering ${x_0}^{(1)}, \cdots, {x_0}^{(n)}$ if needed). Then, ${x_0}^{(1)}$ and ${x_0}^{(n)}$ are, respectively, the closest to failure and the safest samples among ${x_0}^{(1)}, \ldots, {x_0}^{(n)}$, since ${y_0}^{(1)}$ and ${y_0}^{(n)}$ are the largest and the smallest responses. In general, the smaller i, the closer to failure the sample ${x_0}^{(i)}$ is. This is shown schematically in Fig.8.

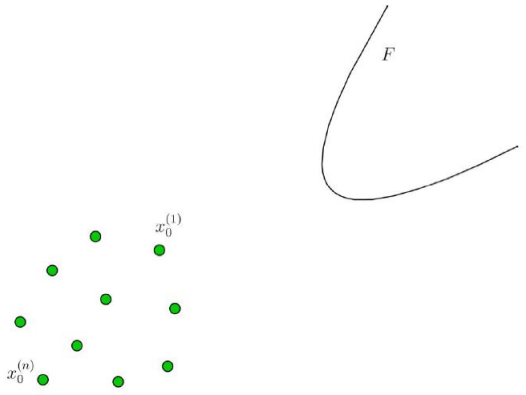



Figure 8: Monte Carlo samples ${x_0}^{(1)}, \cdots, {x_0}^{(n)}$ and the failure domain F. ${x_0}^{(1)}$ and ${x_0}^{(n)}$ are, respectively, the closest to failure and the safest samples among ${x_0}^{(1)}, \cdots, {x_0}^{(n)}$

Let $p \in (0, 1)$ be any number such that $np$ is integer. Next, the fist intermediate failure

domain is defined $F_1$ as follows:

$$F_1 = \{x : g(x) > {y_1}^1\} \tag{2.40}$$

where

$${y_1}^1 = \frac{{y_0}^{np} + {y_0}^{np+1}}{2} \tag{2.41}$$

In other words, $F_1$ is the set of inputs that lead to the exceedance of the relaxed threshold ${y_1}^1 < y^*$. Note that by construction, samples ${x_0}^{(1)}, \cdots, {x_0}^{(np)}$ belong to $F_1$, while ${x_0}^{(np+1)}, \cdots, {x_0}^{(n)}$ do not. As a consequence, the crude Monte Carlo estimate for the probability of $F_1$ which is based on samples ${x_0}^{(1)}, \cdots, {x_0}^{(n)}$ is automatically equal to p,

$$\mathbb{P}_{F_1} \approx \frac{1}{n}\sum_{i=1}^{n} I_{F_1}({x_0}^{(i)}) = p. \tag{2.42}$$

The value $p = 0.1$ is often used in the literature, which makes $F_1$ a relatively frequent event. Fig.9 illustrates the definition of $F_1$.

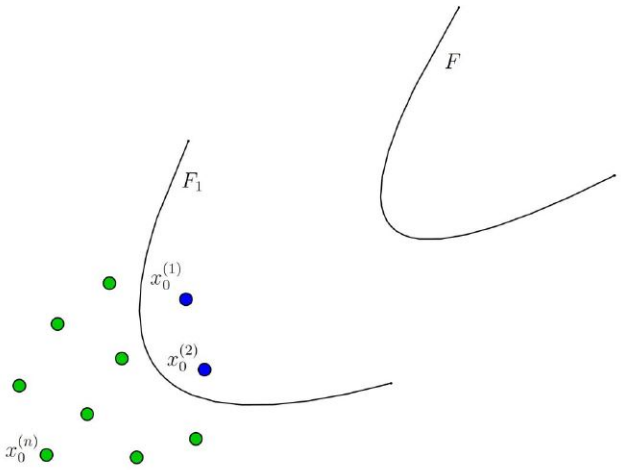



Figure 9: The intermediate failure domain $F_1$. In this schematic illustration ,$n = 10, p = 2$, so that there are exactly $np = 2$ Monte Carlo samples in $F_1$, ${x_0}^{(1)}, {x_0}^{(2)} \in F1$.

The first intermediate failure domain F1 can be viewed as a (very rough) conservative approximation to the target failure domain $F$. Since $F \in F_1$, the failure probability $p_F$ can be written as a product:

$$p_F = \mathbb{P}(F_1)\mathbb{P}(F|F_1) \tag{2.43}$$

where $\mathbb{P}(F|F_1)$ is conditional probability of $F$ given $F_1$. Therefore, in view of 2.42, the problem of estimating $p_F$ is reduced to estimating the conditional probability $\mathbb{P}(F|F_1)$.

In the next stage, instead of generating samples in the whole input space (like in cMC), the SS algorithm aims to populate $F_1$. Specifically, the goal is to generate samples ${x_1}^{(n)}, {x_1}^{(n)}$ from the conditional distribution

$$\pi(x|F_1) = \frac{\pi(x)I_{F_1}}{\mathbb{P}(\mathbb{F}_{\not\vdash})} = \frac{I_{F_1}}{\mathbb{P}(\mathbb{F}_{\not\vdash})}\prod_{k=i}^{d}\phi(x_k) \tag{2.44}$$

First of all, note that samples ${x_0}^{(1)}, ..., {x_0}^{(np)}$ not only belong to $F_1$, but are also distributed according to $\pi(\cdot|F_1)$. To generate the remaining $(n - np)$ samples from $\pi(\cdot|F_1)$, which, in general, is not a trivial task, Subset Simulation uses the so-called Modifieed Metropolis algorithm (MMA). MMA belongs to the class of Markov chain Monte Carlo (MCMC ) algorithms which are techniques for sampling from complex probability distributions that cannot be sampled directly, at least not efficiently. MMA is based on the original Metropolis algorithm and specifically tailored for sampling from the conditional distributions of the form.

#### 2.3.2.1 Modified Metropolis algorithm

Let $x \sim \pi(\cdot|F_1)$ be a sample from the conditional distribution $\pi(\cdot|F_1)$. The Modified Metropolis algorithm generates another sample $\bar{x}$ from $\pi(\cdot|F_1)$ as follows:

1. Generate a 'candidate' sample $\xi$ : For each coordinate $k = 1, ..., d$,

   (a) Sample $\eta_k \sim q_k(\cdot|x_k)$, called the proposal distribution, is univariate PDF for $\eta_k$ centered at $x_k$ with the symmetry property $q_k(x_k|\eta_k)$=$q_k(\eta_k|_k)$. For example, the proposal distribution can be a Gaussian PDF with mean $x_k$ and variance ${\sigma_k}^2$,

   $$q_k(\eta_k|x_k) = \frac{1}{\sqrt{2\pi}\sigma_k}exp(-\frac{{(\eta_k - x_k)}^2}{2{\sigma_k}^2}), \tag{2.45}$$

   or it can be a uniform distribution over $[x_k - a, x_k - a]$, for some $a \geq 0$

   (b) Compute the acceptance ratio

   $$r_k = \frac{\phi(\eta_k)}{\phi(x_k)} \tag{2.46}$$

   (c) Define the $k^{th}$ coordinate of the candidate sample by accepting or rejecting $\eta_k$,

   $$\xi_k = \begin{cases} \eta_k, \text{ with probability min}\{1,\mathrm{r}_k\} \\ \mathrm{x}_k, \text{ with probability 1-min}\{1,\mathrm{r}_k\}. \end{cases} \tag{2.47}$$

2. Accept or reject the candidate sample $\xi$ by setting

$$\bar{x_k} = \begin{cases} \xi, \text{ if } \xi \in F_1 \\ \xi, \text{ if } \xi \notin F_1. \end{cases} \tag{2.48}$$

The Modified Metropolis algorithm is schematically illustrated in Fig.10.

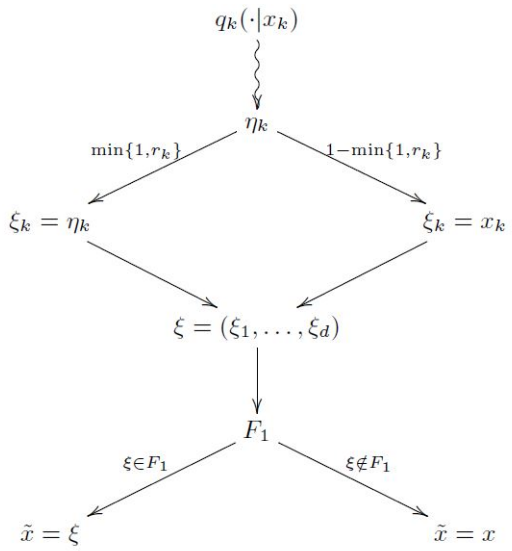



Figure 10: Modified Metropolis algorithm

It can be shown that the sample $\bar{x}$ generated by MMA is indeed distributed according to $\pi(\cdot|F_1)$. If the candidate sample $\xi$ is rejected in 4.16, then $\bar{x} = x \sim \pi(\cdot|F_1)$ and there is nothing to prove. Suppose now that $\xi$ is accepted, $\bar{x} = \xi$, so that the move from $\xi$ to $\bar{x}$ is a proper transition between two distinct points in $F_1$. Let $f(\cdot)$ denote the PDF of $\bar{x}$ (the goal is to show that $f(\bar{x}) = \pi(\cdot|F_1)$. Then

$$f(\bar{x}) = \int_{F_1} \pi(\cdot|F_1) t(\bar{x}|x) dx \ , \tag{2.49}$$

where t(—x) is the transition PDF from $x$ to $\bar{x} \neq x$. According to first step of MMA, coordinates of $\bar{x} = \xi$ are generated independently, and therefore $t(\bar{x}|x)$ can be expressed as a product,

$$t(\bar{x}|x) = \prod_{k=i}^{d} t_k(\bar{x_k}|x_k) \tag{2.50}$$

where $t(\bar{x}|x)$ is the transition PDF for the $k^{th}$ coordinates $\bar{x}_k$. Combining equations (2.44, 2.49, and 2.50) gives

$$
\begin{aligned}
f(\bar{x}) &= \int_{F_1} \frac{I_{F_1}(x)}{\mathbb{P}(F_1)} \prod_{k=i}^{d} \phi(x_k) \prod_{k=i}^{d} t_k(\bar{x_k}|x_k) dx \\
&= \frac{1}{\mathbb{P}(F_1)} \int_{F_1} \prod_{k=i}^{d} \phi(x_k) t_k(\bar{x_k}|x_k) dx
\end{aligned}
\tag{2.51}
$$

The key to the proof of $f(\bar{x}) = \pi(\bar{x}|F_1)$ is to demonstrate that $\phi(x_k)$ and $t_k(\bar{x}|x_k)$ satisfy the so-called detailed balance equation,

$$
\phi(x_k) t_k(\bar{x}|x_k) = \phi(\bar{x}) t_k(x_k|\bar{x}_k). \tag{2.52}
$$

If $\bar{x}_k = x_k$, then (2.52) is trivial. Suppose that $\bar{x}_k \neq x_k$, that is $\bar{x}_k = \xi_k = \eta_k$ in (2.47). The actual transition PDF $t_k(\bar{x}_k|x_k)$ from $x_k$ to $\bar{x_k} \neq x_k$ differs from the projection PDF $q_k(\bar{x_k}|x_k)$ because the acceptance-rejection step (2.47) is involved. To actually make the move from $x_k$ to $\bar{x}_k$, ones needs not only to generate $\bar{x_k} \sim q_k(\cdot|x_k)$, but also to accept it with probability $min\{1, \frac{\phi(\bar{x_k})}{\phi(\bar{x})}\}$. Therefore,

$$
t_k(\bar{x}|x_k) = q_k(\bar{x_k}|x_k) min\{1, \frac{\phi(\bar{x_k})}{\phi(\bar{x})}\}, \qquad \bar{x_k} \neq x_k. \tag{2.53}
$$

Using (2.53), the symmetry property of the proposal PDF, $q_k(\bar{x_k}|x_k) = q_k(x_k|\bar{x_k})$, and the identity $a\ min\{1, \frac{a}{b}\} = b\ min\{1, \frac{a}{b}\}$ for any $a, b > 0$,

$$
\begin{aligned}
\phi(x_k) t_k(\bar{x}|x_k) &= q_k(\bar{x_k}|x_k) \phi(x_k) min\{1, \frac{\phi(\bar{x_k})}{\phi(\bar{x})}\} \\
&= q_k(x_k|\bar{x_k}) \phi(\bar{x_k}) min\{1, \frac{\phi(\bar{x_k})}{\phi(\bar{x})}\} \\
&= \phi(\bar{x_k}) t_k(x_k|\bar{x_k}),
\end{aligned}
\tag{2.54}
$$

and the detailed balance (2.52) is thus established. The rest is a straightfoward calculation:

$$
\begin{aligned}
f(\bar{x}) &= \frac{1}{\mathbb{P}(F_1)} \int_{F_1} \prod_{k=i}^{d} \phi(\bar{x_k}) t_k(x_k|\bar{x_k}) dx \\
&= \frac{1}{\mathbb{P}(F_1)} \prod_{k=i}^{d} \phi(\bar{x_k}) \int_{F_1} t(x|\bar{x_k} dx \ = \pi(\bar{x}|F_1),
\end{aligned}
\tag{2.55}
$$

since the transition PDF $t(x|\bar{x})$ integrates to 1, and $IF_1(\bar{x}) = 1$.

#### 2.3.2.2 Subset Simulation at higher conditional levels

Given $x_0^{(1)}, \cdots, x_0^{(np)} \sim \pi(\cdot|F_1)$, it is clear now how to generate the remaining $(n-np)$ samples from $\pi(\cdot|F_1)$. Namely, starting from each $x_0^{(i)}, i = 1, ..., np$, the SS algorithm generates a sequence of $(1 - \frac{1}{p})$ new MCMC samples $x_0^{(i)} = x_{0,0}^{(i)} \mapsto x_{0,1}^{(i)} \mapsto \ldots \mapsto x_{0,1-\frac{1}{p}}^{(i)}$ using he Modified Metropolis transition rule described above. Note that when $x_{0,j}^{(i)}$ is generated, the previous sample $x_{0,j-1}^{(i)}$ is used as an input for the transition rule. The sequence $x_{0,0}^{(i)}, x_{0,1}^{(i)}, ..., x_{0,1-\frac{1}{p}}^{(i)}$ is called a Markov chain with the stationary distribution $\pi(\cdot|F_1)$, and $x_{0,0}^{(i)} = x_0^{(i)}$ is often referred to as the 'seed' of the Markov chain.

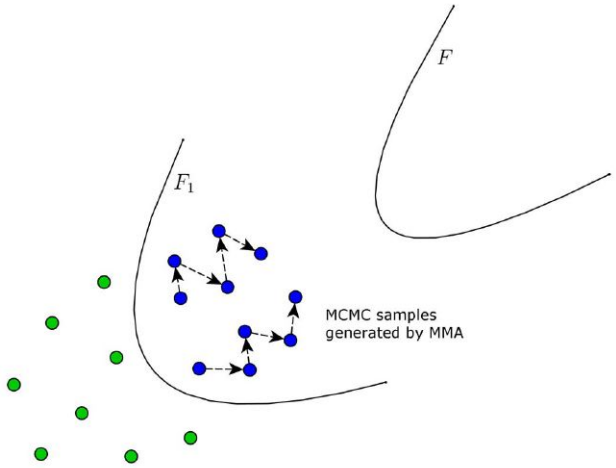



Figure 11: MCMC samples generated by the Modified Metropolis algorithm at the first condition level of Subset Simulation.

To simplify the notation, denote samples $\{x_{0,j}^{(i)}\}_{j=0,\cdots,1-1/p}^{i=1,\cdots,np}$ by $\{x_1^{(1)}, \cdots, x_1^{(n)}\}$. The subscript 1 indicates that the MCMC samples $\{x_1^{(1)}, \cdots, x_1^{(n)}\} \sim \pi(\cdot|F_1)$ are generated at the first conditional level of the SS algorithm. These conditional samples are schematically shown in Fig.11. Also assume that the corresponding system responses $\{y_1^{(1)} = g(x_1^{(1)}), ..., y_1^{(n)} = g(x_1^{(n)})\}$ in the decreasing order, i.e. $y(x_1^{(1)}) \geq ... \geq y(x_1^{(n)})$. If the failure event $F$ is rare enough, that is if $p_F$ is sufficient small, then it is very likely that none of the samples $x_1^{(1)}, ..., x_1^{(n)}$ belongs to $F$, i.e. $y_1^{(i)} < y^*$ for all $i = 1, ..., n$. Nevertheless, these MCMC samples can be used in the similar way the Monte Carlo samples $x_1^{(1)}, ..., x_1^{(n)}$ were used.

By analogy with (2.38), define the second intermediate failure domain $F_2$ as follows:

$$F_2 = \{x : g(x) > {y_2}^*\} \tag{2.56}$$

where

$$y_2{}^* = \frac{y_1{}^{np} + y_1{}^{np+1}}{2} \tag{2.57}$$

Note $y_2{}^* > y_1{}^*$ since $y_1{}^{(i)} > y_1{}^*$ for all $i = 1, ..., n$. This means that $F \subset F_2 \subset F_1$, and therefore $F_2$ can be viewed as a conservative approximation to $F$ which is still rough, yet more accurate than $F_1$. Fig.12. illustrates the definition of $F_2$. By construction, samples $x_1^{(1)}, ..., x_1^{(n)}$ belong to $F_2$, while $x_1^{(np+1)}, ..., x_1^{(n)}$ do not. As a result, the estimate for the conditional probability of $F_2$ given $F_1$ which is based on samples $x_1^{(np+1)}, ..., x_1^{(n)} \sim \pi(\cdot|F_1)$ is automatically equal to p,

$$\mathbb{P}(F_1|F_2) \approx \frac{1}{n}\sum_{i=1}^{n} I_{F_2}(x_0{}^{(i)}) = p. \tag{2.58}$$

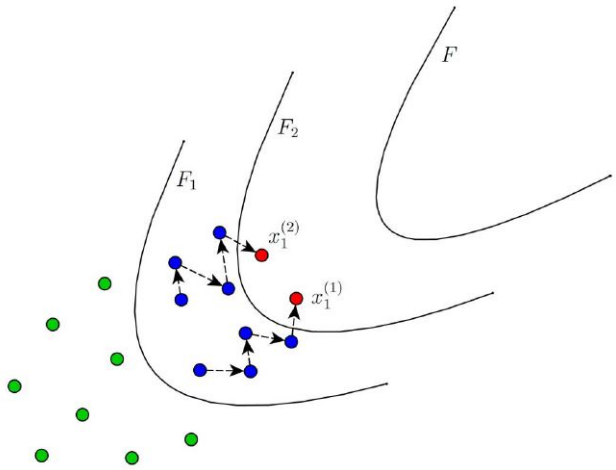



Figure 12: The second intermediate failure domain $F_2$. In this schematic illustration, $n = 10, p = 0.2$, so that there are exactly $np = 2$ MCMC samples in $F_2, x_1^{(1)}, x_1^{(2)} \in F_2$.

Since $F \subset F_2 \subset F_1$, the conditional probability $\mathbb{P}(F|F_1)$ that appears in (2.44) can be expressed as a product:

$$\mathbb{P}(F|F_1) = \mathbb{P}(F_2|F_1)\mathbb{P}(F|F_2) \tag{2.59}$$

Combining (2.44) and (2.59) gives the following expression for the failure probability:

$$p_F = \mathbb{P}(F_1) = \mathbb{P}(F_2|F_1)\mathbb{P}(F|F_2). \tag{2.60}$$

Thus, in view (2.42) and (2.58), the problem of estimating $p_F$ is now reduced to estimating the conditional probability $\mathbb{P}(F|F_2)$.

In the next step, as one may have already guessed, the Subset Simulation algorithm: populates $F_2$ by generating MCMC samples $x_2^{(1)}, ..., x_2^{(n)}$ from $\pi(\cdot|F_2)$ using the Modified Metropolis algorithm; defines the third intermediate failure domain $F_3 \subset F_2$ such that $\mathbb{P}(F_3|F_2) = \mathbb{P}(F_1|F_2) \approx \frac{1}{n}\sum_{i=1}^{n} I_{F_3}(x_2{}^{(i)}) = p$; and reduces the original problem of estimating the failure probability $p_F$ to estimating the conditional probability $\mathbb{P}(F|F_3)$ by representing $p_F = \mathbb{P}(F_1)\mathbb{P}(F_2|F_1)\mathbb{P}(F_3|F_2)\mathbb{P}(F|F_3)$. The algorithm proceeds in this way until the target failure domain $F$ has been sufficiently sampled so that the conditional probability $\mathbb{P}(F|F_L)$ can be accurately estimated by $\frac{1}{n}\sum_{i=1}^{n} I_F(x_L{}^{(i)})$, where $F_L$ is the $L^{th}$ intermediate failure domain, and $x_L^{(1)}, ..., x_L^{(n)} \sim \pi(\cdot|F_L)$ are the MCMC samples generated at the $L^{th}$ conditional level. Subset Simulation can thus be viewed as a method that decomposes the rare failure event $F$ into a sequence of progressively 'less-rare' nested events, $F \subset F_L \subset ... \subset F_1$, where all intermediate failure events $F_1, ..., F_L$ are constructed adaptively by appropriately relaxing the value of the critical threshold $y_1^* <, ..., y_L^* < y^*$.

#### 2.3.2.3 Stopping criterion

In what follows, the stopping criterion for Subset Simulation is described in detail. Let $n_F(l)$ denote the number of failure samples at the $l^{th}$ level, that is

$$n_F(l) = \sum_{i=1}^{n} I_F(x_l{}^{(i)}), \tag{2.61}$$

where $x_l^{(1)}, ..., x_l^{(n)}$ from $\pi(\cdot|F_l)$. Since $F$ is a rare event, it is very likely that $n_F(l) = 0$ for the first few conditional levels. As $l$ gets larger, however, $n_F(l)$ starts increasing since $F_l$, which approximates $F$ 'from above', shrinks closer to $F$. In general, $n_F(l) \geq n_F(l-1)$, since $F \subset F_l \subset F_{l-1}$ and the $np$ closest to $F$ samples among $x_{l-1}^{(1)}, ..., x_{l-1}^{(n)}$ are present among $x_1^{(1)}, ..., x_l^{(n)}$. At conditional level $l$, the failure probability $p_F$ is expressed as a product,

$$p_F = \mathbb{P}(F_1)\mathbb{P}(F_2|F_1)...\mathbb{P}(F_l|F_{l-1})\mathbb{P}(F|F_l) \tag{2.62}$$

Furthermore, the adaptive choice of intermediate critical thresholds $y_1^*, ..., y_l^*$ guarantees that the first $l$ factors in (2.62) approximately equal to p, and, thus,

$$p_F \approx p^l \cdot \mathbb{P}(F|F_l) \tag{2.63}$$

Since there are exactly $n_F(l)$ failure samples at the lth level, the estimate of the last conditional probability in (2.62) which is based on samples $x_l^{(1)}, ..., x_l^{(n)}$ from $\pi(\cdot|F_l)$ is given by

$$\mathbb{P}(F|F_L) = \frac{1}{n}\sum_{i=1}^{n} I_F({x_L}^{(i)}) = \frac{n_F(l)}{n}. \tag{2.64}$$

If $n_F(l)$ is sufficiently large, i.e. the conditional event $(F|F_l)$ is not rare, then estimate (2.64) is fairly accurate. This leads to the following stopping criterion:

- If $n_F(l)/n \geq p$, i.e. there are at least $np$ failure samples among $x_l^{(1)}, ..., x_l^{(n)}$, then Subset Simulation stops: the current conditional level l becomes the last level, $L = 1$, and the failure probability estimate derived form (2.63) and (2.64) is

$$p_F \approx \widehat{p_F^{SS}} = p^L \frac{n_F(L)}{n}. \tag{2.65}$$

- If $n_F(l)/n < p$, i.e. there are less than np failure samples among $x_l^{(1)}, ..., x_l^{(n)}$, then algorithm proceeds by defining the next intermediate failure domain $F_{l+1} = \{x : g(x) > {y_{l+1}}^*\}$, where ${y_{l+1}}^* = \frac{{y_l}^{(np)} + {y_l}^{np+1}}{2}$, and expressing $\mathbb{P}(F|F_l)$ as a product $\mathbb{P}(F|F_l) = \mathbb{P}(F_{l+1}|F_l)\mathbb{P}(F|F_{l+1}) \approx p\mathbb{P}(F|F_{l+1})$.

The described stopping criterion guarantees that the estimated values of all factors in the factorization $p_F = \mathbb{P}(F_1)\mathbb{P}(F_2|F_1)...\mathbb{P}(F_L|F_{L-1})\mathbb{P}(F|F_l)$ are not smaller than p. If p is relatively large ($p = 0.1$ is often used in applications), then it is likely that the estimates $\mathbb{P}(F_1) \approx p$, $\mathbb{P}(F_2|F_1) \approx p$,...,$\mathbb{P}(F_L|F_{L-1}) \approx p$ and $\mathbb{P}(F_L) \approx \frac{n_F(L)}{n} (\geq p)$ are accurate even when the sample size $n$ is relatively small. As a result, the SS estimate (2.65) is also accurate in this case. This provides an intuitive explanation as to why Subset Simulation is efficient in estimating small probabilities of rare events.

#### 2.3.2.4 Implementation Details

1.**Level probability**

The parameter $p$, called the level probability and the conditional failure probability governs how many intermediate failure domains $F_l$ are needed to reach the target failure domain $F$. As it follows form (2.65), a small value of $p$ leads to a fewer total number of conditional levels $L$. But at the same time, it results in a large number of samples n needed at each conditional level $l$ for accurate determination of $F_l$ (i.e. determination of $y_l^*$) that satisfies $\frac{1}{n}\sum_{i=1}^{n} I_F({x_{l-1}}^{(i)}) = p$. In the extreme case when $p \leq p_F$, no levels are needed, $L = 0$, and Subset Simulation reduces to the crude Monte Carlo method. On the other, increasing the value of $p$ will mean that fewer samples are

needed at each conditional level, but it will increase the total number of levels $L$. The choice of the level probability p is thus a trade off between the total number of level $L$ and the number of samples n at each level. In [3], it has been found that the value $p = 0.1$ yields good efficiency.

2.**Proposal distributions**

The efficiency and accuracy of Subset Simulation also depends on the set of univariate proposal PDFs $\{g_k\}$, $k = 1, ..., d$ that are used within the Modified Metropolis algorithm for sampling from the conditional distributions $\pi(\cdot|F_l)$. To see this, note that in contrast to the Monte Carlo samples $x_0^{(1)}, ..., x_0^{(n)} \sim \pi(\cdot)$ which are i.i.d, the MCMC samples $x_0^{(1)}, ..., x_0^{(n)} \sim \pi(\cdot|F_l)$ are not independent for $l \geq 1$, since the MMA transition rule uses $x_l^{(i)} \sim \pi(\cdot|F_l)$ to generate $x_l^{(i+1)} \sim \pi(\cdot|F_l)$. This means that although these MCMC samples can be used for statistical averaging is reduced if compared with the i.i.d case. Namely, the morre correlated $x_0^{(1)}, ..., x_0^{(n)}$ are, the slower is the convergence of the estimate $P(F_{l+1}|F_l) \approx \frac{1}{n}\sum_{i=1}^{n} I_{F_{l+1}}(x_l{}^{(i)})$, and, therefore the less efficient it is. The correlation between samples $x_0^{(1)}, ..., x_0^{(n)}$ is due to proposal PDFs $\{q_k\}$, which govern the generation of the next sample $x_l^{(i+1)}$ from the current one $x_l^{(i+1)}$. Hence, the choice of the $\{q_k\}$ is very important.

It was observed in [4] that the efficiency of MMA is not sensitive to the type of the proposal PDFs (Gaussian, uniform, etc), however, it strongly depends on their spread (variance). Both small and large spreads tend to increase the correlation between successive samples. Large spreads may reduce the acceptance rate in (4.16), increasing the number of repeated MCMC samples. Small spreads, on the contrary, may lead to a reasonably high acceptance rate, but still produce very correlated samples due to their close proximity. As a rule of thumb, the spread of $q_k$, $k = 1, ..., d$, can be taken of the same order as the spread of the corresponding marginal PDF $\{\pi_k\}$. Proposal PDFs can also be Gaussian with unit variance, as this choice is found to give a balance between efficiency and robustness [4].

The spread of proposal PDFs can also be chosen adaptively. The nearly optimal scaling strategy for the Modified Metropolis algorithm was adopted: at each conditional level, select the spread such that the the corresponding acceptance rate in (4.16) is between 30% and 50%. In general, finding the optimal spread of proposal distributions is problem specific and a highly non-trivial task not only for MMA, but also for almost all MCMC algorithms.

#### 2.3.2.5 Example-SS

Assume the same problem as in 2.2.3.1, but now we calculate the probability of failure using Subset Simulation. The number of simulations is 1000 per conditional level. The probability of failure is $p_F = 9.3 \times 10^{-3}$.

#### 2.3.2.6 MATLAB Script -SS

A MATLAB script (SubsetSim), which solve example 2.3.2.5 using SS simulation, exists in the appendix, while it is based on [4].

```
clear; clc; close all;
1. m_fy=300*10^-3; m_As=0.0026;
2. m_fc=20*10^-3; m_Med=230;
3. s_fy=31.5*10^-3; s_As=0.000052;
4. s_fc=2.8*10^-3; s_Med=27.6;
5. Properties=[m_fy s_fy 1
6.             m_As s_As 2
7.             m_fc s_fc 3
8.             m_Med s_Med 4];
9.% Create distribution objects
10.fy_dist1 = makedist('Normal', 'mu', m_fy,'sigma', s_fy);
11.As_dist1 = makedist('Normal', 'mu',m_As,'sigma', s_As);
12.fc_dist1 = makedist('Normal', 'mu', m_fc,'sigma', s_fc);
13.Q_dist1 = makedist('Nomral', 'mu', m_Q,'sigma', s_Q);
14.
15.dist = [fy_dist1 , As_dist1 , fc_dist1 , Q_dist1];
16.N_Simulations_Level=1000;
17.SubsetSim(Problem_Properties,dist,N_Simulations_Level);
```

# 3. Structural Reliability Applications

## 3.1 Introduction

In order to assess the performances of the reliability methods, three test examples are considered, which are based on structural analysis problems.

## 3.2 Example 1: Continuous beam-3 random variables

Reliability analysis of three-span continuous beam.

Consider the reliability analysis of three-span beam with L = 5 m as shown in Fig.13, the performance function is defined as the maximal deflection of three-span beam not exceeding L/360.

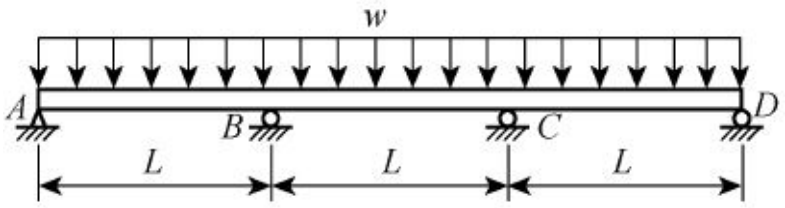



Figure 13: Schematic of three-span continuous beam

The limit state is shown as

$$g(w, E, I) = L/360 - 0.0069wL^4/EI, \tag{3.1}$$

where $w$ denotes distributed loads, $E$ is the modulus of elasticity and $I$ is the moment of inertia. The basic variables are distributed normally, and their distribution parameters are given in Table.2.

Table 2: Distribution parameters of basic variables

| Random variable | Mean | Standard deviation |
|---|---|---|
| w | 10kN/m | 0.4kN/m |
| E | $2\times10^{7}kN/m^{2}$ | $0.5\times10^{7}kN/m^{2}$ |
| I | $8\times10^{-4}kN/m^{4}$ | $1.5\times10^{-4}kN/m^{-4}$ |

As it is shown in Fig.14, the **reliability index,** $\beta$, which is obtained by Hasofer Lind method, is calculated 3.18 after three iterations and the probability of failure is shown in the Table.3, while the precision of the method is $10^{-3}$. The limit state equation is simple, thus the method converges quickly. The start point of the method is chosen to be the **design point** (mean values), which is close to the final point.

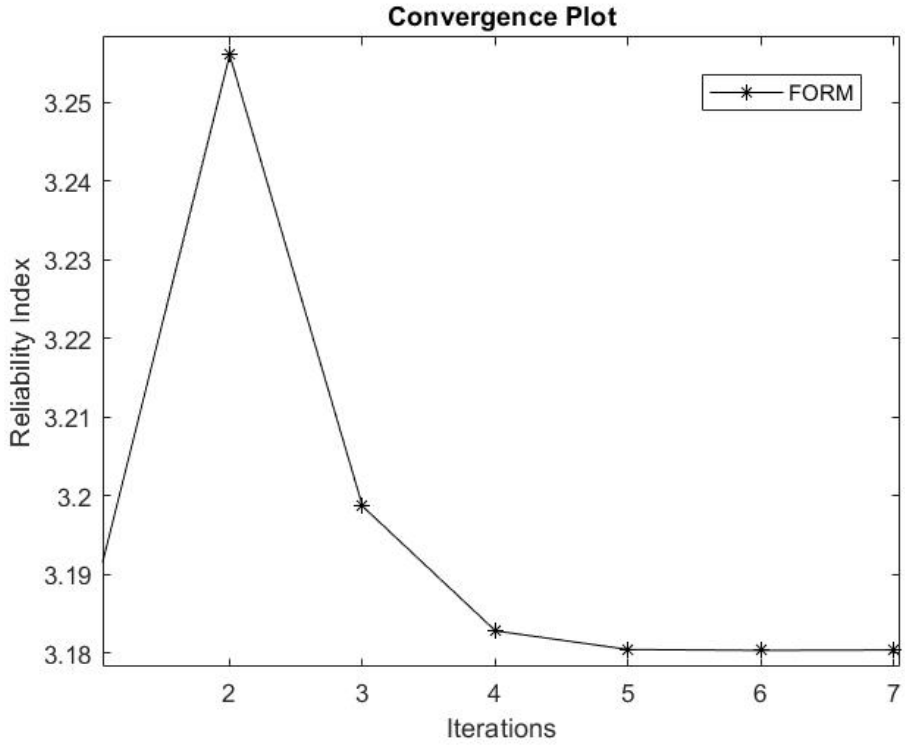



Figure 14: Iterative processes of reliability index for Example 1.

Next, using the MC method with 1.000.000. simulations the probability is equal to $8.57 \times 10^{-4}$, similar to FORM. In Table.3 are presented the probability of failures, using the two methods.

Table 3: Failure probabilities ($\times10^{-4}$)

| FORM | MCS |
|---|---|
| 7.35 | 8.57 |

## 3.3 Example 2: Truss-13 random variables

The second test example considered in this thesis is the evaluation of the failure probability of a 23-bar truss, which is shown in the Fig.15. The total length is 24m, while there are 6 vertical loads at the top chord. The material properties of top and bottom chord are $E_1, A_2$ and $E_2, A_2$, respectively. The diagonals have $E_3$ and $A_3$. Each bar has modeled with truss element, and the displacements have been calculated through linear static analysis. The vertical deformation of the center node is restricted to L/320, where L is the length of the bridge.

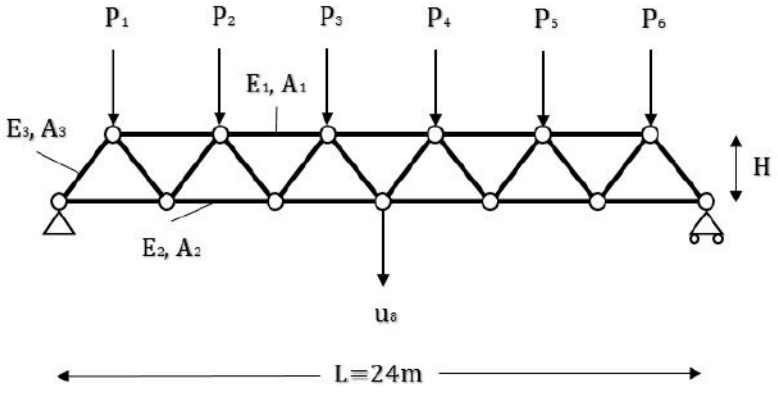



Figure 15: 2D Truss example: geometry and loads

The performance function is:

$$g(x) = L/320 - u_{center}(x), \tag{3.2}$$

where $x$ is the vector with the random variables. The truss has modeled with 13 random variables with properties as they are shown in the Table 4. Firstly, all random variables were assumed to follow the Gaussian distribution. Moreover, it is considered that material and geometry properties follow lognormal distribution, while loads follow extreme value. The **failure probability**, $p_f$ has been calculated with FORM, MC, and Subset simulation.

Table 4: Parameters of variables in Example 2.

| Random variable | Mean | C.O.V. | $\sigma$ |
|---|---|---|---|
| $E_1, E_2, E_3$ | 210 GPa | 10% | 21 GPa |
| $A_1, A_2$ | $20\times10^{-4}m^2$ | 5% | $1\times10^{-4}m^2$ |
| $A_3$ | $10\times10^{-4}m^2$ | 5% | $0.5\times10^{-4}m^2$ |
| $P_1 \sim P_6$ | 50 kN | 15% | 7.5 kN |
| H | 2 m | 5% | 0.1 m |

1.**FORM**. Calculation the probability of failure for the truss with 13 normal and non-normal random variables, using FORM-HL and FORM-HLRF.

For this problem, in which the limit state function is not explicit, the partial derivatives are calculated numerically, using the Central Difference Method (Fig.16).

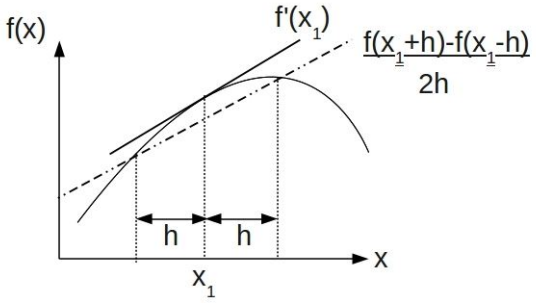



Figure 16: A graphical representation of a central-difference approximation to the gradient at a point $x_1$. The approximation gives way to the true value of $f^{'}(x_1)$ as the distance $h$ shrinks to become infinitesimally small.

In Fig.17 is shown the convergence plot, between the reliability index $\beta$ and the number of iteration. The results obtained by FORM-HL and FORM–HLRF methods show a periodic oscillation, while the results is similar for the two methods.

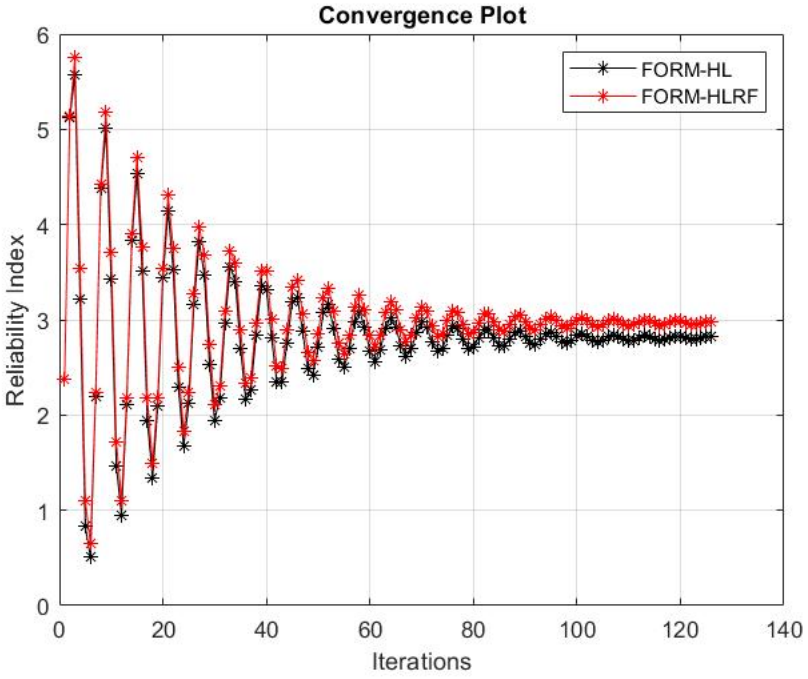



Figure 17: Iterative processes of reliability index for Example 2.

The **Importance Factors** for every random variable have also defined, while it is obvious from the Fig.18 and Fig.19 that $H$ and $E_1$ mainly affect the probability.

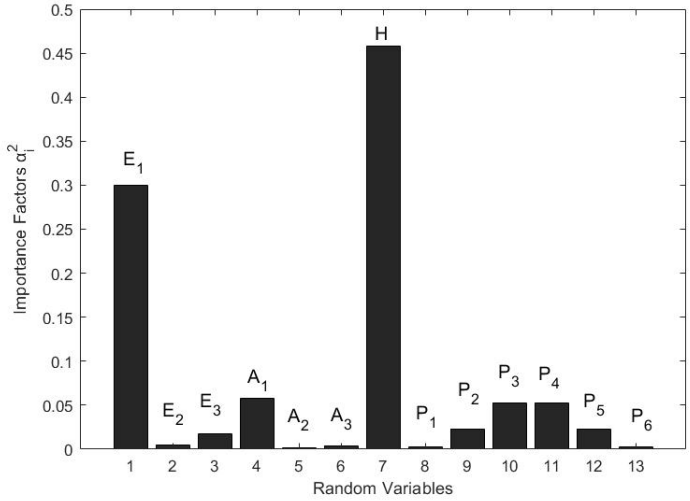



Figure 18: Importance Factors for normal random variables

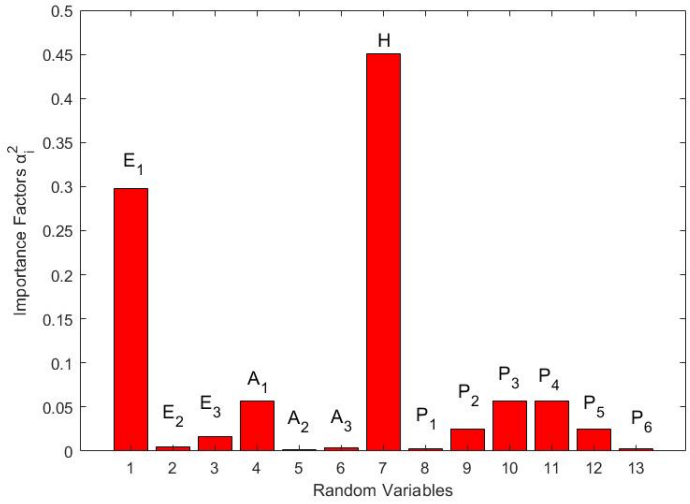



Figure 19: Importance Factors for non-normal random variables

2.**MC**.Calculation the probability of failure for the Truss application with 13 normal and non-normal random variables, using MC.

In Fig.20 and Fig.21 are presented the histograms of the failure function for 100.000. samples, using Monte Carlo. The first histogram is for **normal** variables, while the second for **non-normal**. Moreover, at each histogram there are two distribution fits, a normal fit and a non-parametric fit. When these two fits are the same the central limit theorem is valid. This is not observed in the histogram, which means that the number of data should be increased, for the validation of central limit theorem. However, this number (100.000) of data is sufficient for the calculation of the probability of failure, using Monte Carlo. The coefficient of variation for $p_F = 10^{-3}$ is 10%.

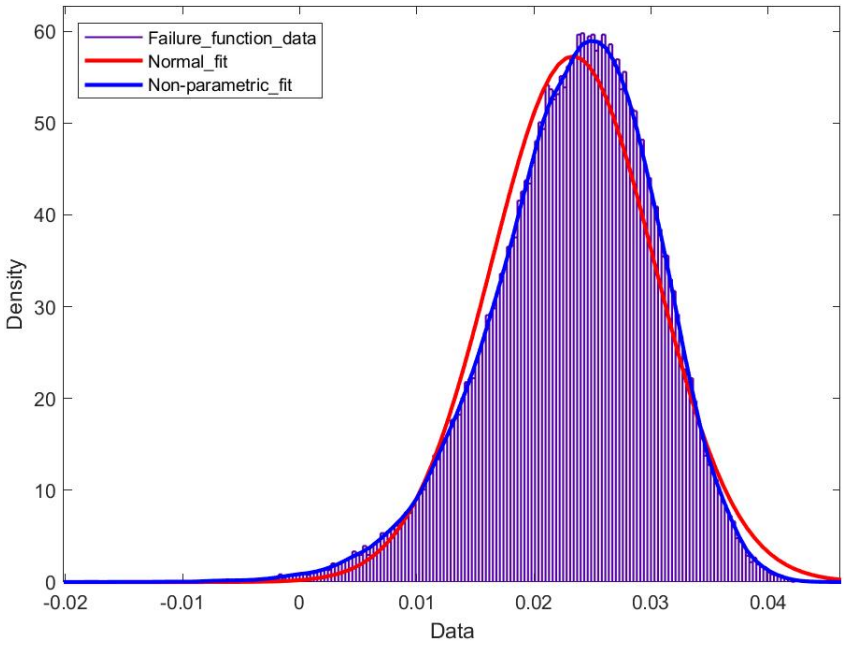



Figure 20: Histogram of failure function for 13 normal random variables

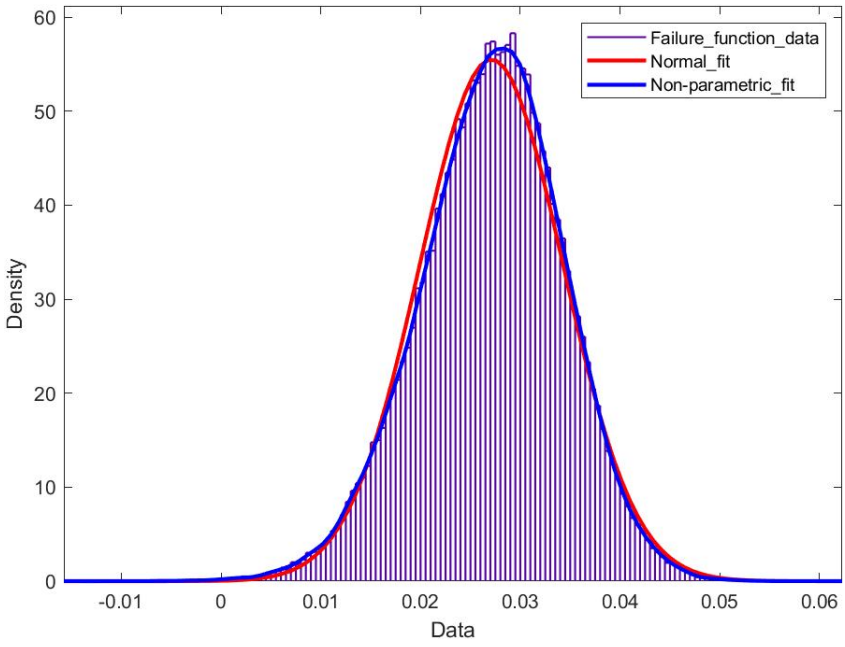



Figure 21: Histogram of failure function for 13 non-normal random variables

3.**Subset Simulation**. The condition failure probability, which define the number of the intermediate levels, is chosen to be 0.1. Three levels are arised. The proposal PDF is Gaussian with unit variance. This choice balances between efficiency and robustness for Modified Metropolis Algorithm [4].

In Fig.22 are depicted the estimated probability of failures from three independent Subset simulations for different limits of vertical displacement of the truss. The total number of samples are 3000 for each simulation. Moreover, the results of a basic Monte Carlo simulation with 100.000 (the coefficient of variation for $p_F = 10^{-3}$ is 10%) samples are illustrated. It is obvious that results from the Subset simulation and Monte Carlo do not diverge significantly.

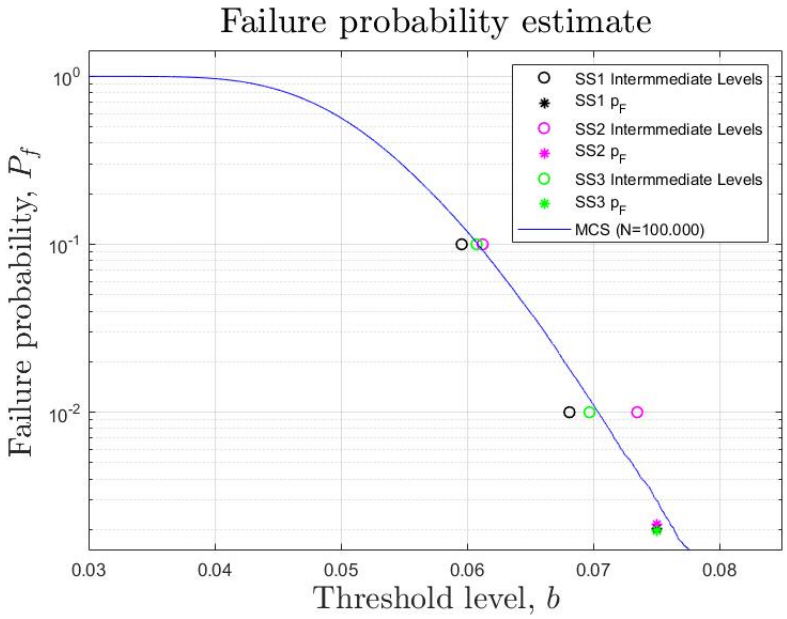



Figure 22: Failure probability and intermediate conditional levels from 3 independent simulations.
Blue line: Empirical distribution function for the Monte Carlo ( N = 100.000) sampling distributions.

The bias of the estimated probability of failures are useful to be examined. For this purpose, 50 independent Subset simulations are executed and the sample's average has been defined. The results are illustrate in Fig.23. It is noticed that the sample's average of the estimated probability of failure is almost the same with the results of Monte Carlo, thus the bias due to the correlation of the conditional probabilities at the intermediate simulation levels is inconsiderable.

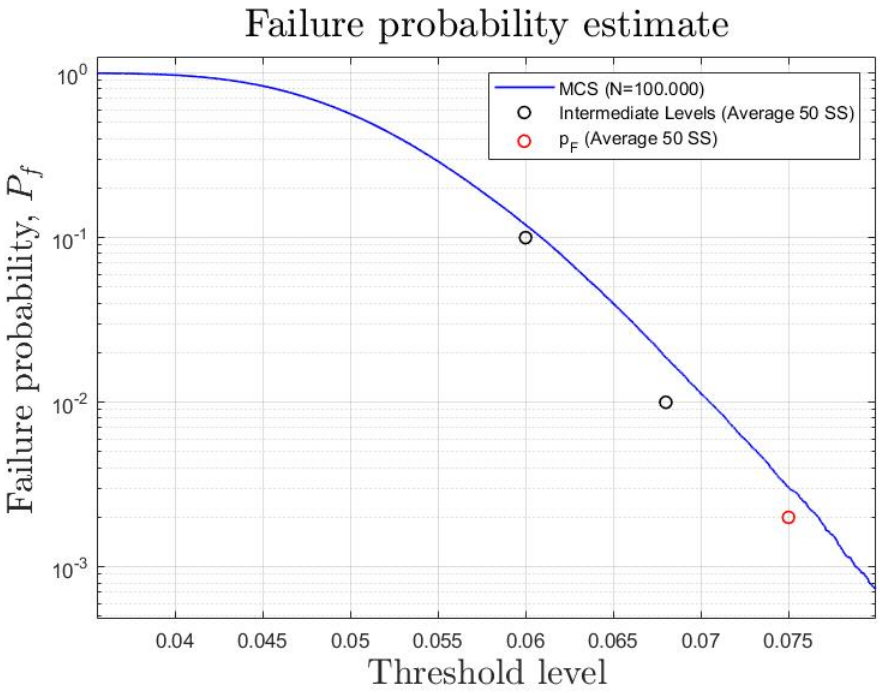
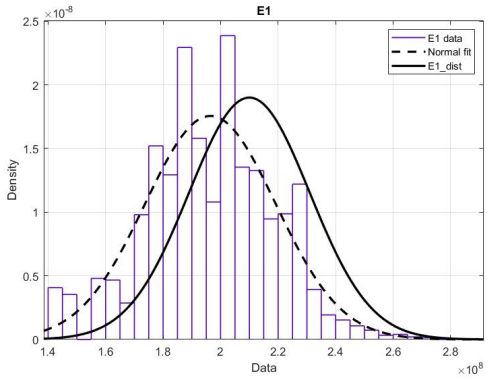


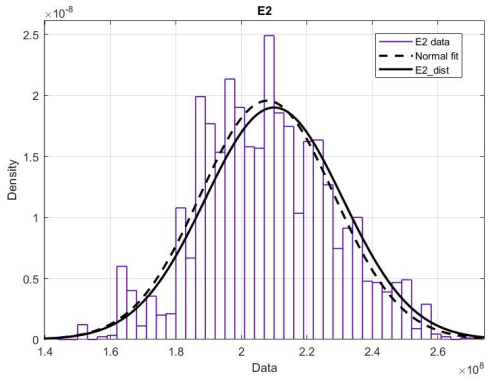

Figure 23: Sample average of failure probability estimates from 50 simulation runs for Example 2.

Typical histograms of conditional samples of uncertainty parameters from a typical simulation, are plotted in Fig. 24,25. The distribution of the conditional samples is used to determine the sensitivity of the model to each individual uncertain parameter. As shown in Fig. 24a, 25a, the distribution of the conditional sample for a typical parameter with high sensitivity, experiences a significant deviation from the unconditional distribution (i.e., the predefined PDF, plotted as the black line). This indicates that, at higher exceedance probabilities, the parameter was skewed in one direction from its unconditional distribution and hence contributed to the exceedance. On the other hand, the histogram of an insensitive parameter (Fig. 24b, 25b) at high exceedance probability levels exhibits insignificant deviation from the unconditional one; that is, there is no significant relation between the distribution of the parameter and the exceedance rate.

The obtained probabilities from the three methods are represented in the Table.10. It is obvious that the results do not diverge significantly. However, the computational time for each method differs. Monte carlo simulation demands the largest time, while Subset simulation minimizes the computational cost. Generally, Subset simulation is a powerful tool and more efficient than FORM and MC that becomes more obvious in the next example, in which the probability arises lower and the computational time is increased.

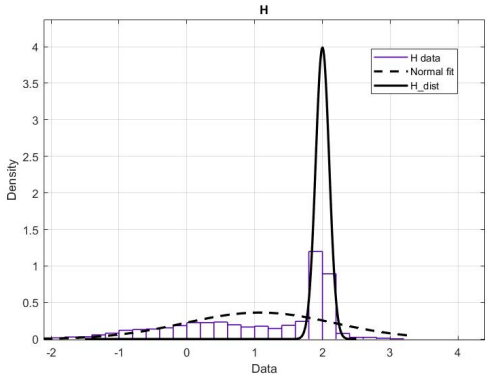



(a) Histogram of E1,
Nomral fit: normal distribution
for histogram's datas,
E1_dist: unconditional distribution of E1

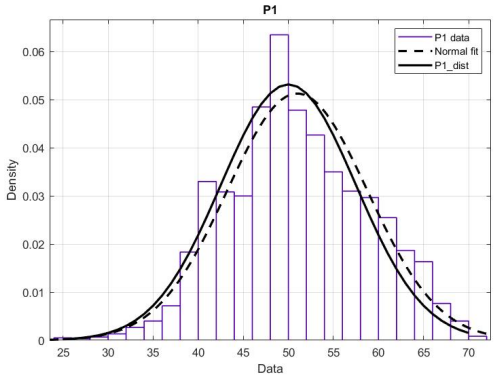



(b) Histogram of E2,
Nomral fit: normal distribution
for histogram's datas,
E2_dist: unconditional distribution of E2

Figure 24: Histograms of conditional samples.




(a) Histogram of H,
Nomral fit: normal distribution
for histogram's datas,
H_dist: unconditional distribution of H




(b) Histogram of P,
Nomral fit: normal distribution
for histogram's datas,
P_dist: unconditional distribution of P

Figure 25: Histograms of conditional samples.

Table 5: Probability of failure calculating with three methods

| Method | $p_f$ | time(s) |
|---|---|---|
| FORM | 2.3*$10^{-3}$ | - |
| MC | 2.2*$10^{-3}$ | 69.52 |
| SS | 2.02*$10^{-3}$ | 2.23 |

## 3.4 Example 3: Frame-11 random variables

The third test example considered in this thesis is the evaluation of the failure probability of a two-dimensional frame, which is shown in the Fig.48. The frame has 3 members, the length of the vertical and horizontal members is 6m. In joint 2 there is a lateral force and counterclockwise moment. The frame has modeled with three beam elements, while the displacements have been calculated with linear static analysis. The horizontal deformation of node 2 is restricted to 1mm.

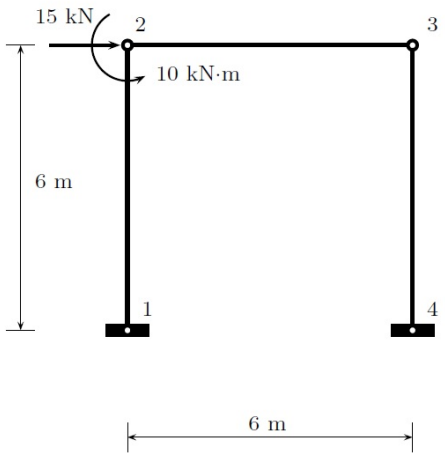



Figure 26: 2D frame example:geometry and loads.

The performance function is defined as

$$g(x) = 0.001 - u_2^{hor}(x). \tag{3.3}$$

where $x$ is the vector with the random variables. The frame has modeled with 11 random variables with properties as they are shown in the Table.6. Firstly, all random variables were assumed to follow the Gaussian distribution. Moreover, for FORM and MC, it is considered that material and geometry properties follow lognormal distribution, while loads follow extreme value. The **failure probability**, $p_f$ has been calculated with FORM, MC, and Subset simulation.

Table 6: Parameters of variables in Example 2.

| Random variable | Mean | C.O.V. | $\sigma$ |
|---|---|---|---|
| $E_1, E_2, E_3$ | 210 GPa | 10% | 21 GPa |
| $A_1, A_2, A_3$ | $200\times10^{-6}m^2$ | 5% | $10\times10^{-6}m^2$ |
| $I_1, I_2, I_3$ | $2\times10^{-4}m^4$ | 5% | $1\times10^{-4}m^4$ |
| P | 15 kN | 15% | 2.25 kN |
| M | 10 kNm | 15% | 1.5 KNm |

1.**FORM**. Calculation the probability of failure for the truss with 11 normal and non-normal random variables, using FORM-HL and FORM-HLRF. For this problem, in which the limit state function is not explicit, the partial derivatives are calculated numerically, using the Central Difference Method (Fig.27).

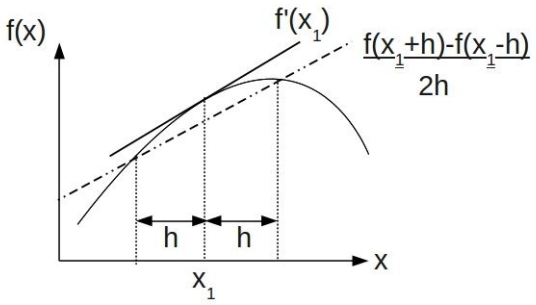



Figure 27: A graphical representation of a central-difference approximation to the gradient at a point $x_1$. The approximation gives way to the true value of $f^{'}(x_1)$ as the distance $h$ shrinks to become infinitesimally small.

In Fig.28 is shown the convergence plot, between the reliability index $\beta$ and the number of iteration. The results obtained by FORM-HL and FORM–HLRF method show a periodic oscillation, while the results is similar for the two methods.

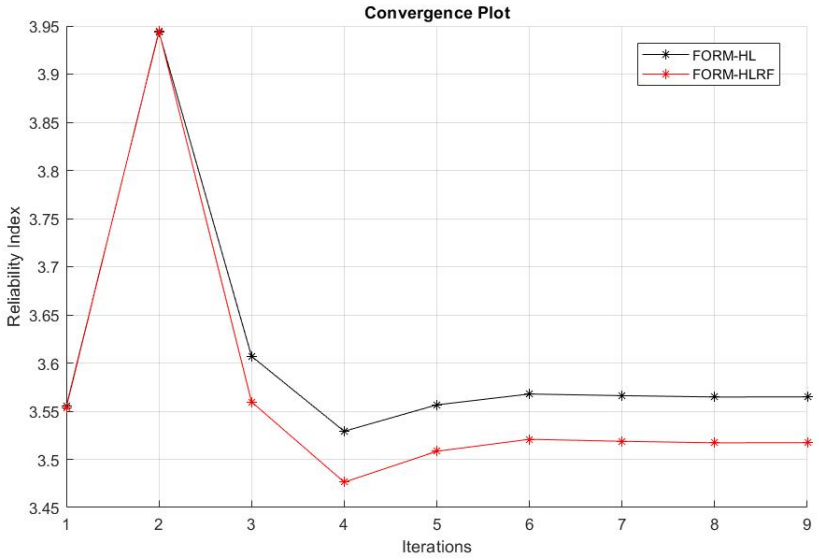



Figure 28: Iterative processes of reliability index for Example 3

The **Importance Factors** for every random variable have also defined, while it is obvious from the Fig.29 and Fig.30 that $I_1$ and $P$ mainly affect the probability.

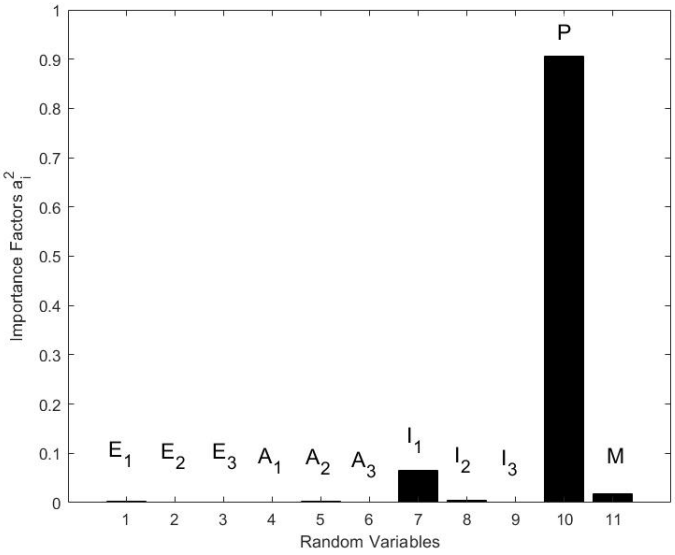



Figure 29: Sensitivity measures, 11 normal variables

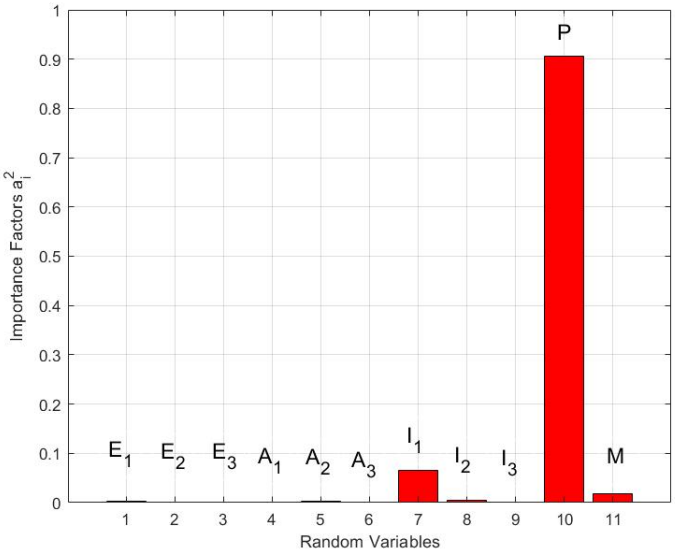



Figure 30: Sensitivity measures, 11 non-normal variables

2**MC**.Calculation the probability of failure for the frame with 13 normal and non-normal random variables, using MC.

In Fig.31 and Fig.32 are presented the histograms of the failure function for 1.000.000. samples, using Monte Carlo. The first histogram is for **normal** variables, while the second for **non-normal**. Moreover, at each histogram there are two distribution fits, a normal fit and a non-parametric fit. When these two fits are the same the central limit theorem is valid. The two fits are identify, thus the central limit theorem is valid.

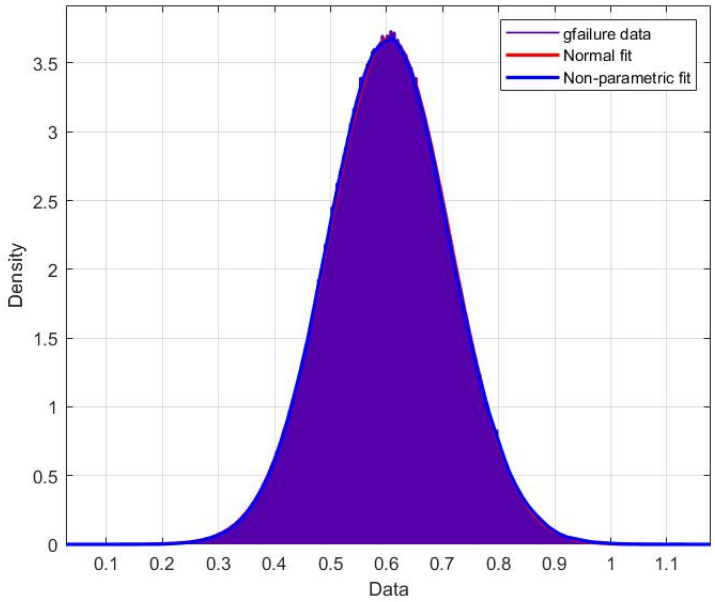



Figure 31: Histogram of failure function for 11 normal random variables

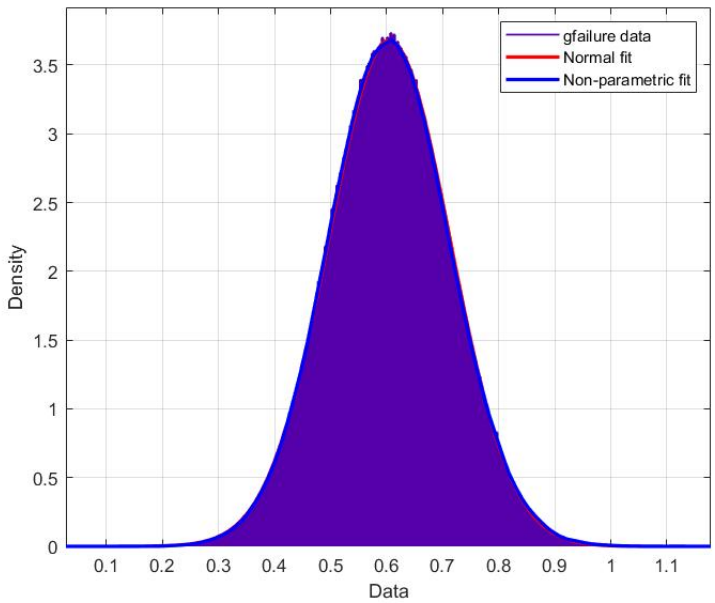



Figure 32: Histogram of failure function for 11 non-normal random variables

3.**Subset Simulation**. The condition failure probability, which define the number of the intermediate levels, is chosen to be 0.1. Three levels are arised. The proposal PDF is Gaussian with unit variance. This choice balances between efficiency and robustness for Modified Metropolis Algorithm [4].

In Fig.33 are depicted the estimated probability of failures from three independent Subset simulations for different limits of vertical displacement of the truss. The total number of samples are 15000 for each simulation. Moreover, the results of a basic Monte Carlo simulation with 1.000.000 (the coefficient of variation for $p_F = 10^{-4}$ is 10%) samples are illustrated. It is obvious that results from the Subset simulation and Monte Carlo do not diverge significantly.

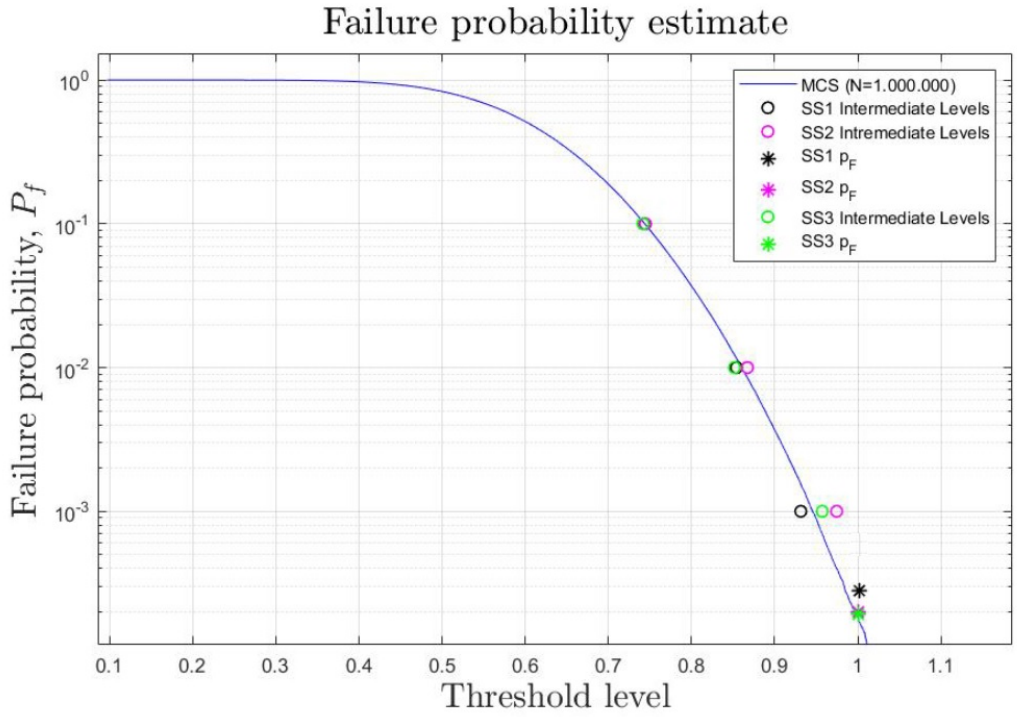



Figure 33: Failure probability and intermediate conditional levels from 3 independent simulations.
Blue line: Empirical distribution function for the Monte Carlo ( N = 1.000.000) sampling distributions.

The bias of the estimated probability of failures are useful to be examined. For this purpose, 50 independent Subset simulations are executed and the sample's average has been defined. The results are illustrate in Fig.34. It is noticed that the sample's average of the estimated probability of failure is almost the same with the results of Monte Carlo, thus the bias due to the correlation of the conditional probabilities at the intermediate simulation levels is inconsiderable.

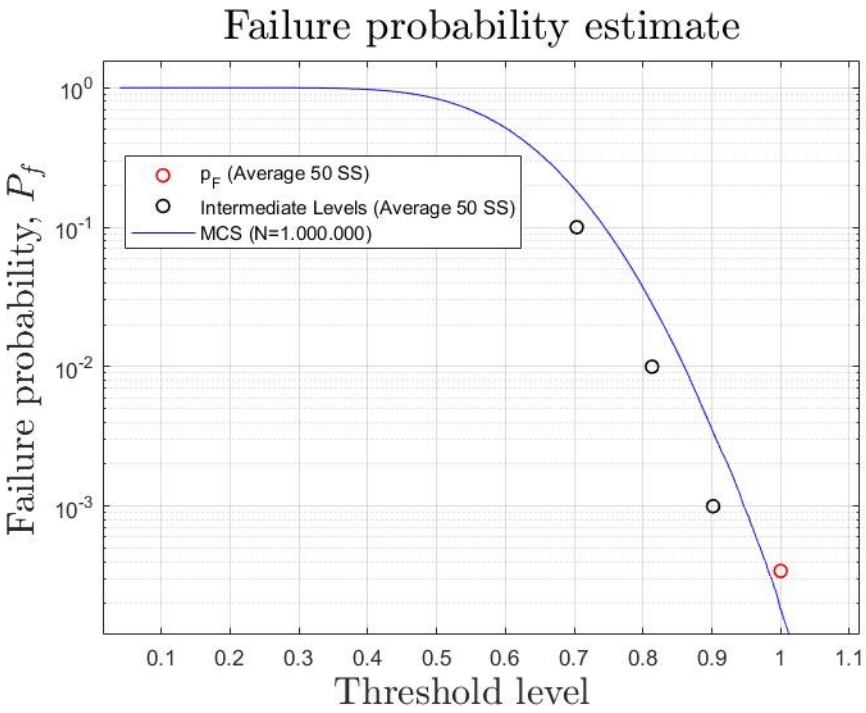
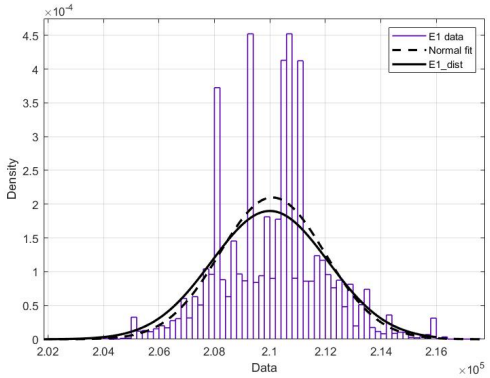


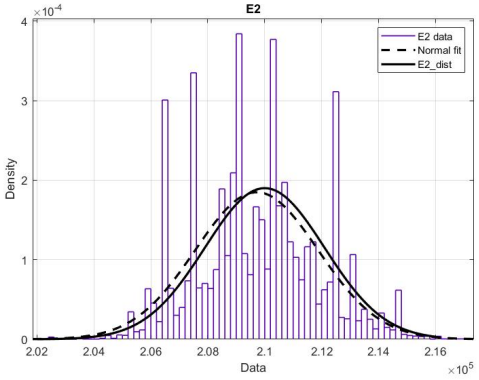

Figure 34: Sample average of failure probability estimates from 50 simulation runs for Example 3.

Typical histograms of conditional samples of uncertainty parameters from a typical simulation, are plotted in Fig. 35,36. The distribution of the conditional samples is used to determine the sensitivity of the model to each individual uncertain parameter. As shown in Fig. 36a, 36b, the distribution of the conditional sample for a typical parameter with high sensitivity, experiences a significant deviation from the unconditional distribution (i.e., the predefined PDF, plotted as the black line). This indicates that, at higher exceedance probabilities, the parameter was skewed in one direction from its unconditional distribution and hence contributed to the exceedance. On the other hand, the histogram of an insensitive parameter (Fig. 35a, 35b) at high exceedance probability levels exhibits insignificant deviation from the unconditional one; that is, there is no significant relation between the distribution of the parameter and the exceedance rate.

The obtained probabilities from the three methods are represented in the table.10. It is obvious that the results do not diverge significantly. However, the computational time for each method differs. Monte carlo simulation is a time-consuming method, as the computational cost is increased, as the probability arises lower and the FEM model more complex. Subset simulation is a efficient method, which minimize computational time and provide accurate results.

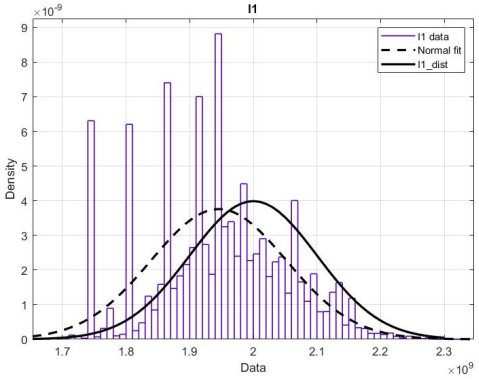



(a) Histogram of E1,
Nomral fit: normal distribution
for histogram's datas,
E1_dist: unconditional distribution of E1

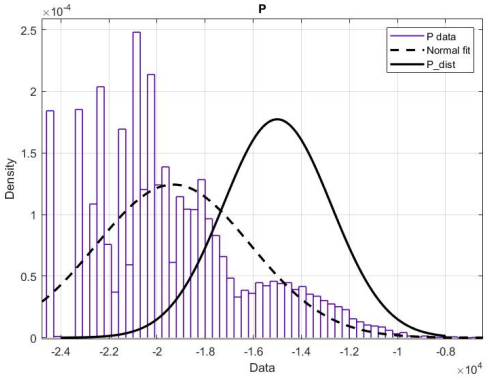



(b) Histogram of E2,
Nomral fit: normal distribution
for histogram's datas,
E2_dist: unconditional distribution of E2

Figure 35: Histograms of conditional samples.




(a) Histogram of I1,
Nomral fit: normal distribution
for histogram's datas,
I1_dist: unconditional distribution of I1




(b) Histogram of P,
Nomral fit: normal distribution
for histogram's datas,
P_dist: unconditional distribution of P

Figure 36: Histograms of conditional samples.

Table 7: Probability of failure calculating with three methods

| Method | $p_f$ | time(s) |
|---|---|---|
| FORM | $1.9*10^{-4}$ | - |
| MC | $2*10^{-4}$ | 845.38 |
| SS | $2.20*10^{-4}$ | 34.89 |

# 4. Machine Learning Algorithms

## 4.1 Introduction

Structural Reliability analysis is one of the prominent fields in civil and mechanical engineering. However, an accurate analysis in most cases deals with complex and costly numerical problems. Machine learning-based techniques have been introduced to the structural reliability analysis problems to deal with this huge computational cost and increase accuracy. Aiming towards a fast and accurate analysis, the machine learning techniques adopted for the approximation of the limit state function with Monte Carlo simulation. In this regard, the focus of the current chapter is to examine two Machine Learning models in combination with structural reliability analysis.

Machine learning is the scientific study of algorithms and statistical models that computer systems use to effectively perform a specific task without using explicit instructions, relying on models and inference instead. It is seen as a subset of artificial intelligence. These algorithms are used to automatically find the valuable underlying patterns within complex data that we would otherwise struggle to discover. The hidden patterns and knowledge about a problem can be used to predict future events. Machine learning algorithms build a mathematical model of sample data (Fig.37), known as training data, in order to make predictions or decisions without being explicitly programmed to perform the task.

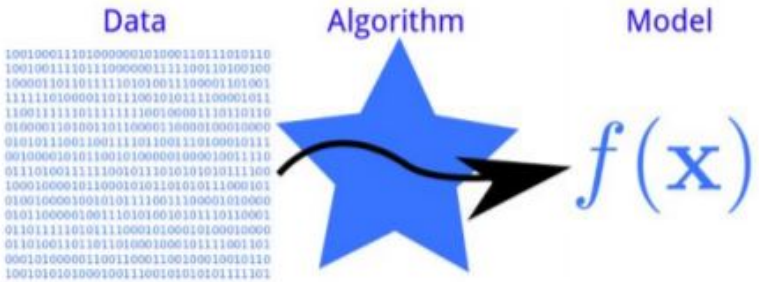



Figure 37: Schematic representation of the general idea, with which a machine learning algorithm is used the training data to define a surrogate model.

The reliability analysis of implicit performance function is one of challenges in structural reliability discipline. In theory, any algorithm used for explicit performance function reliability evaluation may be adapted to deal with implicit performance function. However, many difficulties occurring in realistic problem can not be conquered. As it has already represented in the previous chapter, there are surrogate methods, are based on a functional surrogate of the performance function. In this thesis, some machine learning algorithms are used as surrogates methods.

There are two categories of machine learning algorithms, **supervised** or **unsupervised**. In supervised learning, each training example is a pair consisting of an **input feature** and a **desired output** value and the algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. In unsupervised learning, the training data has only the **input values**. In this thesis will be used only supervised learning. Supervised learning problems are divided into to types, regression and classification. The regression methods have been applied in the structural reliability to alleviate the difficulties since 1990s. In the presented regression methods, an explicit function intended to substitute the implicit performance function is fitted through random samples under the empirical risk minimization (ERM), and the failure probability of explicit function replaces that of the actual implicit performance function.

In this dissertation, only regression methods are examined, which are named for their continuous outputs, meaning they may have any value within a range. There are plenty of algorithms, however Gaussian Process Regression and Support Vector Machine are chosen to become an efficient surrogate to the numerical solver of the finite element model which is repeatedly invoked in the Monte Carlo Simulation method.

## 4.2 Algorithms

### 4.2.1 Gaussian Process Regression

A wide variety of machine-learning models are used for structural reliability analysis in the literature as limit state function models. GPs appear as a promising model to use for approximating limit state function because they are the only approach that has the following properties: does not require a predefined structure, can approximate highly non-linear function landscapes, has meaningful hyperparameters, and includes a theoretical framework for optimizing their hyperparameters. We will exploit these advantages in GPR approximation method for structural reliability analysis.

In probability theory, a stochastic process is a mathematical object which is defined as the collection of an infinite number of random variables. A stochastic process $\mathcal{X}$ is said to be Gaussian if the joint pdf of the random variables $\{\mathcal{X}(t_1), ..., \mathcal{X}(t_n)\}$ is Gaussian for any n and $t_i = 1, ..., n$ and every finite collection of those random variables has a multivariate normal distribution, i.e. every finite linear combination of them is normally distributed. Gaussian Processes (GPs) are a powerful technique for modelling and predicting numerical data.

Gaussian process regression is nonparametric (i.e. not limited by a functional form), so rather than calculating the probability distribution of parameters of a specific function, GPR calculates the probability distribution over all admissible functions that fit the data [9]. However, similar to the above, we specify a prior (on the function space), calculate the posterior using the training data, and compute the predictive posterior distribution on our points of interest. There are several ways to interpret Gaussian process (GP) regression models. One can think of a Gaussian process as defining a distribution over functions, and inference taking place directly in the space of functions, the function-space two equivalent views. In machine learnig, Gaussian Processes are a generic supervised learning method designed to solve regression and probabilistic classification problems. It is concerned with the prediction of continuous continuities based on a discrete set of labeled data. A Gaussian Process assumes that the covariance between any set of points is a multivariate Gaussian.

Let us denote our training set of observations as $\mathcal{D}\{(x_i, y_i)|i = 1, ..., n\}$, where $x$ denotes an input vector (covariates) of dimension $D$ and $y$ denotes a scalar output or target (dependent variable). We write the inputs $x_i$ of $\mathcal{D}$ as a column vectors and form the D $\times$ n matrox $X$, called design matrix. Also, the targets are collected in the vector $y$, so we can write $\mathcal{D} = (X, y)$. We are interested in making inferences about the relationship between inputs and targets, which in this setting is viewed as a conditional distribution of the targets given the inputs.

The standard linear regression model with Gaussian noise

$$f(x) = x^\top w, \qquad y = f(x) + \epsilon, \tag{4.1}$$

where $x$ is the input vector, $w$ is a vector of weights (parameters) of the linear model, $f$ is the function value and $y$ is the observed target value. Often a bias weight or offset is included, but as this can be implemented by augmenting the input vector $x$ with an additional element whose value is always one, we do not explicitly include it in our notation. We have assumed that the observed values $y$ differ from the function values $f(x)$ by additive noise, and we will further assume that this noise follows an independent, identically distributed Gaussian distribution with zero mean and variance $\sigma_n^2$.

$$\epsilon \sim \mathcal{N}(0, \sigma_n^2) \tag{4.2}$$

The error variance $\sigma_n^2$ and the coefficients $w$ are estimated from the data. A GPR model explains the response by introducing latent variables, $f(x_i), i = 1, 2, ..., n$, from a Gaussian process (GP), and explicit basis functions, h. The covariance function of the latent variables captures the smoothness of the response and basis functions project the inputs x into a p-dimensional feature space. A GP is a set of random variables, such that any finite number of them have a joint Gaussian distribution. If $\{f(x), x \in \mathbb{R}^d\}$ is a Gaussian process, then $E(f(x)) = m(x)$ and $Cov[f(x), f(x^{'})] = E[\{f(x) - m(x)\}\{f(x^{'}) - m(x^{'})\}] = k(x, x^{'})$.

Now consider the following model.

$$h(x)^T w + f(x) \tag{4.3}$$

where $f(X) \sim GP(0, k(x, x^{'}))$, that is f(x) are from a zero mean GP with covariance function, $k(x, x^{'})$. $h(x)$ are a set of basis functions that transform the original feature vector $x$ in $\mathbb{R}^d$ into a new feature vector $h(x)$ in $R^p$. w is p-by-1 vector of basis function coefficients. This model represents a GPR model. An instance of response $y$ can be modeled as

$$P(y_i|f(x_i), x_i) \sim N(y_i|h(x_i)^T w + f(x_i), \sigma_n^2) \tag{4.4}$$

Hence, a GPR model is a probabilistic model. There is a latent variable $f(x_i)$ introduced for each observation $x_i$, which makes the GPR model nonparametric. In vector form, this model is equivalent to

$$P(y_i|f, X) \sim N(y|Hw + f, \sigma_n^2 I) \tag{4.5}$$

where $X = \begin{pmatrix} {x_1}^T \\ {x_2}^T \\ \vdots \\ {x_n}^T \end{pmatrix}$ , $y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$ , $H = \begin{pmatrix} h({x_1}^T) \\ h({x_2}^T) \\ \vdots \\ h({x_n}^T) \end{pmatrix}$ , $f = \begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{pmatrix}$ .

The joint distribution of latent variables $f(x_1), f(x_2), \cdots, f(x_n$ in the GPR model is as follows:

$$P(f|X) \sim N(f|0, K(X, X)), \tag{4.6}$$

close to a linear regression model, where $K(X, X)$ looks as follows:

$$K(X, X) \begin{pmatrix} k(x_1, x_{1)}) & k(x_1, x_{2)}) & \cdots & k(x_1, x_{n)}) \\ k(x_2, x_{1)}) & k(x_2, x_{2)}) & \cdots & k(x_2, x_{n)}) \\ \vdots & \vdots & \vdots & \vdots \\ k(x_n, x_{1)}) & k(x_n, x_{2)}) & \cdots & k(x_n, x_{n)}) \end{pmatrix} \tag{4.7}$$

The covariance function $k(x, x^{'})$ is usually parameterized by a set of kernel parameters or hyperparameters, $\theta$. Often $k(x, x^{'})$ is written as $k(x, x^{'}|\theta)$ to explicitly indicate the dependence on $\theta$.

Some of the commonly used GPR kernel functions are:

- Linear
$$k(x, x') = x^T x'$$
- Exponential
$$k(x, x') = \sigma_f^2 \exp(-\frac{r}{\sigma_l})$$
- Matern 5/2
$$k(x, x') = \sigma_f^2(1 + \frac{\sqrt{5}r}{\sigma_l} + \frac{\sqrt{5}r^2}{\sigma_l} \exp - \frac{\sqrt{5}r}{\sigma_l})$$
- Rational Quadratic
$$k(x, x') = \sigma_f^2(1 + \frac{r^2}{2\alpha\sigma_l^2})^{-a}$$

where, r is the Euclidean distance between $x$ and $x'$.

While the basis function can be one of the below functions:

- Constant $H = 1$
- Linear $H = [1, X]$
- Pure Quadratic $H = [1, X, X_2]$

### 4.2.2 Support Vector Machine

Support vector machine (SVM) analysis is a popular machine learning tool for classification and regression. SVM regression is considered a nonparametric technique because it relies on kernel functions. It is popularly and widely used for classification problems in machine learning. SVM tries to find a line/hyperplane (in multidimensional space) that separates these two classes. Then it classifies the new point depending on whether it lies on the positive or negative side of the hyperplane depending on the classes to predict.

In two-dimensions you can visualize this as a line and let's assume that all of our input points can be completely separated by this line.

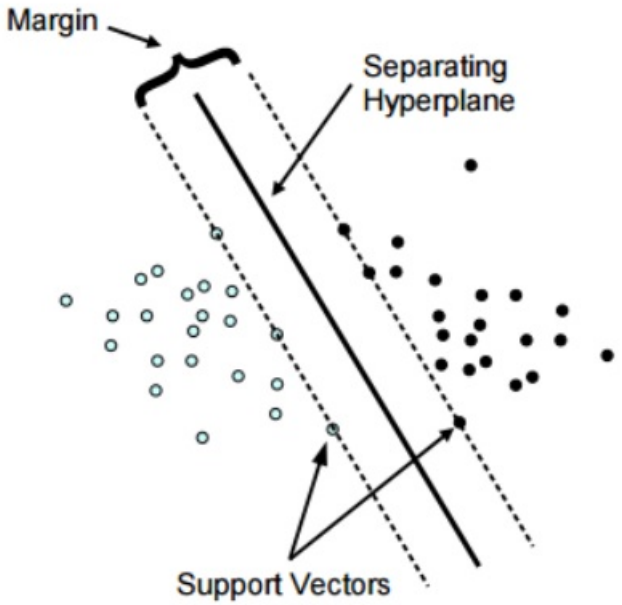



Figure 38: Maximum margin between the classes, Separating Hyperplane and Support Vectors.

- The distance between the line and the closest data points is referred to as the margin.
- The best or optimal line that can separate the two classes is the line that as the largest margin. This is called the Maximal-Margin hyperplane.
- The hyperplane is learned from training data using an optimization procedure that maximizes the margin

Support Vector Machine can also be used as a regression method, maintaining all the main features that characterize the algorithm (maximal margin). The Support Vector Regression (SVR) uses the same principles as the SVM for classification, with only a few minor differences. First of all, because output is a real number it becomes very difficult to predict the information at hand, which has infinite possibilities. In the case of regression, a margin of tolerance (epsilon) is set in approximation to the SVM which would have already requested from the problem. But besides this fact, there is also a more complicated reason, the algorithm is more complicated therefore to be taken in consideration. However, the main idea is always the same: to minimize error, individualizing the hyperplane which maximizes the margin, keeping in mind that part of the error is tolerated.

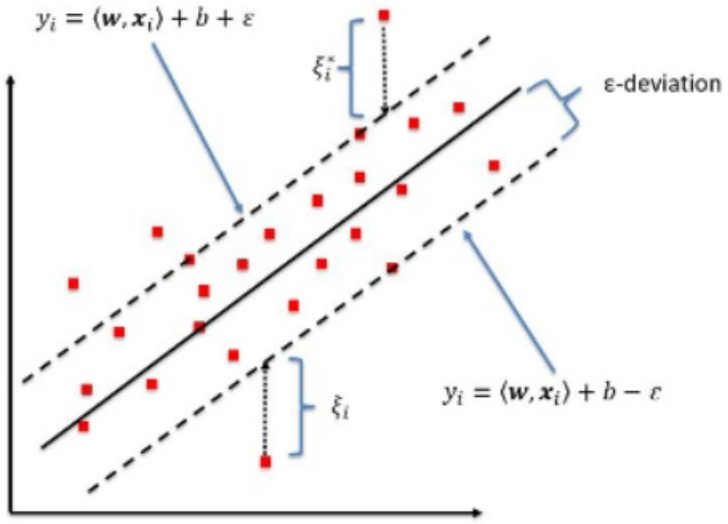



Figure 39: The insensitive band for a non-linear regression function

Consider these two dashed lines (Fig.39) as the decision boundary and the black line as the hyperplane. Our objective, when we are moving on with SVR, is to basically consider the points that are within the decision boundary line. Our best fit line is the hyperplane that has a maximum number of points. Support Vector Machines it will ensure sparseness of the dual variables. The idea of representing the solution by means of a small subset of training points has enormous computational advantages. Using the $\varepsilon$-insensitive loss function has that advantage, while still ensuring the existence of a global minimum and the optimization of a reliable generalization bound.

The mathematical formulation of the support vector machines according to [6] is presented below.

**Linear SVM Regression: Primal Formula**

Suppose we have a set of training data where $x_n$ is a multivariate set of $N$ observations with observed response values $y_n$.

To find the linear function

$$f(x) = x^{'}\beta + b \tag{4.8}$$

and ensure that it is as flat as possible, find $f(x)$ with the minimal norm value $(\beta\beta)$. This is formulated as a convex optimization problem to minimize

$$J(\beta) = \frac{1}{2}\beta^{'}\beta \tag{4.9}$$

subject to all residuals having a value less than $\epsilon$ or, in equation form:

$$\forall_n : |y_n - (x_n^{'\beta} + b)| \leq \epsilon \tag{4.10}$$

It is possible that no such function $f(x)$ exists to satisfy these constraints for all points. To deal with otherwise infeasible constraints, introduce slack variables $\xi_n$ and $\xi_n^*$ for each point. This approach is similar to the "soft margin" concept in SVM classification, because the slack variables allow regression errors to exist up to the value of $\xi_n$ and $\xi_n^*$, yet still satisfy the required conditions.

Including slack variables leads to the objective function, also known as the primal formula:

$$J(\beta) = \frac{1}{2}\beta^{'}\beta + C\sum_{n=1}^{N}(\xi_n + \xi_n^*), \tag{4.11}$$

subjected to:

$$\forall : y_n - (x_n)^{'}\beta + b) \leq \epsilon + \xi_n \tag{4.12}$$

$$\forall : (x_n)^{'}\beta + b) - y_n \leq \epsilon + \xi_n^* \tag{4.13}$$

$$\forall_n : \xi_n \geq 0 \tag{4.14}$$

$$\forall_n : \xi_n \geq 0 \tag{4.15}$$

The constant C is the box constraint, a positive numeric value that controls the penalty imposed on observations that lie outside the epsilon margin ($\epsilon$) and helps to prevent overfitting (regularization). This value determines the trade-off between the flatness of $f(x)$ and the amount up to which deviations larger than $\epsilon$ are tolerated.

The linear $\epsilon$-insensitive loss function ignores errors that are within  distance of the observed value by treating them as equal to zero. The loss is measured based on the

distance between observed value y and the  boundary. This is formally described by

$$L_\epsilon = \begin{cases} 0, if|y - f(x)| \leq \epsilon \\ |y - f(x)| - \epsilon, otherwise. \end{cases} \tag{4.16}$$

**Linear SVM Regression: Dual Formula**

The optimization problem previously described is computationally simpler to solve in its Lagrange dual formulation. The solution to the dual problem provides a lower bound to the solution of the primal (minimization) problem. The optimal values of the primal and dual problems need not be equal, and the difference is called the "duality gap." But when the problem is convex and satisfies a constraint qualification condition, the value of the optimal solution to the primal problem is given by the solution of the dual problem.

To obtain the dual formula, construct a Lagrangian function from the primal function by introducing nonnegative multipliers $\alpha_n$ and $\alpha_n^*$ for each observation $x_n$. This leads to the dual formula, where we minimize

$$L(\alpha) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(\alpha_i + \alpha_i^*)(\alpha_j + \alpha_j^*)\xi_i^{'}\xi_j + \epsilon\sum_{i=1}^{N}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{N} y_i(\alpha_i + \alpha_i^*) \tag{4.17}$$

subjected to constraints

$$\sum_{j=1}^{N}(\alpha_n + \alpha_n^*) = 0 \tag{4.18}$$

$$\forall_n : 0 \leq \alpha_n \leq C \tag{4.19}$$

$$\forall_n : 0 \leq \alpha_n^* \leq C \tag{4.20}$$

The  parameter can be completely described as a linear combination of the training observations using the equation

$$\beta = \sum_{j=1}^{N}(\alpha_n + \alpha_n^*)x_n. \tag{4.21}$$

The function used to predict new values depends only on the support vectors:

$$f(x) = \sum_{j=1}^{N}(\alpha_n + \alpha_n^*)(x_n^{'}x) + b. \tag{4.22}$$

**Nonlinear SVM Regression: Primal Formula**

Some regression problems cannot adequately be described using a linear model. In such a case, the Lagrange dual formulation allows the previously-described technique to be extended to nonlinear functions.

Obtain a nonlinear SVM regression model by replacing the dot product $x_1, x_2$ with a nonlinear kernel function $G(x_1, x_2) =< \phi(x_1), \phi(x_2) >$, where $\phi(x)$ is a transformation that maps $x$ to a high-dimensional space.

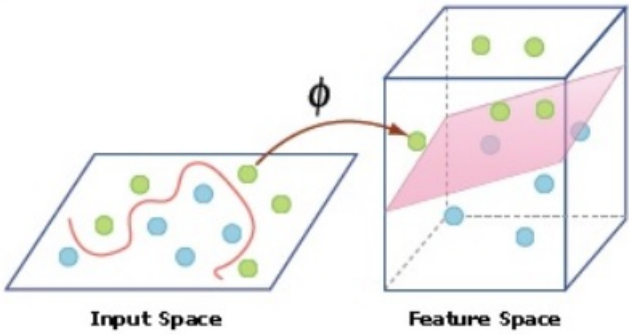



Figure 40: A non-linear separating region transformed in to a linear one.

The Gram matrix is an n-by-n matrix that contains elements $g_{bj} = G(x_i, x_j)$. Each element $g_{bj}$ is equal to the inner product of the predictors as transformed by $\phi$. However, we do not need to know $\phi$, because we can use the kernel function to generate Gram matrix directly. Using this method, nonlinear SVM finds the optimal function f(x) in the transformed predictor space.

**Nonlinear SVM Regression: Dual Formula** The dual formula for nonlinear SVM regression replaces the inner product of the predictors $(x_i, x_j)$ with the corresponding element of the Gram matrix $(g_i, j)$.

Nonlinear SVM regression finds the coefficients that minimize

$$L(\alpha) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(\alpha_i+\alpha_i^*)(\alpha_j+\alpha_j^*)G(x_i,x_j)+\epsilon\sum_{i=1}^{N}(\alpha_i+\alpha_i^*)-\sum_{i=1}^{N}y_i(\alpha_i+\alpha_i^*) \quad (4.23)$$

subject to

$$\sum_{j=1}^{N}(\alpha_n+\alpha_n^*)=0 \quad (4.24)$$

$$\forall_n : 0 \leq \alpha_n \leq C \quad (4.25)$$

$$\forall_n : 0 \leq \alpha_n^* \leq C \quad (4.26)$$

The function used to predict new values is equal to

$$f(x) = \sum_{j=1}^{N}(\alpha_n+\alpha_n^*)G(x_i,x_j+b. \quad (4.27)$$

**Kernel Functions**

- Linear

$$G(x_j^{'}, x_k) = x_j^{'} x_k \tag{4.28}$$

- Gaussian

$$G(x_j, x_k) = exp(-\|x_j - x_k\|^2) \tag{4.29}$$

- Polynomial

$$G(x_j, x_k) = (1 + x_j^{'} x_k)^q, \text{where } q \text{ is in the set} \{2, 3, \cdots\} \tag{4.30}$$

## 4.3 Bias and Variance

Solving the issue of bias and variance is really about dealing with over-fitting and under-fitting. Bias is reduced and variance is increased in relation to model complexity. As more and more parameters are added to a model, the complexity of the model rises and variance becomes our primary concern while bias steadily falls.

**Bias**: It gives us how closeness is our predictive model's to training data after averaging predict value. Generally algorithm has high bias which help them to learn fast and easy to understand but are less flexible. That looses it ability to predict complex problem, so it fails to explain the algorithm bias. This results in underfitting of our model.

**Variance**: It defines as deviation of predictions, in simple it is the amount which tell us when its point data value change or a different data is use how much the predicted value will be affected for same model or for different model respectively. Ideally, the predicted value which we predict from model should remain same even changing from one training data-sets to another, but if the model has high variance then model predict value are affect by value of data-sets.

Poor performance of a model is caused either by underfitting or overfitting of the data. Both are visualized in Fig.41, as well as a well fitted/robust model. In the case of underfitting, the model cannot capture the complex behaviour of the data, in other words the model is too simple, resulting in poor performance. An example is a linear model that cannot capture non-linear behaviour. Overfitting occurs when the model is 'too well' fitted to the training data and has the tendency to capture the noise of the dataset instead of its general trends. In the case of overfitting, the model is not generalizing enough for new data, meaning that it doesn't perform adequately on data it has never seen and therfore results in poor performance.

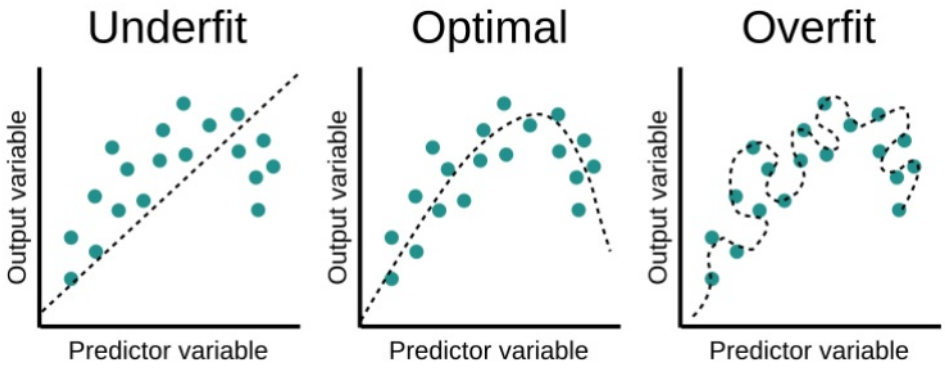



Figure 41: Underfitting, Optimal, Overfitting

Cross-validation is a model assessment technique used to evaluate a machine learning algorithm's performance in making predictions on new datasets that it has not been trained on. This is done by partitioning the known dataset, using a subset to train the algorithm and the remaining data for testing. Each round of cross-validation involves randomly partitioning the original dataset into a training set and a testing set. The training set is then used to train a supervised learning algorithm and the testing set is used to evaluate its performance. This process is repeated several times and the average cross-validation error is used as a performance indicator.

When training a model, it is important not to overfit or underfit it with algorithms that are too complex or too simple. Your choice of training set and test set are critical in reducing this risk. However, dividing the dataset to maximize both learning and validity of test results is difficult. This is where cross-validation comes into practice. Cross-validation offers several techniques that split the data differently, to find the best algorithm for the model.

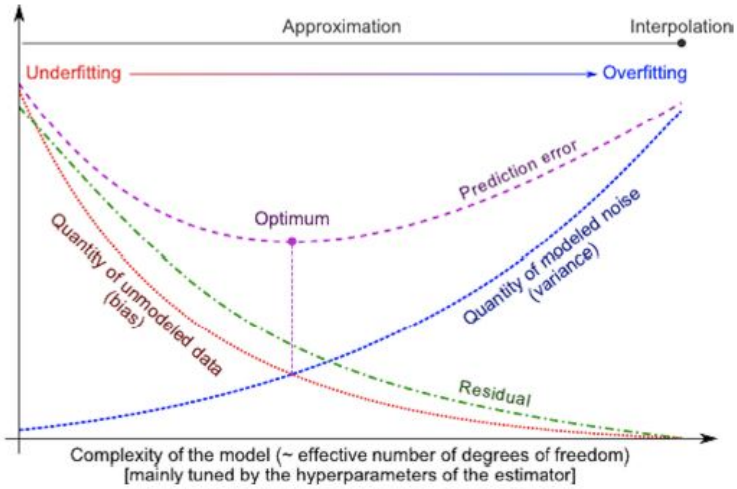



Figure 42: Optimum model complexity

## 4.4 Cross-Validation

In 4.3, it is has mentioned that the goal is to create a model with optimal behavior. Using Cross-validation, the goal is to test the model's ability to predict new data that was not used in estimating it, in order to flag problems like overfitting or selection bias and to give an insight on how the model will generalize to an independent dataset (i.e., an unknown dataset, for instance from a real problem). Cross-validation is a resampling method that uses different portions of the data to test and train a model on different iterations. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice. In a prediction problem, a model is usually given a dataset of known data on which training is run (training dataset), and a dataset of unknown data (or first seen data) against which the model is tested (called the validation dataset or testing set).

In machine learning (ML), generalization usually refers to the ability of an algorithm to be effective across various inputs. It means that the ML model does not encounter performance degradation on the new inputs from the same distribution of the training data. For human beings generalization is the most natural thing possible. We can classify on the fly. For example, we would definitely recognize a dog even if we didn't see this breed before. Nevertheless, it might be quite a challenge for an ML model. That's why checking the algorithm's ability to generalize is an important task that requires a lot of attention when building the model.

Cross-validation is a technique for evaluating a machine learning model and testing its performance. It helps to compare and select an appropriate model for the specific predictive modeling problem. CV is easy to understand, easy to implement, and it tends to have a lower bias than other methods used to count the model's efficiency scores. Moreover, it can also give estimates of the variability of the true error estimation which is a useful feature. All this makes cross-validation a powerful tool for selecting the best model for the specific task. One round of cross-validation involves partitioning a sample of data into complementary subsets, performing the analysis on one subset (called the training set), and validating the analysis on the other subset (called the validation set or testing set). To reduce variability, in most methods multiple rounds of cross-validation are performed using different partitions, and the validation results are combined (e.g. averaged) over the rounds to give an estimate of the model's predictive performance.

Many techniques are available for cross-validation. Among the most common are:

**Holdout**: Partitions data randomly into exactly two subsets of specified ratio for training and validation. This method performs training and testing only once, which cuts execution time on large sets, but interpret the reported error with caution on small data sets.

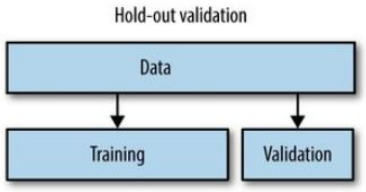



Figure 43: Hold-out data split

**k-fold**: Partitions data into k randomly chosen subsets (or folds) of roughly equal size. One subset is used to validate the model trained using the remaining subsets. This process is repeated k times such that each subset is used exactly once for validation. The average error across all k partitions is reported as $\epsilon$. This is one of the most popular techniques for cross-validation but can take a long time to execute because the model needs to be trained repeatedly. The image below illustrates the process. For example, a 5-fold technique is shown in the the below picture.

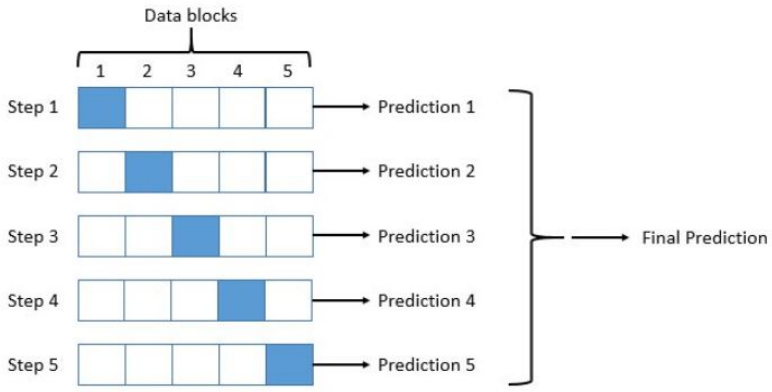



Figure 44: Diagram of the 5-fold cross-validation method (blocks in blue represent the testing folds at each step).

As can be seen, cross-validation is very similar to the holdout method. Where it diers, is that each data point is used both to train models and to test a model, but never at the same time.

In this dissertation, the 5-fold cross-validation method is used to assess the performance of the SVM and GPR models. It is worth noting that, the cross-validation loss is also used as the objective function within the Bayesian optimization, which is discussed in the next section.

## 4.5 Hyper-parameter Optimization

In machine learning, hyperparameter optimization or tuning is the problem of choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a parameter whose value is used to control the learning process. By contrast, the values of other parameters (typically node weights) are learned. The aim of hyper-parameter optimization in machine learning is to find the hyperparameters of a given machine learning algorithm that return the best performance as measured on a validation set. In other words, hyper-parameter optimization finds a group of hyper-parameters that yields an optimal model which minimizes a predefined loss function on given independent data.

The same kind of machine learning model can require different constraints, weights or learning rates to generalize different data patterns. These measures are called hyperparameters, and have to be tuned so that the model can optimally solve the machine learning problem. Hyperparameter optimization finds a tuple of hyperparameters that yields an optimal model which minimizes a predefined loss function on given independent data. The objective function takes a tuple of hyperparameters and returns the associated loss. Cross-validation is often used to estimate this generalization performance.

Manual hyper-optimization is labor-intrensive issue, as the problem is that the evaluating the objective function to find the score extremely expensive. Considering that each time different hyper-parameters are tried, a model on the training data is trained to make predictions on the validation data and then the validation metric is calculated. Grid search and random search overcome the problem of the manual hyper-optimization, because a grid of model hyper-parameters is set up and the train-predict-evaluate cycle runs automatically in a loop. The accuracy and the efficiency of these methods are relatively low due to the fact that the next hyper-parameters for evaluation are not based on previous results. Grid and random search are completely uninformed by past evaluations and as a result, often spend a significant amount of time evaluating “bad” hyperparameter.

Bayesian approach considers as the most efficient solution, it keeps track of past evaluation results which they use to form a probabilistic model mapping hyper- parameters to a probability of a score on the objective function $P(score|hyperparameters)$. This model is called a surrogate for the objective function. Optimization with surrogate is much more accurate and easier, rather than with the objective function, as Bayesian methods work by finding the next set of hyper-parameters to evaluate on the actual objective function by selecting hyper-parameters that perform best on the surrogate function $f(x)$. Gaussian Process is used by Bayesian Optimization to fit the surrogate model $f(x)$. One innovation in Bayesian optimization is the use of an acquisition func-

tion, which the algorithm uses to determine the next point to evaluate. The acquisition function can balance sampling at points that have low modeled objective functions and exploring areas that have not yet been modeled well.

The algorithm evaluates $y_i = f(x_i)$ for $n$ points $x_i$, taken at random within the variable bounds. If there are evaluation errors, it takes more random points until there are n successful evaluations. The probability distribution of each component is either uniform or log-scaled.

Then it repeats the following steps:

1. Updates the Gaussian process model of $f(x)$ to obtain a posterior distribution over functions $Q(f|x_i, y_i for i = 1, ..., t)$.

2. Finds the new point $x$ that maximizes the acquisition function $a(x)$.

The algorithm stops after reaching a fixed number of iterations or a fixed time or a stopping criterion.

The acquisition function evaluates a point x based on the posterior distribution function $Q$. Then it selects the point with the lowest expected loss. Some of the acquisition functions that can be used in Bayesian optimization is expected improvement, probability of improvement and lower confidence bound.

In this dissertation, Bayesian hyper-parameter optimization is used with expected improvement acquisition function. The expected improvement acquisition functions evaluates the expected amount of improvement in the objective function, ignoring values that cause an increase in the objective. In other words, it defines $x_{best}$ as the location of the lowest posterior mean and $\mu_Q(x_{best})$ as the lowest value of the posterior mean. Then the expected improvement is $EI(x, Q) = E_Q[max(0, \mu_Q(x_{best}) f(x))]$. Bayesian hyper-parameter optimization finds the optimum hyper-parameters which minimize the cross-validation loss, following the above procedure. Then, these objective function evaluations, namely the optimum model hyper-parameters are used to train a new cross-validated model.

## 4.6 Performance Evaluation Metrics

In order to compare different models among each other, performance metrics are needed. Dependent on the type of problem different metrics can be used. The most common used metric of machine learning regressions models are discussed below:

The error rate of the regression models is measured with the root mean square error, which is given:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{4.31}$$

Besides the root mean square error, the predicted vs. actual plot and the residuals plot are also used in order to visualize the results of a regression model. The predicted vs. actual shows how well the regression model makes predictions for different response values. The predicted response of the model is plotted against the actual, true response. A perfect regression model has a predicted response equal to the true response, so all the points lie on a diagonal line. The vertical distance from the line to any point is the error of the prediction for that point. A good model has small errors, and so the predictions are scattered near the line. The residuals plot displays the difference between the predicted and true responses. Usually a good model has residuals scattered roughly symmetrically around zero. If any clear patterns in the residuals exists, it is likely that the model needs improvement.

# 5. Machine Learning Applications

## 5.1 Introduction

The theory behind the machine learning algorithms, the hyper-parameter tuning as well as their performance evaluation are discussed in chapter 4. Reliability methods are presented in chapter 2 , which are an analytical method, First-order Reliability method and simulation methods, such as **Monte Carlo Simulation** and the Subset simulation. In this chapter machine learning algorithms and the Monte Carlo method are combined to replace the explicit limit state function and to calculate **probability of failure**, with shortened computational cost.

Since a machine learning model with low generalization error is trained, the response of the failure function is calculated faster than before, as it is not calculated by the finite element model. A well trained model can be very useful, for replacement the FEM model at every iteration or simulation of the reliabiliy methods. The considered models, the preparation of the training data, the implementation of the models using MATLAB and their performance comparison are presented in the next sections.

Specifically, a training set which consists the random variables of the problem and the response of the failure function is created and used for two regression models, **Gaussian Process Regression** [10], [11] and **Support Vector Machine** [7], [8]. The performance and accuracy are compared in combination with the compatibility with reliability methods.

In the proposed method, regressions models are applied to approximate the limit state function. The approximation function is used to replace the finite element analysis to save the computation time. The main procedure of the proposed method includes the following: (1) generating training datasets to establish a regression model; (2) training the regression model with the training datasets; (3) extracting the explicit approximation formulation using the well trained regression model; (4) predicting the failure probability using the Monte Carlo simulation.

## 5.2 Machine Learning Models-MATLAB

### 5.2.1 Gaussian process regression

MATLAB's **fitrgp** function trains and cross-validates a Gaussian process regression model. Bayesian optimization is used to estimate the optimum kernel and basis functions and the noise variance $\sigma^2$ for the examined data-set. The objective function of the optimization is $log(1 + cross - validationloss)$, as it is in any regression model.

```
1. gprMdl = fitrgp(...
2.                 predictors,...
3.                 response,...
4.                 'BasisFunction','none',...
5.                 'KernelFunction','ardrationalquadratic',...
6.                 'Standarize',false,...
7.                 'Sigma',σ_opt...
8.               )
```

### 5.2.2 Support vector regression

MATLAB's **fitrsvm** trains or cross-validates a support vector machine (SVM) regression model on a low- through moderate-dimensional predictor data set. It supports mapping the predictor data using kernel functions, and supports SMO, ISDA, or L1 soft-margin minimization via quadratic programming for objective-function minimization.

```
1. regressionSVM = fitrsvm(...
2.                 predictors,...
3.                 response,...
4.                 'Standarize',true,...
5.                 'KFold',5,...
6.                 'KernelFunction','polynomial',...
7.                 'Polynomialorder',3,...
8.                 'KernerScale','auto')
```

## 5.3 Example 1: Truss-13 random variables

The 2d-truss in Fig.45 has the same material, geometry properties and random variables as in section 3.3.

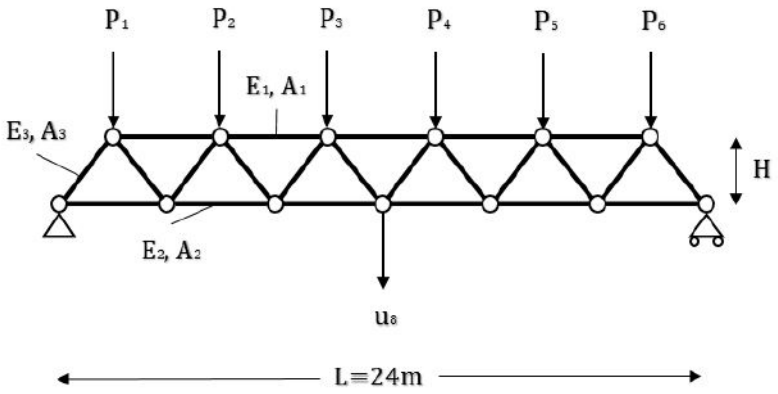



Figure 45: 2D-Truss: geometry and loads

### 5.3.1 Sample Data

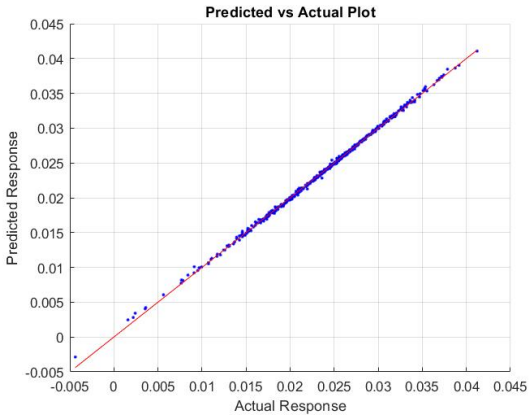
The aim of the machine learning regression models, as it has already mentioned, is to replace the explicit limit state function for the calculation of the probability of failure, reducing the computational cost. Thus, a sample data set is needed for the training and the cross-validation of the models, in order to predict response of the limit state function with low error. More specifically a set of input and output is produced with Monte Carlo Simulation. The input contains all the random variable of the truss application ($E_1, E_2, E_3, A_1, A_2, A_3, H, P_1 - P_6$) and the output the result of the limit state equation for each of the Monte Carlo simulation.

For the **Gaussian Process Regression**, 500 data are produced with acceptable error, while for **Support Vector machine regression** 1000 have produced.

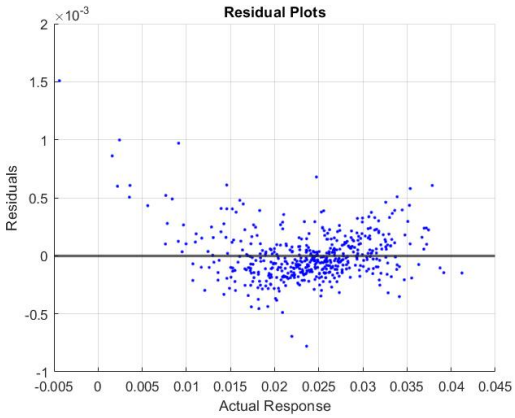
### 5.3.2 Performance Evaluation of the models

The evaluation of each machine learning model arises from the RSME error and two plots, actual-predicted response and residual plot.

1.**Gaussian Process Regression**



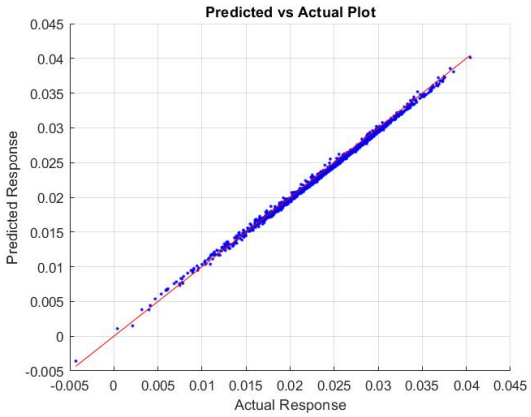

(a) Response calculated from the FEM model (Actual)-Response calculated from the machine learning algorithm (Predicted)



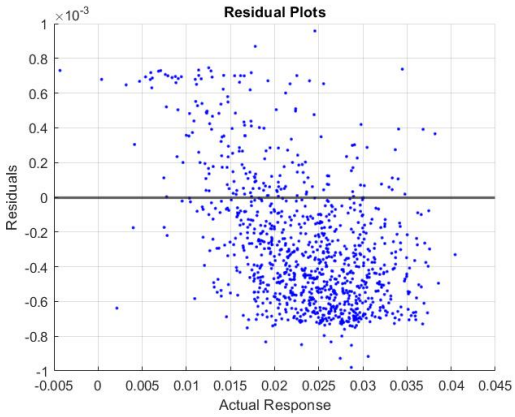

(b) Residual: Actual-Predicted

Figure 46: Performance of Gaussian process regression

2.**Support Vector Machine**




(a) Response calculated from the FEM model (Actual)-Response calculated from the machine learning algorithm (Predicted)




(b) Residual: Actual-Predicted

Figure 47: Performance of Support Vector regression

Figures.46 and 47 compare the predictions of the trained regression model with the finite element analyses for the training set. An excellent correlation can be observed between the predictions of the regressions model and the finite element analysis results. This result suggests that the model can be used to replace the FEM model with great efficiency and accuracy.

Table 8: Regression error

| Model | RMSE |
|---|---|
| GPR | $2.01\times10^{-5}$ |
| SVR | $4.56\times10^{-5}$ |

As can be seen at the performance plots, the Gaussian process regression model has the best performance. Besides the RMSE which is very low, the performance plots of the model indicate that it perfectly fits the training data and accurately predicts the failure function.

### 5.3.3 Monte carlo plus machine learning

The results from different methods are shown in Table 9. The result of the proposed methods show good agreement with those of MCS with 100.000 samples. Different numbers of simulations are generated and the computational time is presented in Table.9. GPR performance's arise the most robust and most accurate, as the probability of failure is almost the same with MCS, however the computational time is the largest of all the models. For SVM, the accuracy is lower, while the number of training data should be 1.000, however the computational time is minimized.

Table 9: Computational time with Monte Carlo Simulation and **Machine learning models** for the **Truss** problem.
MCS (N=100.000), $p_f$=$2.2\times10^{-3}$, time=69.52s

| MC simulations | **GPR**-Time (s) | $p_f$ | **SVM**-Time (s) | $p_f$ |
|---|---|---|---|---|
| 1000 | 7.54 | | 2.00 | |
| 10000 | 7.63 | | 2.07 | |
| 100000 | 10.15 | $1.90\times10^{-3}$ | 2.13 | $1.82\times10^{-3}$ |
| 1000000 | 35.34 | | 2.34 | |
| 10000000 | | | 6.02 | |

## 5.4 Example 2: Frame-11 random variables

The 2d-frame in Fig.48 has the same material, geometry properties and random variables as in section 3.4.

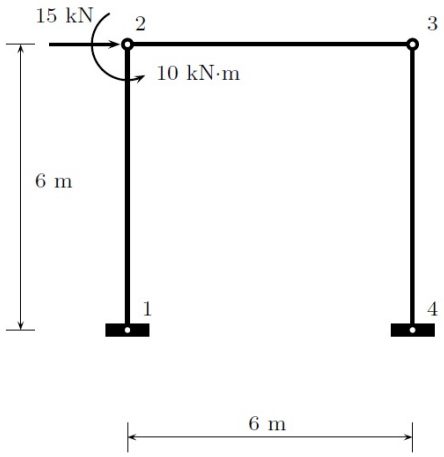



Figure 48: 2D frame example:geometry and loads.

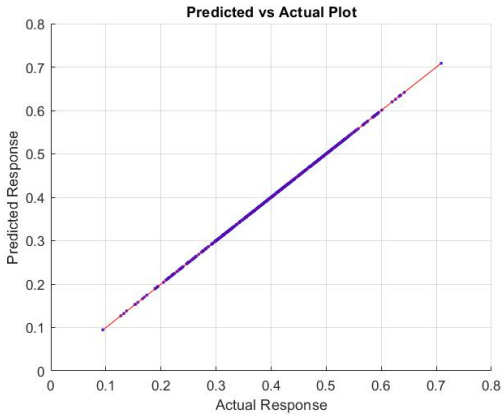
### 5.4.1 Sample Data

The aim of the machine learning regression models, as it has already mentioned, is to replace the explicit limit state function for the calculation of the probability of failure, reducing the computational cost. Thus, a sample data set is needed for the training and the cross-validation of the models, in order to predict response of the limit state function with low error. More specifically a set of input and output is produced with Monte Carlo Simulation. The input contains all the random variable of the frame application ($E_1, E_2, E_3, A_1, A_2, A_3, I_1, I_2, I_3, P, M$) and the output the result of the limit state equation for each of the Monte Carlo simulation.

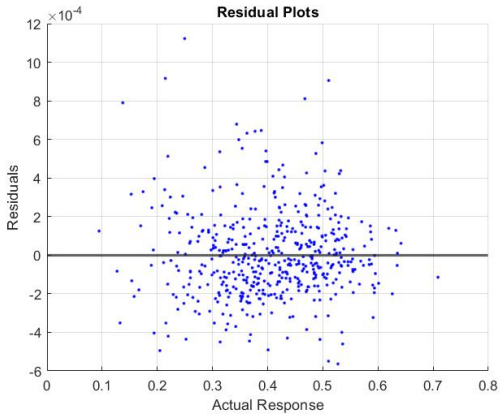
For the **Gaussian Process Regression**, 500 data are produced with acceptable error, while for **Support Vector machine regression** 1000 have produced.

### 5.4.2 Performance Evaluation of the models

#### 1.Gaussian Process Regression



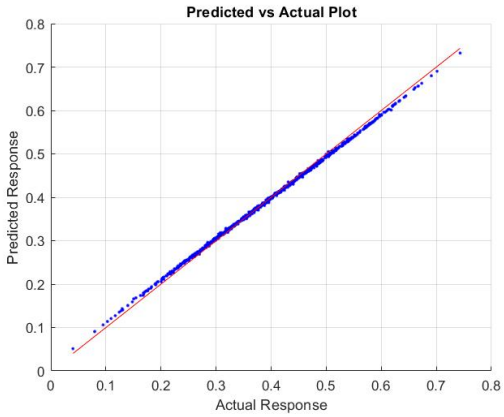

(a) Response calculated from the FEM model (Actual)-Response calculated from the machine learning algorithm (Predicted)



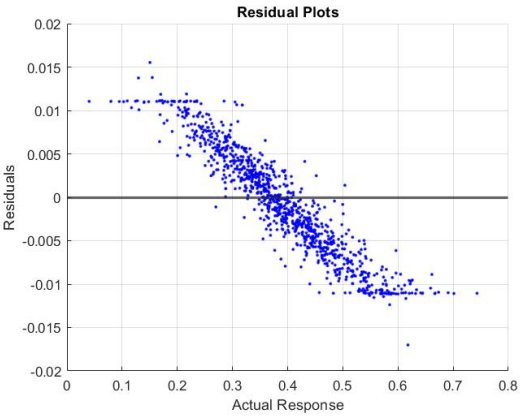

(b) Residual: Actual-Predicted

Figure 49: Performance of Gaussian process regression

#### 2.Support Vector Machine




(a) Response calculated from the FEM model (Actual)-Response calculated from the machine learning algorithm (Predicted)




(b) Residual: Actual-Predicted

Figure 50: Performance of Support Vector regression

Figures.49 and 50 compare the predictions of the trained regression model with the finite element analyses for the training set. An excellent correlation can be observed between the predictions of the regressions model and the finite element analysis results. This result suggests that the model can be used to replace the FEM model with great efficiency and accuracy.

As can be seen at the performance plots, the Gaussian process regression model has the best performance. Besides the RMSE which is very low, the performance

Table 10: Regression error

| Model | RMSE |
|---|---|
| GPR | $1.41\times10^{-5}$ |
| SVR | $7.52\times10^{-4}$ |

plots of the model indicate that it perfectly fits the training data and accurately predicts the failure function.

### 5.4.3 Monte carlo plus machine learning

The results from different methods are shown in Table 11. The result of the proposed methods show good agreement with those of MCS with 1.000.000 samples. Different numbers of simulations are generated and the computational time is presented in Table.11. Gaussian process regression performance's arises the most accurate, as the probability of failure is almost the same with MCS, however the computational cost is the largest of all the models, while the number of training data should be 500. For Support vector machine 1000 sampling data are generated, but the computational time is the least of all the models, while the accuracy of the model is efficient.

Table 11: Computational time with Monte Carlo Simulation and **Machine learning models** for the **Frame** problem.
MCS (N=1.000.000), $p_f$=$1.9\times10^{-4}$, time=845.38s

| MC simulations | **GPR**-Time (s) | $p_f$ | **SVM**-Time (s) | $p_f$ |
|---|---|---|---|---|
| 1000 | 7.37 | | 1.93 | |
| 10000 | 7.51 | | 1.96 | |
| 100000 | 9.68 | | 2.01 | |
| 1000000 | 33.04 | $1.81\times10^{-4}$ | 2.21 | $1.21\times10^{-4}$ |
| 10000000 | | | 4.89 | |

# 6. Conclusions

The purpose of this master's dissertation is to investigate the application of several Structural Reliability algorithms, as their eficiency and accuracy in combination with machine learning regression algorithms. First-Order Reliability, Monte Carlo simulation and Subset Simulation have been applied to structural analysis problems to define the probability of failure. Next, Gaussian process regression and Support Vector machine regression have been used to approximate the limit state equation, which includes a finite element model, while using Monte Carlo simulation the probability of failure has been calculated.

Initially, the three reliability methods have been implemented in structural analysis problem, a three-span continuous beam, in which the limit state equation is explicit, a 23-bar two dimensional truss and frame with three beam elements, in which the performance function involves the finite element method. The results from all the methods are compatible. FORM method is applied efficiently to these three problems. Using FORM, it has been calculated some sensitivity measures corresponding to each random variable, which defines if this random variable is crucial for the probability of failure. This measures provide useful information for the most significant parameters, which should be considered as random variables.

Monte Carlo simulation has verified the accuracy of FORM method results, however the computational time especially for the applications, which includes FEM model is significant. Considering that a probability $10^{-3}$ demands 100.000 (c.o.v.=10%), and $10^{-4}$ needs 1.000.000. simulations.

Subset Simulation reduces drastically the computational cost of Monte Carlo and it produces reliable results. This method estimates small probability of failure replacing it from a sequence of conditional probabilities, which are more frequent. For the efficiency calculation of these probabilities the modified Metropolis algorithm is used. The conditional probability is chosen to be $p = 0.1$, which affect the number of samples at each conditional level. Moreover, the samples of Markov chains, which are produced with Subset simulation could be used to define if a random variable affects (histogram) significant the probability of failure. Thus, considering the random variables with right skewed histograms (truss:24, 25,frame:35, 36), they affect more the probability of failure, while the results are compatible with the sensitivity measures

with FORM method (truss:18, frame:29 ) .

To deal extensively with issues on implicit performance function and huge computational cost in reliability analysis, a method for structural reliability analysis was proposed by combining regression models with Monte Carlo simualtion. Firstly, a small amount of training dataset is generated by the structural analysis to train the machine learning models. Then, the implicit performance is approximated by the trained models using explicit formulations. Two numerical examples, structural problems, illustrated the application and effectiveness of this combination. Comparisons were made with the classical reliability methods to evaluate the accuracy and computational efficiency of MC with ML. The examples showed that this mixture provides accurate results and is a computationally efficient approach for estimating the probability of failure of structures. Compared with classical methods, the combination is much more economical in achieving reasonable accuracy when dealing with problems with implicit limit state function and evaluating implicit limit state function frequently performed using the time-consuming finite element analysis.

Gaussian process regression and support vector regression algorithms were used for predicting the failure function. Gaussian process is the most accurate and efficient, as it needs the least number of sample data (500) to estimate the probability of failure, however the computational time is significant. Support vector machines are considered as the best performing regarding the computational time, as the demanding computational cost is minimized, however it needs 1000 (planar truss) and 1000 (planar frame) samples, while the estimated probability of failure is calculated with lower accuracy, as it slightly differs from the original, which is calculated using the classical reliability methods. In the case of large sample, SVR-MCS method is obviously superior to MCS method in contrast to computation cost, considering that 10.000.000 simulations are executed in 6.02s and 4.89s for the two examples respectively, while 100.000 MC simulations require 69.52s and 845.38s.

# Bibliography


[1] Andrzej S. Nowak, Kevin R. Collins. *Reliability of Structures*, McGraw-Hill Science/Engineering/Math 2000.

[2] Robert E. Melchers, André T. Beck. *Structural Reliability Analysis and Prediction*, John Wiley  Sons Ltd 2017.

[3] Siu-Kui Au, James L. Beck, *Estimation of small failure probabilities in high dimensions by subset simulation.* Probabilistic Engineering Mechanics;16(2001): 263-277.

[4] Zuev K.M. (2013). *Subset Simulation Method for Rare Event Estimation: An Introduction.* In: Beer M., Kougioumtzoglou I., Patelli E., Au IK. (eds) Encyclopedia of Earthquake Engineering. Springer, Berlin, Heidelberg.

[5] A. M. Hasofer and M. C. Lind, *An Exact and Invariant First Order Reliability Format*, Journal of Engineering Mechanics, Vol. 100, 1974, pp. 111-121.

[6] N. Christianini, J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods.*, Cambridge, UK: Cambridge University Press, 2000

[7] LI Hong-shuang, U Zhen-zhou, YUE Zhu-feng, *SUPPORT VECTOR MACHINE FOR STRUCTURAL RELIABILITY ANALYSIS*, Applied Mathematics and Mechanics, 2006, 27(10):1295–1303.

[8] Claudio M. Rocco, Jose Ali Moreno *Fast Monte Carlo reliability evaluation using support vector machine*, Reliability Engineering and System Safety 76 (2002) 237-243.

[9] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning.* MIT Press, 2006.

[10] Guoshao Su, Bo Yu, Yilong Xiao and Liubin Yan, *Gaussian Process Machine-Learning Method for Structural Reliability Analysis*, Advances in Structural Engineering, 2014, 17(9):1257-1270.

[11] Guoshao Su, Jianqing Jiang, Bo Yu and Yilong Xiao*A Gaussian process-based response surface method for structural reliability analysis*, Structural Engineering Mechanics, 2015, 56(4):549-567.

# A. Appendix

```
% FORM - Hasofer-Lind and Rackwitz-Fiessler tranform
1. function[pf,b,a] = FORM_HLRF(RV_list,Properties,initialRV,dist)
2. N_possible_rv = size(Properties,1);
3.index_RV = 0;
4.index_NRV =0;
5.N_RV = ones(1,3);
% Identification of the random variables
6.for indexprv=1:N_possible_rv
7.     if isequal(Properties(indexprv,3),RV_list(indexprv,1))==1
8.          index_RV = index_RV +1;
9.          RV(index_RV,1) = Properties(indexprv,1);
10.          RV(index_RV,2) = Properties(indexprv,2);
11.          RV(index_RV,3) = Properties(indexprv,3);
12.     else
13.          index_NRV= index_NRV + 1;
14.          N_RV(index_NRV,1) = Properties(indexprv,1);
15.          N_RV(index_NRV,2) = Properties(indexprv,2);
16.          N_RV(index_NRV,3) = Properties(indexprv,3);
17.      end
18.     if initialRV == Properties(indexprv,3)
19.          eq_initialRV = index_RV;
20.       end
21.end
22.N_RV = flipud(sortrows(N_RV,3));
23.not_rv = size(N_RV,1);
24.rv = size(RV,1);
```

```
25.b = 0;
26.iteration =0;
27.a = zeros(1,rv);
28.g = zeros(1,rv);
29.gold = ones(1,rv);
30.X = Properties(:,1);
31.X(initialRV) = double(find_initial_RV(X,initialRV));
32.RV(eq_initialRV,1) = X(initialRV);
33.k =0;
34. tol=0.001;
35. X = Problem_Properties(:,1);
36. X(initialRV) = double(find_initial_RV(X,initialRV));
37. while true
38.       for index_var1=1:rv
39.       distr=dist(index_var1);
40.           if strcmp(class(distr),class(makedist('Normal')))==0
41.             nnormalrv=nnormalrv+1;
42.            eq_normal(nnormalrv)=RF_transf(distr,double(RV(index_var1,1)));
43.           end
44.       end
45.       nonnormal = 0;
46.       for index_var2=1:rv
47.           if strcmp(class(dist(index_var2)),class(makedist('Normal')))==1
48.               N_means(index_var2) = mean(dist(index_var2));
49.              N_std(index_var2) = std(dist(index_var2));
50.          elseif strcmp(class(dist(index_var2)),class(makedist('Normal')))==0
51.               nonnormal = nonnormal +1;
52.               N_means(index_var2) = mean(eq_normal(nonnormal));
53.               N_std(index_var2) = std(eq_normal(nonnormal));
54.           end
55.           RV(index_var2,1) = double(N_means(index_var2));
56.           RV(index_var2,2) = double(N_std(index_var2));
57.      end
58.       aold=a;
59.       bold=b;
60.       iteration=iteration+1;
```

```
61.  if iteration==1
62.      X(initialRV)=double(find_initial_RV(X,initialRV));
63.      z=zeros(1,rv);
64.      z(eq_initialRV)=(X(initialRV) - RV(eq_initialRV,1))/RV(eq_initialRV,2);
65.  else
66.      gold=g;
67.     z=double(a.*b);
68.     z(eq_initialRV) = (X(initialRV) - RV(eq_initialRV,1))/RV(eq_initialRV,2);
69.     m = RV(:,1).';
70.     s = RV(:,2).';
71.     X = double(m+z.*s);
72.     if rv<N_possible_rv
73.        temp=[X.';N_RV(:,1)];
74.        temp1 = [RV(:,3);N_RV(:,3)];
75.        temp2 = [temp temp1];
76.        temp3 = sortrows(temp2,2);
77.        X = temp3;
78.        X = double(X(:,1));
79.      end
80.   X(initialRV) = double(find_initial_RV(X,initialRV));
81.  for k=1:N_possible_rv
82.     d(k) = -double(central_dif_method(X,k));
83.  end
84.  if rv<N_possible_rv
85.     for j=1:size(N_RV,1)
86.       d(N_RV(j,3)) = [];
87.       X(N_RV(j,3)) = [];
88.     end
89.  end
90.  g = RV(:,2).*d.';
91.  sq = sqrt(g.'*g);
92.  gt = g.'*z.';
93.  b = gt/sq;
94.   a = g.'/sq;
```

```
89.     t = max(abs(a-aold));
90.     t = max(t,abs(b-bold));
91.     if t > tol
92.
93.     else
94.       break
95.     end
101.     B(ii)=b;
102.     P_Failure(ii)=normcdf(-b)
103.end
104.grid on
105.plot(No_iterations,B,'r-*')
106.ylabel('Reliability Index')
107.xlabel('Iterations')
108.title('Convergence Plot')
109.b = double(b);
110.pf = double(normcdf(-b));
111.fprintf('Using the FORM-HLRF :');
112.fprintf('Iterations:  %g\nFailure probability = %g\nbeta=%g\n\n', ii,pf,b);
113.end
```

```
% Rackwitz-Fiessler - Function
1.function[obj_normal] = RF_transf(obj_dist,x)
2.w = norminv(cdf(obj_dist,x));
3.s_N = normpdf(w)/(pdf(obj_dist,x);
4.m_N = x - norminv(cdf(obj_dist,x))*s_N;
5.obj_normal = makedist('Normal', 'mu', m_N, 'sigma', s_N);
6.end
```

```
% Central Difference Method - Function
function [gd] = central_dif_method(X,k)
h1 = 0.25;
h = h1*X(k);
X(k) = X(k) + h;
u1 = gfail(X);
X(k) = X(k) - 2*h;
u2 = gfail(X);
gd = (u1-u2)/(2*h);
end
```

```
%Monte Carlo - Function
function [pf,gf,X]=MC(Properties,dist,N_MC_Simulation)
N_possible_rv = size(Problem_Properties,1);
X = ones(N_MC_Simulations,N_possible_rv);
for i=1:N_possible_rv
        X(:,i)=random(dist(i),[N_MC_Simulations,1]);
end
for j=1:N_MC_Simulations
        gf(j) = gfail(X(j,:));
end
N_Failure = length(find(gf<0));
pf = N_Failure/N_MC_Simulations;
fprintf('Using the crude MC :');
fprintf('Total number of samples: \%g\nFailure probability = %g\n\n',
N_MC_Simulations,pf);
end
```

```
% Failure function
% This file depends on the problem.
% Random variables are placed in vector X according to matrix Properties in
% the data file.
function[g] = gfail(X)
g = (X(:,2)*X(:,1)*(d-0.59*X(:,2)*X(:,1))./(X(:,3)*b))-X(:,4);
end
```

```
% Subset Simulation - Function
function[pF_SS] = SubsetSim(Properties,dist,N_Simulations_Level)
1.p=0.1;
2.nc=N_Simulations_Level*p;
3.ns=(1-p)/p;
4.Level=0;
5.[∼,N,gf,x] = MC(Properties,dist,N_Simulations_Level);
6.nF = N;
7.y = gf;
8.YF = 0;
9.index_nrv=0;
10.index_rv=0;
11.for index_var=1:size(Properties,1)
12.     if std(dist(index_var))==0
13.         index_nrv = index_nrv +1;
14.         n_rv(index_nrv,1) = index_var;
15.     else
16.         index_rv = index_rv+1;
17.         rv(index_rv,1)=index_var;
18.     end
19.end
20.if size(rv,1)< size(Properties,1)
21.     n_rv = flipud(sortrows(n_rv,1));
22.     for index_nrv2=1:size(n_rv,1)
23.         x(n_rv(index_nrv2),:) = [];
24.     end
25.end
26.dimensions = index_rv;
27.j=1;
28.while nF(Level+1)/N_Simulations_Level¡p
29.     Level = Level + 1;
30.     [y(Level,:),ind]=sort(y(Level,:),'descend');
31.    x(:,:,Level)=x(:,ind(:),Level);
32.    Y(Level)=(y(Level,nc)+y(Level,nc+1))/2
33.    z(:,:,1)=x(:,1:nc,Level);
```

```
34.   for j=1:nc
35.     for m=1:ns
36.       for k=1:dimensions
37.         a=z(k,j,m)+randn;
38.         r=min(1,normpdf(a)/normpdf(z(k,j,m)));
39.          if rand<r
40.           q(k)=a;
41.         else
42.           q(k)=z(k,j,m);
43.         end
44.       end
45.       if sum(q)>Y(Level)
46.         z(:,j,m+1)=q;
47.       else
48.         z(:,j,m+1)=z(:,j,m);
49.       end
50.     end
51.   end
52.   for j=1:nc
53.     for m=1:ns+1
54.       x(:,(j-1)*(ns+1)+m,Level+1)=z(:,j,m);
55.     end
56.   end
57.   if size(rv,1)<size(Properties,1)
58.     temp1 =[rv,x(:,:,Level+1)];
59.     for h=1:size(n_rv,1)
60.       temp2(h,:) = mean(dist(n_rv(h))).*(ones(1,N_Simulations_Level));
61.     end
62.     temp3 =[n_rv,temp2];
63.     temp4 = [temp1;temp3];
64.     temp5 =sortrows(temp4,1);
65.     temp5(:,1) = [];
66.   end
```

```
67.    clear z;
68.    nF(Level+1)=0;
69.    for i=1:N_Simulations_Level
70.      if size(rv,1)¡size(Properties,1)
71.        y(Level+1,i) = gfail(temp5(:,i));
72.      else
73.        y(Level+1,i) = gfail(x(:,i,Level+1));
74.      end
75.      if y(Level+1,i)>YF
76.        nF(Level+1)=nF(Level+1)+1;
77.      end
78.    end
79.end
80.pF_SS = pLevel*nF(Level+1)/N_Simulations_Level;
81.k = nF(Level+1)/N_Simulations_Level;
82.N=N_Simulations_Level+N_Simulations_Level*(1-p)*(Level);
83.fprintf('Using the Subset Simulation Method :\n');
84.fprintf('Total number of samples: %g\nNumber of Levels = %g\nFailure
probability = %g\n\n', N, Level, pF_SS)
85.end
```