CHALMERS
UNIVERSITY OF TECHNOLOGY

Application of Probabilistic Methods to Slope Stability

A Comparative Study Using Three Computational Tools: PEM, SLOPE/W, and a Custom Python Script for PLAXIS

Master's thesis in Infrastructure and Environmental Engineering

EMMA GUMMESSON & WILLIAM RYBERG

DEPARTMENT OF ARCHITECTURE AND CIVIL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026
www.chalmers.se

Master's thesis 2026

Application of Probabilistic Methods to Slope Stability

A Comparative Study Using Three Computational Tools: PEM, SLOPE/W, and a Custom Python Script for PLAXIS

EMMA GUMMESSON & WILLIAM RYBERG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Architecture and Civil Engineering
Division of Geology and Geotechnics
Chalmers University of Technology
Gothenburg, Sweden 2026

Application of Probabilistic Methods to Slope Stability
A Comparative Study Using Three Computational Tools: PEM, SLOPE/W, and a Custom Python Script for PLAXIS

EMMA GUMMESSON, WILLIAM RYBERG

Supervisor: Per Nylander, Sweco Sverige AB
Supervisor & Examiner :Ayman Abed, Associate Professor, Department of Architecture and Civil Engineering

Master's Thesis 2026
Department of Architecture and Civil Engineering
Division of Geology and Geotechnics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Study area of Nolhaga with the examined slope.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2026

Application of Probabilistic Methods to Slope Stability
A Comparative Study Using Three Computational Tools: PEM, SLOPE/W, and a Custom Python Script for PLAXIS
EMMA GUMMESSON, WILLIAM RYBERG
Department of Architecture and Civil Engineering
Chalmers University of Technology

Abstract

Slope stability calculations are predominantly performed using deterministic methods, where a single value is assigned to each soil parameter based on data from a limited number of boreholes. Since soil properties vary naturally, and assumptions must be made, uncertainties arise. Traditionally, these uncertainties are handled through partial factors and dimensioning must be based on minimum factors of safety defined in Eurocode. An alternative is to introduce a probabilistic approach, which yields a probability of failure. This can provide a more comprehensive picture of the problem and has the potential to enable more optimized designs.
In this thesis, three probabilistic methods are applied: the Point Estimate Method, Monte Carlo and Latin Hypercube simulations. The software programs SLOPE/W and PLAXIS have been used to apply these methods through limit equilibrium analysis and finite element modeling. Probability distributions are assigned to friction angle, unit weight, and effective cohesion, with the geometry derived from a slope in Nolhaga by the riverbank of Säveån. Since PLAXIS lacks built-in probabilistic functionality, a custom Python script was developed to automate the process.
The results show that all three methods produce comparable mean factors of safety, while differences in the estimated probability of failure are more pronounced. The choice of distribution has limited effect on the mean factor of safety but can influence the probability of failure by orders of magnitude. Spatial variability is found to have a significant impact on the results. Although the water level is identified as a key driver of slope stability, further development is required to introduce it as a varying input parameter, enabling it to be analyzed alongside the other parameters and improve the robustness of the results. The thesis concludes that probabilistic analysis offers valuable insight beyond deterministic methods, but requires careful consideration of input assumptions and the limitations of available tools.

Keywords: Slope stability, Probability, Monte Carlo, Latin hypercube, Point Estimate Method, Geotechnics, LEM, FEM, Statistics.

Acknowledgements

We would like to express our sincere gratitude to our supervisor and examiner, Ayman Abed, for guiding and supporting us through this challenging journey.

We also want to thank Sweco, and especially Per Nylander, for providing valuable industry insights and for granting us access to the necessary borehole data.

A very special thanks goes to Göran Sällfors for taking his time with us, patiently clearing up misconceptions, and for generously allowing us to utilize the PEM tool he developed. We would also like to thank Björn Anklew for giving us the opportunity to attend an industry course free of charge.

Our warmest thanks to our great friends in the master studio for turning long, exhausting days into fun and memorable moments. We are also grateful to our families for their endless support, encouragement, and for always believing in us.

Gothenburg, May 2026
Emma Gummesson & William Ryberg

Decleration of AI

AI was used in this master's thesis to check for spelling errors and to generally simplify sentence structure. It also served as a tool when writing the Python script and for generating code that produced figures.

Contents

List of Acronyms & Symbols **xi**

List of Figures **xiii**

List of Tables **xv**

1 Introduction **1**
1.1 Background . . . 1
1.2 Aim . . . 2
1.2.1 Objectives . . . 2
1.3 Limitations . . . 2
1.4 Societal, Ethical and Ecological . . . 3

2 Theoretical background **5**
2.1 Probability in Geotechnics . . . 5
2.2 Probability Functions . . . 6
2.2.1 Continuous Random Variables . . . 6
2.2.2 Normal Distribution . . . 7
2.2.3 Lognormal Distribution . . . 9
2.2.4 Triangular Distribution . . . 10
2.2.5 Uniform Distribution . . . 11
2.2.6 Generalized Spline . . . 12
2.3 Probabilistic Analysis Methods . . . 13
2.3.1 Monte Carlo . . . 13
2.3.2 Latin Hypercube (LHS) . . . 13
2.3.3 Point Estimate (PEM) . . . 14
2.3.4 Comparison of the Methods . . . 16
2.4 Limit Equilibrium Method . . . 17
2.5 Finite Element Method . . . 18

3 Methods **21**
3.1 Case Study . . . 21
3.1.1 Geotechnical Setting . . . 21
3.1.2 Geometry . . . 22
3.2 Derivation of Input Parameters . . . 23
3.2.1 Friction Angle . . . 24
3.2.2 Unit Weight . . . 24

3.2.3 Cohesion . . . 24
3.2.4 Piezometric Line . . . 25

4 Simulations and Modeling **27**
4.1 Input Parameters for Probabilistic Modeling . . . 27
4.2 Point Estimate Method . . . 28
4.2.1 Piezometric line . . . 29
4.3 SLOPE/W . . . 30
4.3.1 Probabilistic Analysis in SLOPE/W . . . 31
4.3.2 Spatial Variation and Correlation . . . 33
4.4 PLAXIS . . . 35
4.4.1 Probabilistic Analysis in PLAXIS . . . 35
4.5 Probability of failure . . . 37

5 Results **39**
5.1 Results from Point Estimate Method . . . 39
5.2 Results from SLOPE/W . . . 41
5.2.1 Spatial Variation . . . 44
5.3 Results from PLAXIS . . . 44
5.4 Comparison of the results . . . 47
5.5 Model uncertainty . . . 47

6 Discussion **49**
6.1 Distributions . . . 49
6.2 Point Estimate Method . . . 50
6.3 SLOPE/W . . . 50
6.4 Spatial Variability . . . 51
6.5 PLAXIS . . . 51
6.6 Reliability Assessment and Model Uncertainty . . . 53
6.7 Hydraulic Conditions . . . 53
6.8 Probability of Failure . . . 54

7 Conclusion **57**

Appendices **I**

List of Acronyms & Symbols

Acronym	Definition
CC-test	Clay Cutting Test
CDF	Cumulative Distribution Function
CONRAD	Software for evaluation of CPT data (SGI)
CPT	Cone Penetration Test
CPTu	Piezocone Penetration Test (with pore pressure measurement)
CRS-test	Constant Rate of Strain consolidation test
CV	Coefficient of Variation
FEM	Finite Element Method
FoS	Factor of Safety
HQ200	Design flood with a 200-year return period
KDE	Kernel Density Estimation
LEM	Limit Equilibrium Method
LH / LHS	Latin Hypercube / Latin Hypercube Sampling
LLW	Lowest Low Water level
MC	Monte Carlo simulation
MQ	Mean discharge / mean flow
ODF	Over Design Factor (Eurocode partial factor output in SLOPE/W)
PDF	Probability Density Function
PEM	Point Estimate Method
P_f	Probability of Failure
PM	Geotechnical Design Memorandum
P_{Msf}	Multiplier on Strength Reduction Factor (PLAXIS output)
RC1/RC2/RC3	Reliability Class 1 / 2 / 3 (EN 1990)
SGI	Swedish Geotechnical Institute
SRM	Strength Reduction Method
SS-EN	Swedish Standard – European Norm
TR Geo 13	Swedish Transport Administration technical guidelines
ULS	Ultimate Limit State

Symbol	Definition
β	Reliability index
c'	Effective cohesion
c_u	Undrained shear strength
γ	Unit weight
γ_{unsat}	Unsaturated unit weight
γ_{sat}	Saturated unit weight
μ	Mean value
φ'	Effective friction angle
σ	Standard deviation
q_c	Cone resistance (CPT)
q_t	Total cone resistance (CPT)
u_2	Pore pressure behind cone tip (CPT)

List of Figures

2.1 Example of the normal distribution with mean 5 and standard deviation 2. 8
2.2 Example of the standard normal distribution with mean 0 and standard deviation 1. 9
2.3 Example of a lognormal distribution with the location of mode, median, and mean. 10
2.4 Example of a triangular distribution. 11
2.5 Example of a uniform distribution with a = 2 and b= 14 in this case. 12
2.6 Example of stratification in LHS, with the areas A-H 14
2.7 The function $y = f(x)$ is evaluated at two points symmetrically placed around the mean value $\bar{x}$, one standard deviation σ_x in each direction. The resulting function values y_1^+ and y_1^- are then used to estimate the mean $\bar{y}$ and the standard deviation of the output variable y. *Figure reproduced with permission. Source: Göran Sällfors, Chalmers/GeoForce AB.* 15
2.8 Illustration of PEM applied to two random variables x_1 and x_2. The four points represents the evaluation points, where each variable is set one std above (+) or below (-) its mean value. The shaded region illustrated the estimated mean $\bar{y}$ of the output function $y = f(x_1, x_2)$ *source: [10] with permission* 15

3.1 Placement of the boreholes in the area 22
3.2 Section B-B geometry with sand overlaying clay 23

4.1 Final Stratification after CV reduction through additional layering . . 28
4.2 Example of normal distribution with cut off in the extremes min =-3.1, max=3.1. *The mean=0 refers to the offset of the mean being 0. 32
4.3 Example of Generalized spline distribution. 32
4.4 Schematic illustration of the slice properties along the slip surface. Left: slightly correlated properties (smooth variation between neighboring slices). Right: uncorrelated properties (independent random variation). 34

5.1 Results from the Point Estimate Method. 40
5.2 Convergence of simulations done in SLOPE/W: Monte Carlo and Latin Hypercube, 1000, 2000 & 10 000 runs (water level at +57.6 m) 41

5.3 Results from SLOPE/W using water level of +57.6 m 42
5.4 Results from SLOPE/W using water level of +58.2 m 42
5.5 Beta values for different sampling distances with three different geometries; slope with 3 layers, simple slope in clay, original slope with one sand layer. 44
5.6 Results from PLAXIS using 3 Layers and normal distribution (water level +57.6 m). 45
5.7 Correlation of the friction angle and unit weight with FoS in the third sand layer. 46
5.8 Input distribution for 3 sand layers in PLAXIS 46

List of Tables

3.1 Effective cohesion c' for clay (kPa) 25

4.1 Summary of input parameters for sand, clay, fill, and erosion protection 27
4.2 Friction angle, Sand layers 28
4.3 Combinations of water levels for PEM 30
4.4 Partial Factors – Eurocode 7 DA3 30
4.5 Summary of analyses conducted in SLOPE/W with LLW 33
4.6 Material properties for clay in the simple slope 35
4.7 Python libraries used in the Monte Carlo simulation framework. . . . 36
4.8 Recommended minimum reliability index β and corresponding failure probability P_f per reliability class according to EN 1990. 38

5.1 Combinations of +/- 1 standard deviation from the mean and the FoS calculated for these. 39
5.2 Calculation of Probability of Failure from the previously calculated FoS. 40
5.3 PEM results with the water level +57.6 m for "Unit wiegth & Friction anlge" and "3 Layers". 41
5.4 SLOPE/W results with water level +57.6 m. 43
5.5 SLOPE/W results with water level +58.2 m. 43
5.6 SLOPE/W results with water level +58.2 m, combined critical slip surface vs combined lowest beta index. 43
5.7 PLAXIS results with water level +57.6 m 45
5.8 Summary of β and P_f for PEM, SLOPE/W simulations, and PLAXIS simulations (water level at + 57.6 m). 47
5.9 Summary of the results from PEM, SLOPE/W, and PLAXIS including model uncertainty for normal distribution with one and three sand layers (water level at 57.6 m). 48

1
Introduction

This chapter is the foundation of this Master Thesis and includes background information, followed by the main aims and objectives highlighting the relevance of the work, and concludes with the defined limitations.

1.1 Background

When working with geotechnical assessments, it is still standard practice to use a deterministic approach, which uses a single value for each parameter to calculate a single factor of safety (FoS) [1]. Soil parameters are inherently variable due to the heterogeneity of the ground. Even more uncertainties are introduced in the interpretation of the data and the set up of the model. To account for this variability, a probabilistic approach can be employed. The variability in the soil can be accounted for using probabilistic distributions rather than fixed values of the soil parameters. The model uncertainty can also be taken into account through incorporation of the coefficient of variation (CV) associated with the model.

The regulatory framework governing geotechnical design in Sweden is TR Geo 13 from the Swedish Transport Administration which is adopted from Eurocode 7(EN 1997-1:2004). Two design methods are defined in this: the use of characteristic values with a minimum acceptable factor of safety against slope failure, or the partial factor method [2]. The latter is a semi-probabilistic approach, however the next generation of Eurocode 7 introduces a step towards a more fully probabilistic analysis as an equivalent alternative to the partial factor method for verification of limit states [3]. This highlights the relevance of probabilistic methods in practical geotechnical engineering and motivates the present study.

Software such as SLOPE/W provides built-in tools for probabilistic analysis, but the impact of different distribution types on the results has not been systematically investigated in practical slope stability analysis. Finite Element Method (FEM) software like PLAXIS 2D lacks an integrated automated function for probabilistic stability analysis. To be able to run several simulations in PLAXIS a custom python script is required to automate the generation of stochastic inputs and the processing of simulation outputs. Developing such a framework is essential not only to perform the analysis efficiently but also to allow for a direct comparison between different modeling environments and distribution models.

The municipality of Alingsås is planning a new kindergarten in the Nolhaga within the existing school grounds. Within the detail plan a natural slope is located with its foot in the river Säveån, Säveån has significant influence on the local geotechnical conditions. Ground investigations, including a number of cone penetrations tests (CPTs), show that the natural riverbank constis of loose sands overlying soft, silty clay layers. The current ground conditions require stability measures, and parts of the riverbank will be excavated and reinforced with erosion protection. The site investigation at Nolhaga has been conducted with a deterministic approach. Given the sensitive nature of the planned land use, there is a critical need to address the existing uncertainties. Consequently, a transition toward probabilistic stability analysis is necessary to explicitly quantify the probability of failure and ensure a robust safety assessment for the Nolhaga site.

1.2 Aim

The purpose of this master's thesis is to analyze and evaluate the slope stability using a probabilistic approach in limit equilibrium and finite element modeling of the Nolhaga riverbank. Further, the aim is to understand and compare the results of the output from different probabilistic tools including Point Estimate Method (PEM), SLOPE/W, and PLAXIS. This thesis also aims to make the probabilistic approach more understandable and practical.

1.2.1 Objectives

Following objectives will help to reach the defined aims of the Master thesis:

1. To understand statistical theory relevant to the probabilistic approach.
2. To understand the different probabilistic methods and their application in limit equilibrium and finite element modeling.
3. To build a realistic model in for both limit equilibrium analysis and finite element analysis, which reflects the real case of Nolhaga, considering the possible uncertainties.
4. To develop a Python code that automates processes to enable a large number of simulations to acquire a probability of slope failure.
5. To evaluate the Factor of Safety (FoS) and the Probability of Failure (P_f).
6. To compare the results the three probabilistic methods.

1.3 Limitations

The data from Nolhaga is limited to the already gathered data from existing boreholes and sections, meaning no more data will be collected. The possible erosion will not be modeled. Not all sections conducted by Sweco in the geotechnical report will be modeled and analyzed. The slope stability analysis will only consider the Ultimate Limit State (ULS). The serviceability limit states will not be investigated, hence deformations and displacements, are not considered. The modeling in the software PLAXIS and SLOPE/W are limited to 2D modeling. No distribution will

be assigned to the water level in the river, therefore the influence of the water level will not be evaluated in detail.

1.4 Societal, Ethical and Ecological

This Master Thesis contributes to the societal, ethical and ecological aspects by promoting transparency in risk assessment. A full disclosure of the uncertainties and variations in soil parameters are provided through evidence-based methods.
Regarding the societal and ethical aspects, the ability to quantify risk is essential when it comes to public safety. Especially in the case of Nolhaga, with the proximity of a planned kindergarten. Using a probabilistic approach instead of a deterministic value can ensure that decision-makers have a better understanding of the actual conditions of a site and thereby minimizing the risk of unforeseen events.
The ecological concerns are addressed indirectly in this thesis, as a potential slope failure along the riverbank of Säve river could lead to sediment transport and contamination of the aquatic ecosystem. Accurate stability measures and appropriately designed mitigation measures are important for structural safety and for the protection of local biodiversity and the maintenance of ecological balance.

2

Theoretical background

In this chapter, the fundamental background of the Thesis is presented. First, the concepts of probability in geotechnical engineering are introduced, followed by an explanation of five probabilistic functions and the fundamental concepts of statistics. Three probabilistic analysis methods are presented, these include Monte Carlo simulation, Latin Hypercube sampling, and the Point Estimate Method. Lastly, the chapter provides an introduction to the limit equilibrium method and the finite element method, which form the numerical basis for the analyses conducted in this study.

2.1 Probability in Geotechnics

The soil that makes up our surroundings is created through natural geological processes. Compared to other man-made materials like steel or concrete, the variations in soil are larger. This variability arises not only from the natural randomness of soil properties, such as spatial variation, but also from uncertainty due to the lack of knowledge. This includes measurement uncertainty and statistical uncertainty resulting from limited data availability [1].
These uncertainties can be captured through random variables and mathematical models in the form of probability distributions. Although deterministic approaches are more commonly used in geotechnical engineering, probabilistic approach may provide a more adequate description of reality. For the deterministic approach, input parameters are assigned a single value, resulting in one factor of safety. The result is failure or not failure. In contrast, the probabilistic approach introduces the natural variability and randomness of soil properties. The assessment is based on the probability of failure rather than a strictly safe or unsafe condition, which is valuable for complex projects associated with high safety risks.

Engineering is always a balance between costs safety, anyone can build a bridge but it takes experience and deep knowledge to build a bridge as cheap as possible while still standing. This probabilistic approach is not meant to replace traditional geotechnical engineering but to provide a method used in other engineering applications and to explain the fundamental theories behind it to ensure it is implemented in a safe way.
This chapter introduces the fundamentals of probability theory and random variables, with a focus on continuous random variables. Furthermore, probability distributions commonly used to describe input parameters along with numerical methods

applied in probabilistic analysis, are presented.

2.2 Probability Functions

To understand probability functions, it is important to introduce some fundamental concepts of statistical theory. In this section, random variables and distributions are introduced. Random variables are used to represent a collection of possible events in numerical terms. Griffiths & Fenton provides the following definition [1]:

Let S be a sample space with outcomes $\{s_1, s_2, \dots\}$. A random variable X is a function

$$X : S \to \mathbb{R}$$

that assigns a real number $X(s)$ to each outcome $s \in S$.
For every outcome s there is exactly one value of $x = X(s)$, however different values of s (outcomes) can give the same value to x. This formulation makes it possible to describe random events with mathematical methods. Once an experiment is conducted and the outcome is known, then the result is referred to using the lower case x_i. The random variable X represents the unknown quantity, however we can define the different possibilities for it to take on certain values.

A basic understanding of probability distributions includes measures describing the location and spread of a distribution. These measures includes the mean, median, mode, variance, standard deviation, and the coefficient of variation. The first three mentioned describe the location while the latter describes the spread of a probability distribution:

- **Mean** represents the average value of the distribution.
- **Median** is the value that divides the distribution into two equal parts:
- **Mode** is the most likely value of the distribution.
- **Variance** is how widely the values spread around the mean.
- **Standard deviation** is the square root of the variance.
- **Coefficient of variation (CV)** is the ratio between the standard deviation and the mean.
- **Reliability index (β-value)** The number of standard deviations the mean is from the specific value (1.0 in the case of slope stability).

2.2.1 Continuous Random Variables

Continuous random variables can take on an infinite number of possible outcomes in one or several intervals on the real line [1]. The probability that a continuous random variable is exactly one value is zero and that is why they are defined over an interval of values and not by certain points. The probability is therefore defined using relative likelihoods:

The probability that X is in the interval $[x, x + dx]$ is given by $f_x(x)dx$, or: $P[x < X \leq x + dx] = f_x(x)dx$ where $f_x(x)$ is the probability density function (PDF) of the

random variable.

In other words the probability that X lies within the interval is the area under the density function. The density function is always non-negative and have a total area of 1. That is why continuous random variables often are used to model continuous varying units such as time, length and load.

Another way of expressing the probabilities associated with a continuous random variable is through the cumulative distribution function defined by Griffiths & Fenton as [1]:
The cumulative distribution function, Fx(x) of a continuous random variable X having probability density function fx(x), is defined by the area under the density function to the left of x

$$F_X(x) = \mathbb{P}[X \leq x] = \int_{-\infty}^{x} f_X(t)\, dt \tag{2.1}$$

2.2.2 Normal Distribution

A normal random variable is a widely used model for a continuous measurement [4]. Many natural phenomena occurs as the sum of many small contributions and according to De Moivre's central limit theorem, these small sums tend to normal distribution [1]. One geotechnical example is cohesion in soil which occurs through several small micro-interactions, and hence is often modeled as normally distributed. Montgomery & Runger provides the following mathematical definition of the formula for normal probability density functions [4]:

A random variable X with probability density function

$$f_X(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \qquad -\infty < x < \infty \tag{2.2}$$

is a normal random variable with parameters μ, where $-\infty < \mu < \infty$, and $\sigma > 0$. Also,

$$E(X) = \mu \qquad \text{and} \qquad \mathrm{Var}(X) = \sigma^2. \tag{2.3}$$

The notation $N(\mu, \sigma^2)$ is used to denote the distribution.

The mean and standard deviation are therefore important parameters. The mean $E(X) = \mu$ determines the center of the probability density function. The variance, denoted $Var(X) = \sigma^2$ measures the spread of the distribution and the standard deviation σ determines the width of the curve.
This function produces the typical appearance of the graph, which also show the property that the distribution is symmetrical about the mean (mean=median), see figure 2.1.

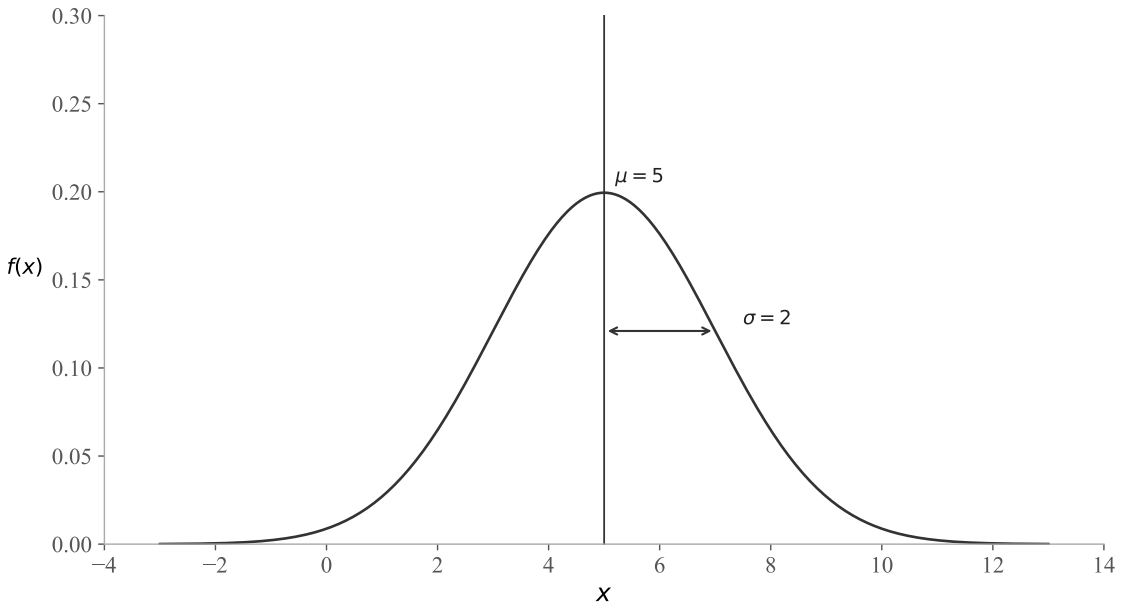


Figure 2.1: Example of the normal distribution with mean 5 and standard deviation 2.

Two other important properties are that the maximum point, or mode, of the distribution occurs at the mean and that the inflection points of $f(x)$ occur at $x = \mu \pm \sigma$ [1].
For a normally distributed variable, 68% of the observations lie within on standard deviation of the mean, about 95% within two standard deviations, and about 99.7% within three deviations. Meaning the probability density function decreases as x moves farther away from the mean, and the probability of observations occurring beyond 3σ from the mean is very small.
Since there is no closed form solution for the integral of the normal probability density function, probabilities need to be determined with numerical integration [1]. To avoid having several different tables for each mean and standard deviation, the random variable X can be standardized with the equation 2.4.

$$Z = \frac{X - \mu}{\sigma} \tag{2.4}$$

This results in a standard normal random variable with the mean equal to zero and unit variance, see figure 2.2. For this standardized distribution, a precomputed table is available that provides cumulative probabilities up to a specified value. Consequently, probabilities associated with any normally distributed random variables can be obtained by evaluating the cumulative distribution function of the standard normal distribution.

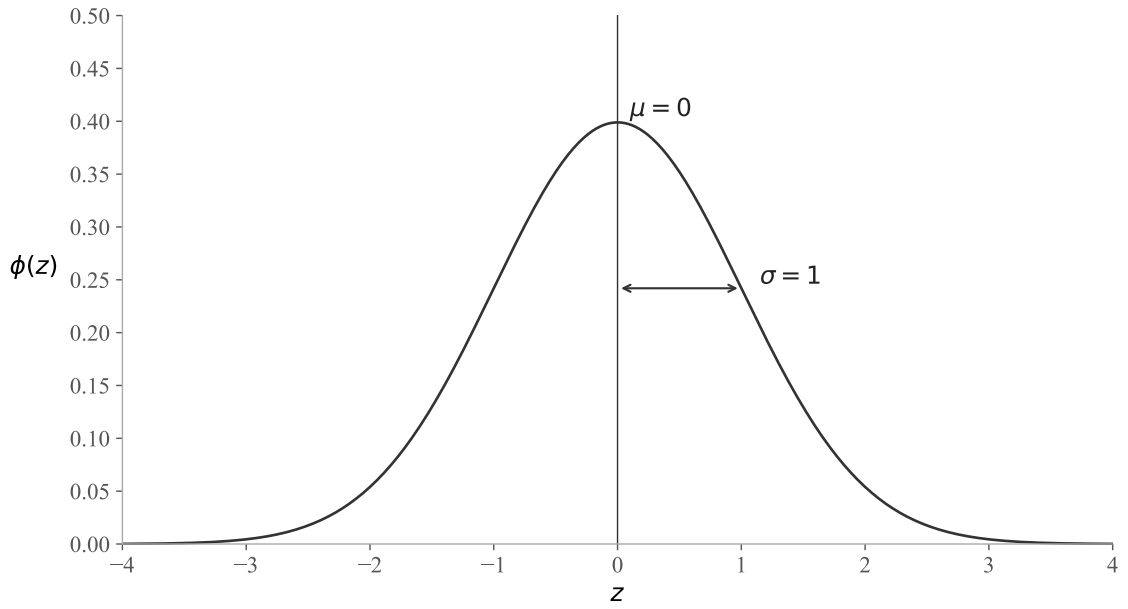


Figure 2.2: Example of the standard normal distribution with mean 0 and standard deviation 1.

The standard normal table allow probabilities for all normally distributed variables to be evaluated after standardization.
The standard normal distribution is given by the symbol $\phi(z)$ and defined by [1]:

$$\phi(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}z^2}, \qquad -\infty < z < \infty \tag{2.5}$$

Although the normal distribution is very powerful, it may not always be the most appropriate choice for variables that are strictly non-negative, such as certain material properties and loads. This introduces the use of the lognormal distribution.

2.2.3 Lognormal Distribution

A lognormal distribution is applied when the distribution cannot allow negative values [1]. One example is the elastic modulus of any soil, which cannot be negative hence the normal distribution cannot be its true distribution since it allows negative values for X. The lognormal distribution arises from the normal distribution through a simple, non-linear transformation:

If G is a normally distributed random variable with the domain $-\infty < g < \infty$ then $X = exp(G)$ will have the domain $0 < x < \infty$. In this case the distribution of G is lognormal.

So, the probabilities are computed from the transform of the normal distribution, where the standard normal table can be used since $ln(x)$ is normally distributed. The transformation allows for the lognormal distribution to be evaluated using the standard normal distribution.

Montgomery & Runger defines the probability density function as follows:

Let W have a normal distribution with mean θ and variance ω^2. Then $X = \exp(W)$ is a lognormal random variable with probability density function

$$f(x) = \frac{1}{x\,\omega\sqrt{2\pi}} \exp\left[-\frac{(\ln(x) - \theta)^2}{2\omega^2}\right], \quad 0 < x < \infty$$

The mean and variance of X are

$$E(X) = e^{\theta+\omega^2/2} \qquad \text{and} \qquad V(X) = e^{2\theta+\omega^2}\left(e^{\omega^2} - 1\right)$$

In the logarithmic domain, the mean and standard deviation is noted as θ and ω^2. It should be noted that this distribution is strictly positive and the mean and variance are derived from the normally distributed random variable.
Figure 2.3 represents a common appearance of the lognormal distribution curve.

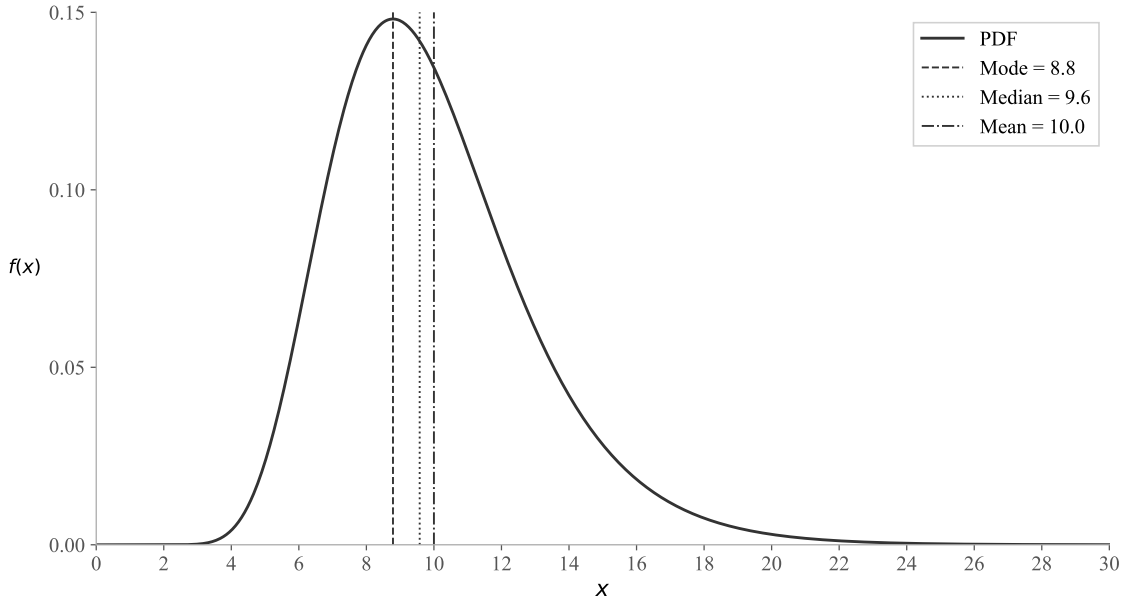


Figure 2.3: Example of a lognormal distribution with the location of mode, median, and mean.

One important characteristic of the lognormal distribution is the positive skewness where the *mode* < *median* < *mean* [1]. Most values are relatively small, but occasionally larger values may occur which can significantly increase the mean and make it less representable of the data. The median divides the distribution into equal parts and is therefore often considered a more representative measure compared with the mean.

2.2.4 Triangular Distribution

Triangular distribution is a simpler form of distribution where the lowest-, highest- and most common value (mode) are used as the points in a triangle (see figure 2.4).

The principle for the distribution is the same as normal and lognormal distribution but is more intuitive [5].

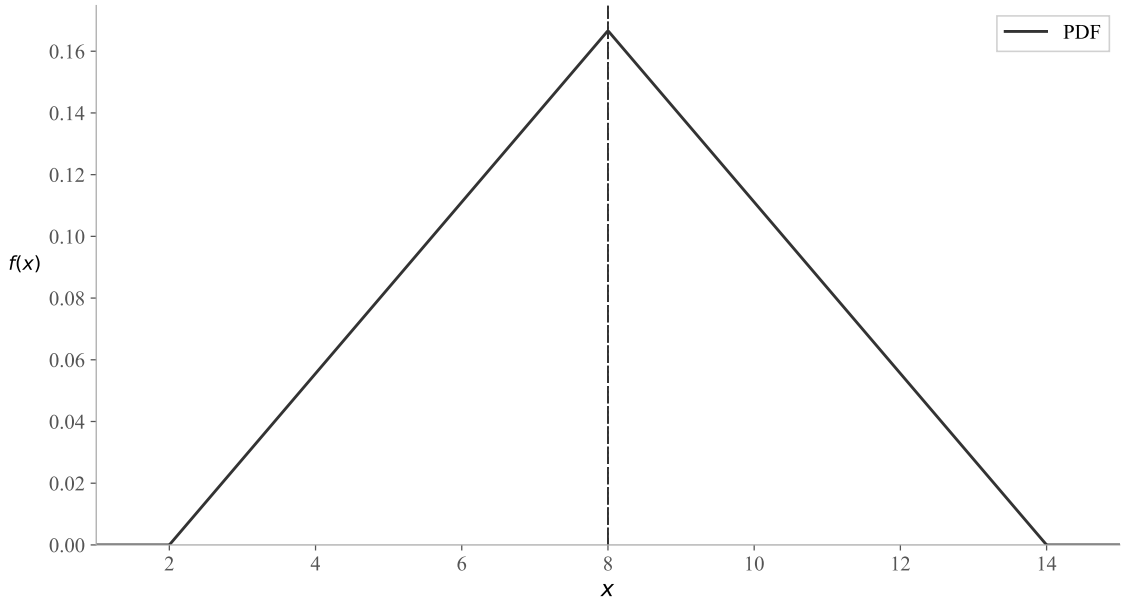


Figure 2.4: Example of a triangular distribution.

The results are not as accurate, but for smaller sample sizes it can be a good substitution to more complicated distribution curves.

$$f(x) = \begin{cases} \frac{2(x-a)}{(b-a)(m-a)} & \text{for } a \leq x \leq m \\ \frac{2(b-x)}{(b-a)(b-m)} & \text{for } m < x \leq b \end{cases} \tag{2.6}$$

where:

a = the minimum value (lower limit)
b = the maximum value (upper limit)
m = the most likely value (mode)

2.2.5 Uniform Distribution

This is the simplest form of distribution, where all possible outcomes have the same probability of occurring. Instead of bell curve, the PDF is a rectangle with height F(x) and width the length represented by the span (a-b) of the probability (see figure 2.5 . The mean of the distribution is simply the half way between the end points and are defined by equation 2.7 and 2.8.

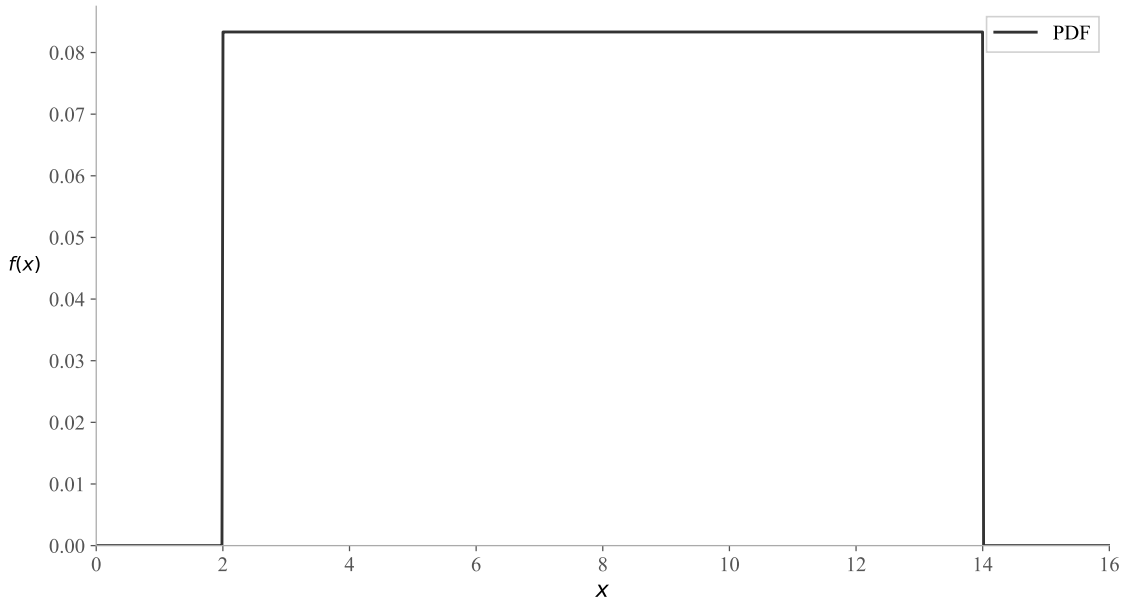


Figure 2.5: Example of a uniform distribution with a = 2 and b= 14 in this case.

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{for } a \leq x \leq b \\ 0 & \text{for } x < a \text{ or } x > b \end{cases} \tag{2.7}$$

$$\sigma = \frac{b-a}{\sqrt{12}} \tag{2.8}$$

This method is used when there is extreme uncertainty in a well defined domain [1], this makes it unsuitable in natural applications. However, it the basis of many of the approximations of more complex distributions presented in the Generalized Spline section.

2.2.6 Generalized Spline

Even though the distribution can be seen as normal or lognormal for large sample sizes (from multiple sites) this can be an oversimplification when deriving parameters for a specific site [6]. If the data do not correspond to any of the standard probability density functions (PDFs), a custom one can be constructed using the generalized spline function. Mixture models can be used to optimize components of the data that is collected. This can identify multiple normal distribution within the data and sequentially and capture multiple peaks (or sub-means) later when the Monte Carlo simulation are done these peaks will be calculated the most frequently, as these are the likely to occur.
The mixture model is a method of processing data to extract a more correct spline, this is done through optimization. Firstly, the raw data will be analyzed to see if there are peaks that could correspond to certain weak layers in the soil. The data is then "cleaned" and a continuous line is created that will be used in SLOPE/W.

2.3 Probabilistic Analysis Methods

To convert the defined probability density functions to input parameters that the probabilistic calculations are based on. This can be done in multiple ways depending on the computational capacity, number of variables and the tails of the distribution. This chapter will introduce three common methods used in statistical analysis and risk assessment to generate a large number of simulations that can be interpreted later.

2.3.1 Monte Carlo

Monte Carlo is one of the first methods developed and is utilized in many engineering problems, from aerospace engineering to the population of ants. This is done with the law of large numbers, i.e. the results are accurate when a infinite number of calculations have been conducted. This is not physically possible, instead a large amount of simulations are done, typically more than N = 10.000 but it can be much higher depending on the required accuracy. Input parameters are grouped to form random constellation of these is denoted as N. When the calculation has been conducted the error is correlated to the number of samples conducted and the standard deviation and inversely proportional to the square root of N [7]. The following Equation 2.9 illustrates how the factor of safety is calculated.

$$\hat{f}_N(x) = \frac{1}{N} \sum_{i=1}^{N} F(x, (\phi^i, c^i, \gamma^i)) \tag{2.9}$$

where:

- $\hat{f}_N(x)$ is the Monte Carlo estimate of the quantity of interest (e.g. the factor of safety) based on N simulations.
- N is the number of random samples (simulations) drawn.
- $F(\cdot)$ is the deterministic model function (the slope stability model) evaluated for a given set of input values.
- x denotes the fixed (deterministic) model inputs that do not vary between simulations, such as geometry and boundary conditions.
- ϕ^i is the randomly sampled friction angle in simulation i.
- c^i is the randomly sampled effective cohesion in simulation i.
- γ^i is the randomly sampled unit weight in simulation i.
- i is the simulation index, running from 1 to N.

In each simulation, one value is drawn for ϕ^i, c^i and γ^i from their respective probability distributions, and the model F is evaluated. The estimate $\hat{f}_N(x)$ is the average over all N evaluations, which converges to the true expected value as $N \to \infty$ in accordance with the law of large numbers.

2.3.2 Latin Hypercube (LHS)

Latin hypercube is a more optimized method when selecting random numbers, it divides the interval into sections with equal probability as seen in 2.6. After the sections have been established random numbers are extracted from each section.

This ensures that variables from the extremes of the distribution are included, with the standard MC method a larger amount of variables has to be generated to make sure the whole distribution is represented. With the stratification of the distribution a more efficient pairing of variables can be achieved.

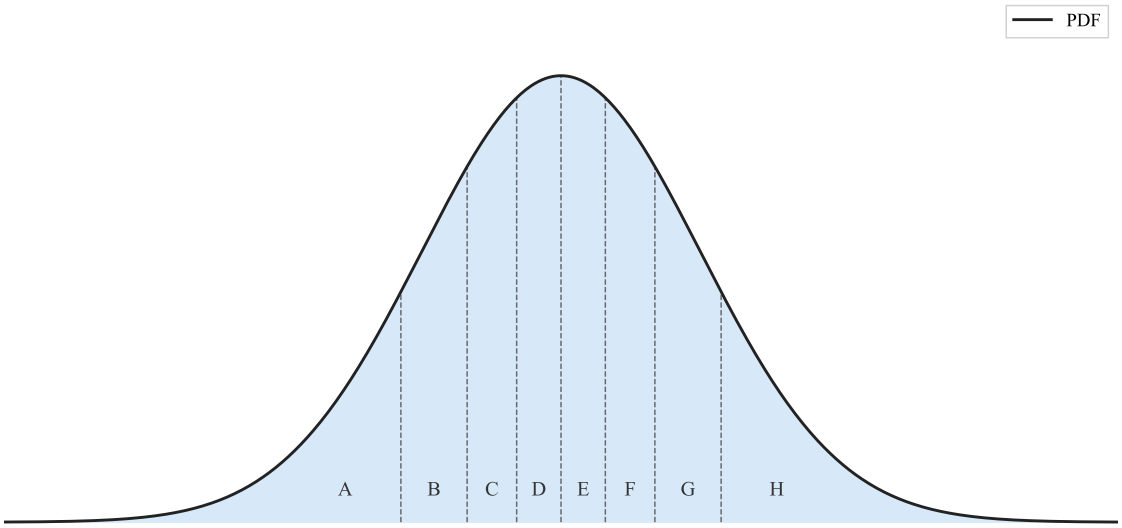


Figure 2.6: Example of stratification in LHS, with the areas A-H

The software shuffles these lists and draws one value from each to form a unique simulation scenario, repeating the procedure until all sections of all variables have been sampled. This pairing method can be optimized in software such as SLOPE/W by changing the c - ϕ correlation (ranging from -1 to 1). This correlation provides the correlation between cohesion and friction angle,is usually negatively correlated [8]. This means that if a high friction angle is sampled it will be more likely a lower value for cohesion will be sampled, and the other way around.

2.3.3 Point Estimate (PEM)

The Point Estimate Method was first introduced by Rosenbluth in 1975 [9]. It is a simple method in which each random variable is defined by two discrete values instead of a full probability distribution, meaning that it does not requires extensive knowledge of probability theory [7].

The calculation method aims to address the question "what happens to the model response if this specific parameter was larger or smaller than its mean value?" [9]. The mathematical background behind PEM is to replace a continuous random variable X with a discrete random variable defined by a small number of carefully chosen representative values. For a random variable X with the mean μ_X and standard deviation σ_X, two representative points replaces the continuous distribution. These points are selected symmetrically around the mean, one higher and one lower, given by $x^{(+)} = \mu_X + \sigma_X$ and $x^{(-)} = \mu_X - \sigma_X$ (see fig. 2.7). Both the central tendency

and variability of the variable are thereby captured.

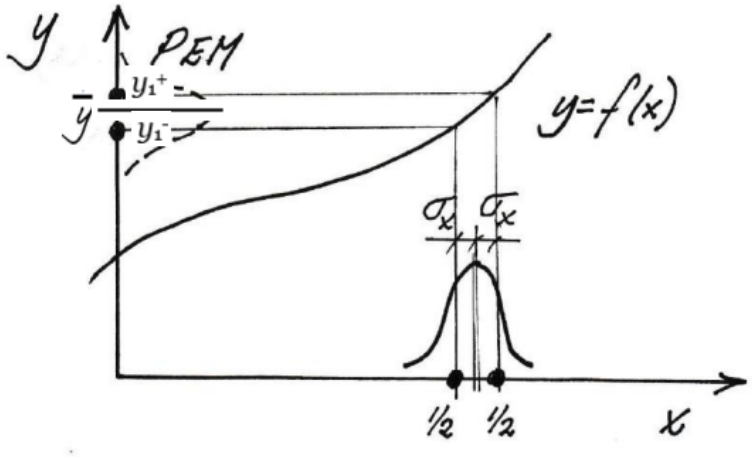


Figure 2.7: The function $y = f(x)$ is evaluated at two points symmetrically placed around the mean value $\bar{x}$, one standard deviation σ_x in each direction. The resulting function values y_1^+ and y_1^- are then used to estimate the mean $\bar{y}$ and the standard deviation of the output variable y. *Figure reproduced with permission. Source: Göran Sällfors, Chalmers/GeoForce AB.*

The number of calculations depends on the number of random variables included in analysis. In the original form of PEM, all possible combinations of the representative values are evaluated, meaning that N number of random variables require 2^N number of calculations. Consequently, this number can become very large when there are a large number of random variables. Figure 2.8 helps describe the output response for more than one variable as input around their mean values.

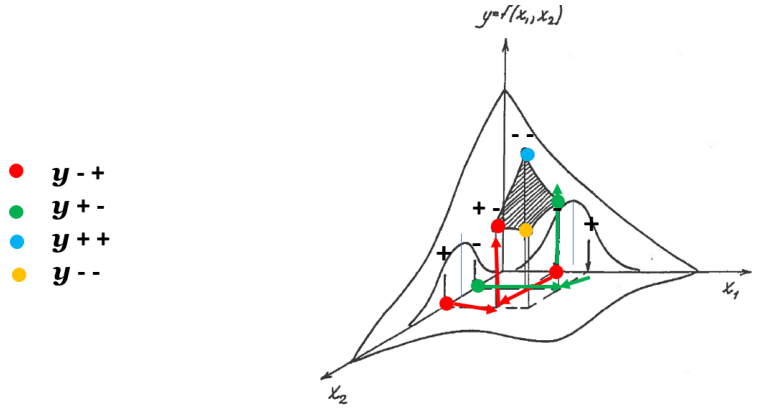


Figure 2.8: Illustration of PEM applied to two random variables x_1 and x_2. The four points represents the evaluation points, where each variable is set one std above (+) or below (-) its mean value. The shaded region illustrated the estimated mean $\bar{y}$ of the output function $y = f(x_1, x_2)$ *source: [10] with permission*

The results from PEM is an estimate of the mean value and standard deviation of the output random variable which can be calculated according to equation 2.10.

$$\sigma_y = \left[E(y^2) - (E(y))^2\right]^{0.5} \tag{2.10}$$

$E(y)$ is the expected value and can for a function of $y = f(x_1, f_2)$ (where x_1 and x_2 are random variables be calculated as [10]:

$$E[y] = p_{++}y_{++} + p_{+-}y_{+-} + p_{-+}y_{-+} + p_{--}y_{--} \tag{2.11}$$

where y_{++} are function values calculated for values of the parameters x_1 and x_2 as follows:

$$y_{++} = f(x_1 + \sigma[x_1], x_2 + \sigma[x_2]) \tag{2.12}$$
$$y_{+-} = f(x_1 + \sigma[x_1], x_2 - \sigma[x_2]) \tag{2.13}$$
$$y_{-+} = f(x_1 - \sigma[x_1], x_2 + \sigma[x_2]) \tag{2.14}$$
$$y_{--} = f(x_1 - \sigma[x_1], x_2 - \sigma[x_2]) \tag{2.15}$$

$$p_{++} = p_{--} = \frac{1+\rho}{4} \tag{2.16}$$
$$p_{+-} = p_{-+} = \frac{1-\rho}{4} \tag{2.17}$$

where ρ is the correlation coefficient for x_1 and x_2. If the two parameters x_1 and x_2 are uncorrelated, $\rho = 0$ and $p_{++} = p_{+-} = p_{-+} = p_{--} = 1/4$.
However no information about the type or appearance of the distribution is given, which instead must be assumed. In this context, extra caution should be taken when assuming normal distribution as it allows negative values which may be physically unrealistic for certain parameters.

The Point Estimate method is preferable to use for problems where the correlation between the input and output is close to linear, where the random variables are not strongly connected, and where the mean value and the variability of the response are of greater interest than the extremes.

2.3.4 Comparison of the Methods

Even though Monte Carlo simulations, Latin Hypercube sampling and Point Estimate Method all are probabilistic approaches used to account for uncertainty in input parameters, they have some differences. The main difference is in terms of sampling strategy, computational effort and the type of information they provide about the model response.

Monte Carlo simulations are, as previously mentioned, the most general and widely used method. Repeating random sampling from the probability distributions of the

input parameters, it generates a distribution of the results which is the probability of failure. This method requires a large number of simulations to achieve accurate results to ensure that the extreme values are included in the sampling. This is computationally expensive and time-consuming, especially in case more complex numerical models such as finite element analyses.

In contrast the Latin Hypercube sampling is more efficient, less time-consuming, and can be regarded as a extension of Monte Carlo simulation. The output is a distribution similarly to Monte Carlo, but instead of drawing samples at random, the range of each input variable is divided into intervals of equal probability from which one value is sampled. The entire input space is more evenly covered for the combination of samples and this generally leads to better estimates of the mean and variance of the output compared to the Monte Carlo simulations. However some examples show that this difference is not significantly large which Baecher et al. argues in their report [7].

The Point Estimate Methods follows another approach compared to the previously mentioned methods. Instead of generating a large number of random samples, PEM replaces each random variable with small number of representative discrete values. Evaluation of the model is then done for all combinations of these representative values. In contrast to MC and LH, the model generates a mean value and a standard deviation as an output and the distribution is assumed afterwards. As a result PEM requires significantly fewer model evaluations and is computationally efficient and regarded as very simple. However, as the number of random variables increases the number of required evaluations increases very fast since it is an exponential relationship. This may limit the applicability of PEM in problems with many uncertain parameters. PEM is more suitable for models with small variations in the input parameters and without extreme non-linear relationships. A consequence of assuming the distribution is the additional source of uncertainty, as the true output distribution may deviate from the assumed one. This becomes significant when the tails of the curve (the extremes) are of interest, which risk being underestimated.

To conclude, the choice of which method to apply depends on the knowledge of the problem, its components and the purpose of the analysis. When a detailed description of the output or an estimate of failure probability is required, Monte Carlo simulation may be more suitable. Similar information can be retrieved from Latin Hypercube sampling, but with improved efficiency and can be suitable when computational resources are limited. For more preliminary analyses where the mean and variability of the response are of interest, the Point Estimate Method can be the most efficient with minimal computational effort.

2.4 Limit Equilibrium Method

Limit equilibrium method (LEM) is one of the oldest and most widely used numerical analysis techniques in geotechnical engineering [8]. It is based on the idea of discretizing a potential sliding mass into a number of vertical slices. For each slice,

equilibrium conditions are formulated, and the global stability of the slope is expressed through a factor of safety (FoS). The FoS is defined as the ratio of available shear strength to the shear stress required for equilibrium along the assumed slip surface.
One of the most well-established software tools for slope stability analysis based on limit equilibrium method is SLOPE/W, which is included in the Geostudio suite. SLOPE/W enables more complexity in the analysis, involving complex stratigraphy, variable pore-pressure conditions, non-linear and linear shear strength models. The software identifies the most critical slip surface and the corresponding minimum factor of safety.

In SLOPE/W there are various solution techniques for computing the factors of safety, but these are in general very similar [8]. Earlier methods includes the Ordinary, Bishop's simplified and Janbu's simplified method which all were developed with simplified assumption to allow hand calculations [8]. These methods neglect interslice shear forces and satisfy only selected equilibrium conditions, which reduces the computational complexity but can limit the accuracy in certain cases. More advanced methods such as Spencer and Morgenstern-Price, satisfy both overall force equilibrium and moment equilibrium. These methods can account for interslice forces and require an iterative solution procedure.

The LEM formulation assumes a single global factor of safety and both the cohesive and frictional components of shear strength are reduced by the same factor. The slip surface may be circular, composite, or defined by a series of straight-line segments [11].

2.5 Finite Element Method

PLAXIS is a software for finite element modeling. It works by dividing complex problems into finite number of elements with simple geometry and known properties. The elements are assigned nodes that have degrees of freedom [12]. The number of degrees of freedom depends on the specific application and the required accuracy. The simplicity of each of the elements enables complex continuous behavior to be incorporated into discrete calculations that can be efficiently performed using matrices. The computation is largely dependent on the number of elements and nodes. To reduce computational load, smaller elements are applied only in critical areas (refined mesh) [13]. For less critical parts of the model larger elements (coarse mesh) can be applied. To ensure that the mesh resolution it is important to check that the results converge as when refining the mesh.
FEM is advantageous as no assumptions are made about the shape or location of the slip surface [14], which allows the identification of more complex failure mechanisms.

The Factor of Safety (FoS) can be determined by incrementally increasing the unit weight, called gravity loading. Another option for calculating FoS is to reduce strength parameters (phi/c) reduction. This iterative process continues until the

numerical solution fails to converge, at which point physical failure is assumed. Boundary conditions are a critical part of the calculation, these include initial flow- and loading conditions.

3
Methods

This chapter introduces the the case study of Nolhaga. First, the geotechnical setting is introduced, followed by the derivation of input parameters from borehole data. The Chapter forms the basis for the later conducted simulations using different probabilistic tools, with the aim of evaluating the probability of failure.

3.1 Case Study

The planned kindergarten will accommodate 120 children, it is located in the western part of Alingsås a small municipality located 42 km north west of Gothenburg. A comprehensive feasibility study was carried out, including traffic impact assessment, bird inventory, and geotechnical site investigation, among other studies. The analyzes were done in a deterministic way but for several different cases including Rapid draw down, low water levels in the river and drained/undrained conditions. The original Geotechnical Design Memorandum (PM) includes six sections (A-A to H-H) of which only H-H is stable without any stabilizing measures.

3.1.1 Geotechnical Setting

The area is located below the highest coastline, facilitating sedimentation of finer particles that form the clay present today. The dynamics of the sedimentation process is complicated. In the area signs of stratification were detected with piston sampling at Nolhalla ice rink, with silty layers[15] [16]. The cone penetration tests (CPTus) show similar trends at the investigated riverbank. The stratification complicates the parameter determination.

The geotechnical investigations have been conducted in multiple phases from 2015 to 2025. The geotechnical data report is based on 17 CPTus, of which one was conducted in 2025 in close proximity to section D-D. Additional undisturbed test was collected at 20, 22 and 26 meters depth at section D-D, close to the existing bridge. These were analyzed with CRS-test, CC-test and direct shear test, to obtain additional confidence in the data.

The section B-B was chosen for the analysis of Nolhaga (see figure 3.1). It was considered as a good representation of the slope and further two boreholes (SW2103 and SW2104) lies directly this section. Together with the two boreholes, six other boreholes were used for the input parameters in the analysis, these being, SW2001,

SW2002, SW2101, SW2125, SW2105, and SW2106.

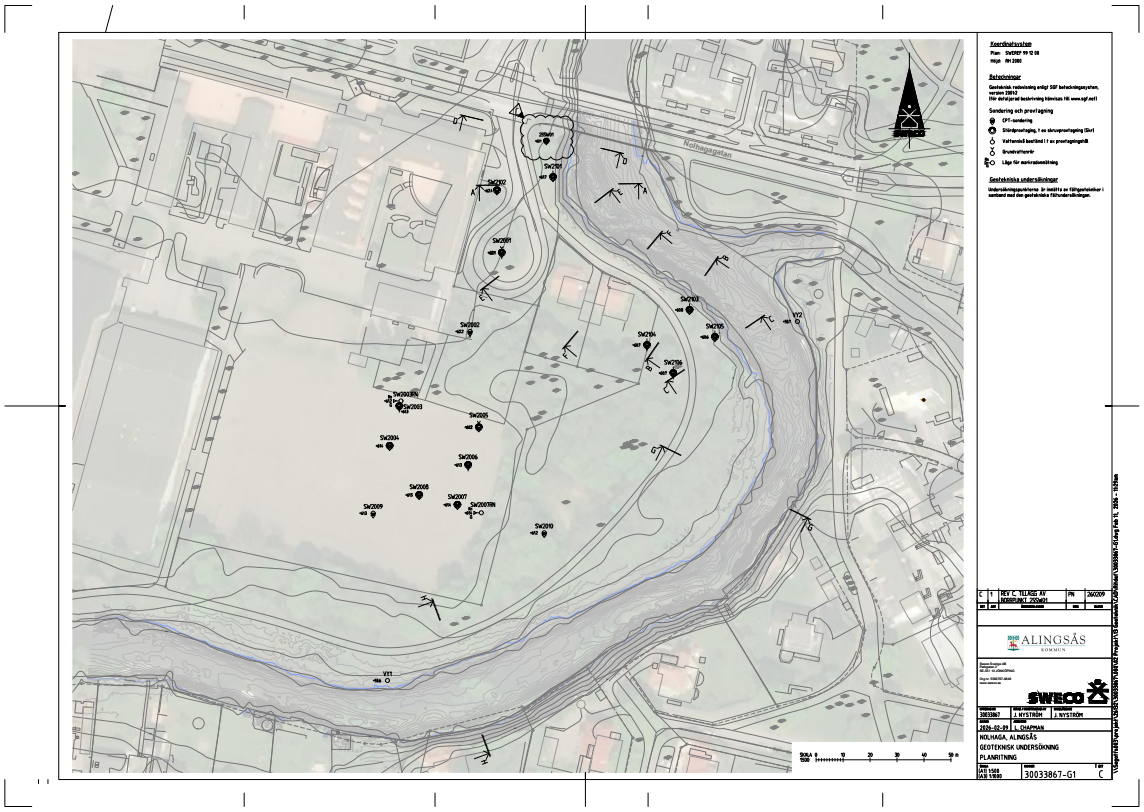


Figure 3.1: Placement of the boreholes in the area

3.1.2 Geometry

To ensure that the Factor of Safety (FoS) is directly comparable to the values calculated by Sweco, the geometry was replicated. Since the parameters were the focus of the comparison, the geometric layout follows the original model as closely as possible and is presented in Figure 3.2. First, a more simple model was analyzed where only two soil layers were modeled (sand and clay). Later analyzes were based on a more complex geometry determined from the Coefficient of Variation (CV) using the N-Sigma-Rule method. In this case 1.65-sigma ($\mu \pm 1.65\sigma$) was used, meaning that 90% of data is captured within the confidence interval. Further discussion is provided in Chapter 4.1.

To reduce the CV, the sand layer was divided into sublayers. Prior to this adjustment, the CV was 8%, whereas a value of approximately 3–4% is expected for these conditions in Western Sweden [17]. While the layer could theoretically be subdivided indefinitely, three sublayers were chosen as a reasonable balance between reducing variability and maintaining practicality. The layering was optimized in Excel by comparing four unique layer combinations, evaluating the standard deviation and mean of each according to (3.1).

$$\mathrm{CV} = \frac{\sigma}{\mu} \tag{3.1}$$

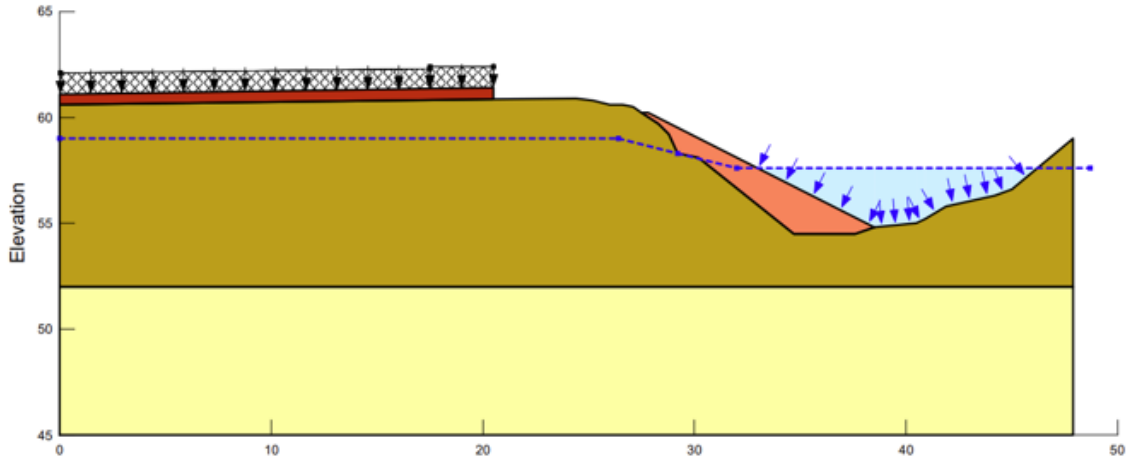


Figure 3.2: Section B-B geometry with sand overlaying clay

3.2 Derivation of Input Parameters

All input data for the analyses conducted, were derived from borehole data provided by Sweco. The data were extracted from the program CONRAD, which is a program from SGI to evaluate and present results from Cone penetration tests. CONRAD evaluates parameters based on the guidelines from SGI information Nr. 15 rev. 2007 [18].

The values of each parameter (unit weight, friction angle, cohesion) were specified for every depth interval of 0.5 m. To enable comparison between the boreholes, the depth measurements were converted into elevations using the reference level for each borehole.

By dividing the values into friction material and clay, the data could be compiled to determine the minimum, maximum, and mean values, as well as the standard deviation. Both the friction layer and the clay layer were initially assumed to have uniform parameter values throughout the entire layer. However, the friction layer were subsequently be divided into three sub-layers to account for variation with depth and thereby reduce variance.

From this, histograms were generated to provide a visual representation of the data distribution. The histograms have a limited number of bins which allows a sufficient number of data in each interval for the observed frequencies to vary smoothly and still provides adequate definition to the shape of the distribution. In this way, the risk of creating noise is avoided which would make it more difficult to recognize the overall distribution. Beacher and Christian argues that a suitable rule of thumb for choosing the number of intervals looking at smaller data sets is the following equation 3.2:

$$k = 1 + 3.3 \log_{10} n \tag{3.2}$$

where k is the number of intervals and n is the number of data values [7].

These histograms were also used to derive values for a generalized spline distribution, which was utilized as input data for SLOPE/W. The generalized spline function requires an offset from the mean, which was calculated by subtracting the overall mean of the data from the mean value of each interval. The probability was then determined by dividing the number of data points within each interval by the total number of data points. From these the offset and their associated probabilities were used to construct a distribution function.

3.2.1 Friction Angle

To determine the friction angle, the cone resistance from each borehole was used through the following relationship in Equation 3.3 according to TR Geo 13 [2]:

$$\phi' = 29 + 2.8 \cdot q_c^{0.45} \leq 42^\circ \tag{3.3}$$

Due to late access to CONRAD, values for q_c had to be extracted from the reported values of q_t and the pore pressure from the CPT-data of each borehole, which were found in the appendices of the site investigation report [19]. Values for each parameter were read graphically at every 0.5 depth. This allowed q_c to be calculated using the following relationship in accordance with SGI Information 15 (see equation 3.4 [20]:

$$q_c = q_t - u_2 \cdot (1 - a) \tag{3.4}$$

Where u_2 is the pore pressure and a is the area factor which is stated in the calibration certificate (Appendix 2 of the site investigation report) for each probe, which is either 0.847 or 0.881 [19].

The process of evaluating and compiling the data to provide a probabilistic distribution was done according to the described workflow in previous chapter.

3.2.2 Unit Weight

The values for the unit weight were derived directly from CONRAD as a function of elevation. The unit weight values obtained from CONRAD represented the saturated unit weight, as the soil below the groundwater table is assumed to be fully saturated. Since PLAXIS requires the unsaturated unit weight as input, it was estimated as $\gamma_{\text{unsat}} = \gamma_{\text{sat}} - 1.0$ kN/m^3, which is a common simplification for saturated sandy and silty soils.

3.2.3 Cohesion

The undrained shear strength was obtained from laboratory tests, a clay cut test done at three depths in borehole SW2125 and fall cone test done with the same samples (see table 3.1).

Table 3.1: Effective cohesion c' for clay (kPa)

Depth	Fall cone	Clay cutting
12 m	2.0	2.8
15 m	2.5	2.0
24 m	6.8	6.8

To convert the undrained shear strength to cohesion (3.5) was used according to Sweco PM [21].

$$c = 0.1 \cdot c_u \tag{3.5}$$

The values at depth 24 m were excluded, as they were considered to be non-representative for the analyzed clay. The c-ϕ correlation coefficient was based on the literature for the SLOPE/W software and was set to -0.3 [8]. A normal distribution was applied to the effective cohesion for all simulations in both SLOPE/W and PLAXIS. Since the measurements of Cu was limited, the standard deviation was estimated based on the available data while ensuring a reasonable coefficient of variation. The standard deviation was set to 0.2 kPa, corresponding to a coefficient of variation (CV) of 8.7 %.

3.2.4 Piezometric Line

The Piezometric line was only used as a input parameter in SLOPE/W. The water level was not treated as a random variable in the primary analysis. Instead, it was held constant at +57.6 m in the river and +59 m in the soil, corresponding to the lowest low-water level as defined in the Sweco geotechnical PM [21]. This combination was selected as it represents the most unfavorable hydraulic condition for slope stability.

An additional analysis was conducted at a water level of +58.2 m, representing the mean flow (MQ) of the watercourse (+59 m in the ground) according to Table 1 in the Geotechnical PM [21]. With the water levels fixed at these values, Monte Carlo and Latin Hypercube simulations were performed in which unit weight and friction angle were assigned distributions.

In a separate analysis, the water level was permitted to vary and was analyzed with the Point Estimate Method (See Chapter 2.3.3), combined with the mean values for friction angle and unit weight in SLOPE/W. Further discussion is provided in Chapter 4.2.1.

4
Simulations and Modeling

The main focus in this chapter will be the simulation as they provide the basis for the statistical analysis. The models will briefly be discussed as they are relatively simple to isolate the effects of parameter distribution.

4.1 Input Parameters for Probabilistic Modeling

The choice of input parameters are important and have to be carefully chosen. How the input parameters were derived and the values that these were based on is described in Chapter 3.2. Following section summarizes the statistical parameters (including the found min, max, mean and standard deviation) used for friction angle, unit weight, and effective cohesion in the probabilistic modeling. These are summarized in Table 4.1.

Table 4.1: Summary of input parameters for sand, clay, fill, and erosion protection

	Friction angle (°)				**Unit weight (kN/m^3)**				**Eff. cohesion (kPa)**
	Sand	Clay	Fill	Erosion prot.	Sand	Clay	Fill	Erosion prot.	Clay
Min	31.1	31.4	–	–	15.0	16.0	–	–	1.8
Max	40.6	35.8	–	–	21.5	20.0	–	–	2.8
Mean	33.8	32.8	37	42	17.6	18.3	20.0	23.0	2.3
Std	2.1	1.2	–	–	1.00	0.71	–	–	0.2

Effective cohesion was not assigned to sand, fill or erosion protection, as these materials are assumed to be cohesionless.

In the three-layer analyses, only the friction angle was varied across the different layers, while the unit weight was kept constant. This is because the layer subdivision was adapted to reduce the coefficient of variation (CV) of the friction angle, and it was considered overly complex to simultaneously match an analysis where both the unit weight and friction angle were subdivided according to their respective CVs. The friction angle was chosen as the varying parameter since it was shown to have a greater influence on FoS compared to unit weight [22]. This represents a potential area for future development. Instead, the mean value of 17.6 kN/m^3 was applied to all layers, which were not assigned a distribution.

Using the N-sigma rule method, outliers were visually removed and an interval was defined to capture data points within a 90% confidence interval. By determining

the length of this interval and dividing by 3.3, the standard deviation was obtained. The mean friction angle for each layer was then determined, and together with the standard deviation, the corresponding + and − values were derived. A summary of the standard deviation, mean, coefficient of variation, and the + and − values of the friction angle for each layer is presented in Table 4.2. The subsequent layering together with the groundwater level is displayed in Figure 4.1

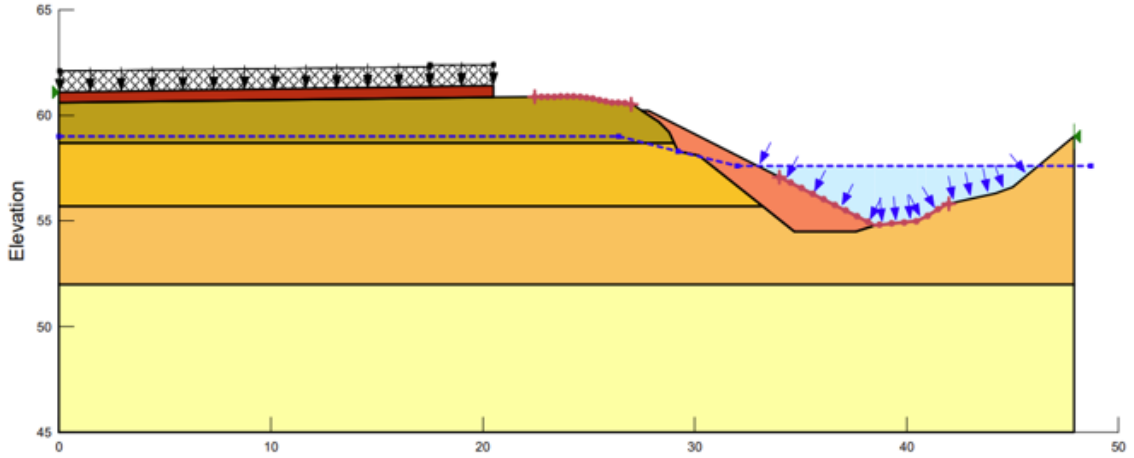


Figure 4.1: Final Stratification after CV reduction through additional layering

Table 4.2: Friction angle, Sand layers

	Stdev (°)	Mean (°)	CV (%)	+value (°)	-value (°)
Layer 1	1.55	33.3	4.6	34.9	31.8
Layer 2	1.45	33.4	4.4	34.8	31.9
Layer 3	1.85	34.2	5.4	36.0	32.3

Griffiths & Fenton explains that the model uncertainty exists due to idealizations made in the physical formulation of the problem [1]. To account for model uncertainties, an additional percentage to the coefficient of variation is introduced. This span is typically between three and five percent [17]. Following equation 4.1 was used to include the model uncertainty in the results:

$$CV_{\text{total}} = \sqrt{CV^2_{\text{geotechnical}} + CV^2_{\text{model}}} \tag{4.1}$$

From this, a new standard deviation could be derived by multiplying it to the mean FoS. The β-value was then calculated according to 4.2:

$$\beta = \frac{\mu_{FoS} - 1}{\sigma_{FoS}} \tag{4.2}$$

4.2 Point Estimate Method

In PEM, it is important to include multiple variables in order to create combinations that yield different factors of safety. PEM was used to analyze the impact on the

factor of safety from the friction angle, unit weight and piezometric line. The FoS for every simulation was calculated through deterministic analyses in SLOPE/W, with the different parameters as input. From PEM, an estimate of the mean and standard deviation of the calculated factors of safety were derived, as described in Section 2.3.3. The probability of failure was calculated in Excel using the built-in function *NORM.DIST(x;mean; standard_dev;cumulative)*, where the mean and standard deviation of FoS were used as inputs together with $x = 1$ and cumulative=TRUE. In this way a graph could be plotted with the FoS on the x-axis and the probability on the y-axis.

The method was first utilized combining the + and - values around the mean, for the friction angle in the sand and the clay layer. Further, a simulation was conducted combining only + and - values of the friction angle and the unit weight of the sand.

Section 3.1.2 and 4.1 describes how the N-Sigma-Rule helps divide the sand layer into three and what the input + and - values around the mean are for each layer. Here the Point Estimate method was utilized by combining these + and - values of each layer of the friction angle. Thereby with the $N = 3$, eight simulations were done combining the different friction angles and resulting in eight factors of safety.

4.2.1 Piezometric line

The Point Estimate Method (PEM) was applied to analyze the effects of the water table. The water table was represented by two independent piezometric lines: one representing the groundwater in the soil and one representing the water level of the river. The other PEM analyses uses ±standard deviations from the mean, however the water levels could not be treated in the same manner. The limited data made statistical derivation of standard deviation unsuitable. Instead, the extreme observed and calculated values were used, representing physically justified worst-case and best-case piezometric conditions.

The upper bound of of +60.2 m was obtained from hydraulic modeling report from SMHI and corresponds to the computed water level in Säveån at Nolhaga during a 200-year return period flood event (HQ200). The same value was adopted as the upper bound for the groundwater table within the slope, representing the rapid drawdown scenario described in "Fall 2" in the Sweco PM.
The lower bound of the river level +57.6 m corresponding to the historically lowest observed low-water level in Säveån. The lowest groundwater table within the slope was determined to +59m in accordance with the "Fall 1" from the Sweco PM. This represents the normal groundwater level within the study area, measured approximately 2.2-3.4 m below ground surface.
Four combinations were evaluated, as presented in Table 4.3.
Four deterministic analyses were performed in SLOPE/W, yielding one factor of safety per run. The standard deviation of the factor of safety could in turn be computed according to equation 2.8 from Chapter 2.3.3. Further, the probability of failure was calculated in Excel using the built-in function `NORM.DIST(1;mean;`

Table 4.3: Combinations of water levels for PEM

Run	WL River	WL Slope
1	+60.2 m (high)	+60.2 m (high)
2	+60.2 m (high)	+59.0 m (low)
3	+57.6 m (low)	+60.2 m (high)
4	+57.6 m (low)	+59.0 m (low)

`standard_dev;TRUE)`, where the mean and standard deviation of FoS were used as inputs.

4.3 SLOPE/W

The first model was conducted using SLOPE/W, where the sections from Nolhaga in Sweco geotechnical PM was used as a guide to the correct geometry. The model was modified until it matched factor of safety of the guide-model. This comparison was made with partial factors listed in 4.4, originally retrieved from Eurocode 7 DA3.

Table 4.4: Partial Factors – Eurocode 7 DA3

Parameter	Category	Value
Permanent Point Loads/Surcharge Loads	Favorable	1.0
	Unfavorable	1.0
Variable Point Loads/Surcharge Loads	Favorable	0
	Unfavorable	1.27
Soil Unit Weight	Favorable	1.0
	Unfavorable	1.0
Other Parameters	Seismic Coefficients	1.0
	Earth Resistance	1.0
Material Parameters	Effective Cohesion	1.3
	Effective Coefficient of Friction	1.3
	Undrained Strength	1.43
	Shear Strength (Other Models)	1.3
Reinforcement Parameters	Pullout Resistance	1.0
	Shear Force	1.0
	Tensile Strength	1.0

Given that this report focuses on the probability of failure and not the probability of under-dimensioning, only characteristic values were used as input parameters going forward. For every calculation conducted with SLOPE/W, the Morgenstern-Price method was used with the half-sine interslice function. This method takes both force- and moment equilibrium into account. The standard in SLOPE/W is that the slip surface is divided into 30 slices, this was later be altered to analyze the

impact of this discretizations.

To determine the possible slip surfaces, the entry and exit method was utilized in this project. This method works by fixating where the slip surface can occur, to use this method it is important to have an understanding of how the failure will behave [8]. To allow for accurate representation of possible slip surfaces the standard of 10 entry points was increased to 15 from the standard 8. The entry range was 4.5 meters and with 15 increments each was 0.3 meters. This increases the computational time, offering a higher chance of finding the most critical slip surface.

4.3.1 Probabilistic Analysis in SLOPE/W

To look beyond the deterministic slope stability analyses, SLOPE/W can perform probabilistic slope stability analyses. It allows variability in the input parameters through application of the distributions described in the theoretical background chapter.

Three probabilistic methods will be used in this thesis including, Monte Carlo simulations, Latin Hypercube and Point Estimate method, each described in chapter 2.2. The implementation of the Monte Carlo method and Latin Hypercube in SLOPE/W includes following steps:

- The selection of a deterministic solution, in our case Morgenstern-Price. (For the deterministic values the the mean value was used.)
- The decisions of what input parameters that will be modeled probabilistically and their distribution model.
- The probability distribution is used to randomly generate new input values, and a new factor of safety is calculated several times for many simulations.

The probability of failure is estimated as the portion of simulations resulting in a factor of safety less than 1.

The analysis in SLOPE/W was based on Latin Hypercube and Monte Carlo simulations. Following simulations were run:

- Monte Carlo 2000, 4000, 8000
- Latin Hypercube 1000, 2000, 4000, 10 000

To compare different distributions of the input parameters and how they affect the results, different combinations of distributions were conducted. Details of all distributions are found in Chapter 2.2. One addition to the distributions is the "Normal distribution with cut off". The "cut off" refers to the distribution being truncated at a certain minimum and maximum value in this thesis the minimum and maximum observed values from field testing, see Figure 4.2. Truncating the distribution avoids assigning unrealistic values in the sampling which can occur since SLOPE/W uses

5 standard deviations with the default settings [8].

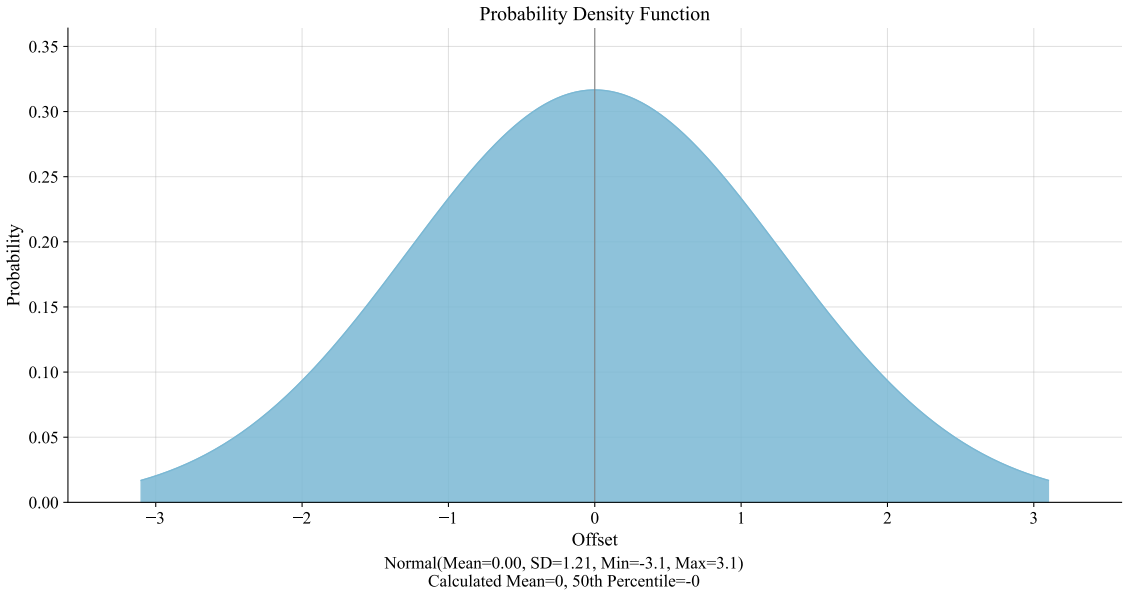


Figure 4.2: Example of normal distribution with cut off in the extremes min =-3.1, max=3.1. *The mean=0 refers to the offset of the mean being 0.

The Generalized spline distribution is presented in Figure 4.3. The distribution was manually entered into SLOPE/W and smoothening of the curve was applied, in this case a 0.20. The smoothing factor regulated how natural the transition between values were. The combined case had the same distribution for friction angle but the unit weight was set to normally distributed.

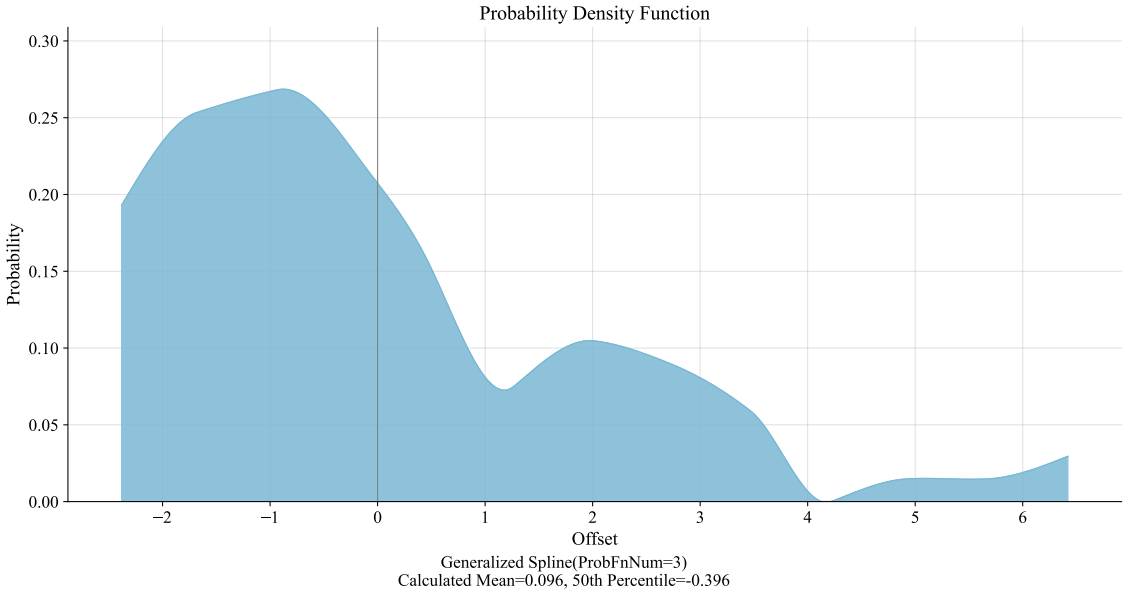


Figure 4.3: Example of Generalized spline distribution.

The different distributions were applied to the friction angle and the unit weight, while the effective cohesion was kept as normally distributed throughout every simulation except for the simulation "Normal distribution with cut off". A summary of the different distribution combinations in simulations are listed below.
An additional division was conducted to the geometry, where the sand layers were divided into three layers. An explanation to this is done in Chapter 3.1.2. For simplicity these layers was considered as normally distributed.
A total of 15 different simulations were done in SLOPE/W (see Table 4.5) combining the different distributions in the input parameters with the different sampling methods.

Table 4.5: Summary of analyses conducted in SLOPE/W with LLW

Sampling Method	**Distribution**	**Trials**
Latin Hypercube	Combined	1000
Latin Hypercube	Combined	2000
Latin Hypercube	Combined	4000
Latin Hypercube	Normal	1000
Latin Hypercube	Normal	10 000
Latin Hypercube	Normal, with cut-off	1000
Latin Hypercube	Generalized spline	1000
Latin Hypercube	Three layers	1000
Monte Carlo	Combined	2000
Monte Carlo	Combined	4000
Monte Carlo	Combined	8000
Monte Carlo	Normal	2000
Monte Carlo	Normal, with cut-off	2000
Monte Carlo	Generalized spline	2000
Monte Carlo	Three layers	2000

4.3.2 Spatial Variation and Correlation

By default, SLOPE/W uses full correlation between the parameters in the different layers along the whole slip surface. This is represented by the matrix only consisting of ones (see 4.3). Full correlation produces unrealistically high probability of failure, as it is unrealistic that the whole soil layer has low strength parameters. To prevent this, SLOPE/W provides a spatial variation alternative where the material properties are allowed to vary along the slip surface. The default setting is that the parameters are sampled for each slice, combined with the standard of 30 slices a low

sampling distance is acquired.

No information about the correlation matrix and if or how it is applied in SLOPE/W is provided in the manual. However, a fixed sampling distance can be applied instead. This correlation can be described through a N-correlation matrix (see 4.3) with ones in the diagonal and the rest is zeros, representing a fully uncorrelated sampling. The size of the matrix depends on how many N sampling points along the slip surface there is.

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{pmatrix} \longrightarrow \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix} \tag{4.3}$$

Figure 4.4 visualizes the effect of correlation between slices. With the shades of red and yellow representing the strength parameters, in this example a fully correlated would all have the same color. The slope on the left has a slight spatial correlation and is between the above presented matrices.

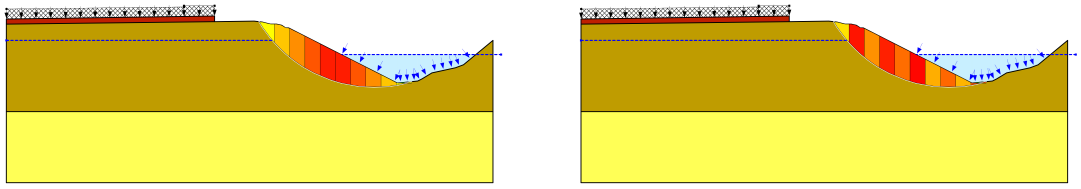


Figure 4.4: Schematic illustration of the slice properties along the slip surface. Left: slightly correlated properties (smooth variation between neighboring slices). Right: uncorrelated properties (independent random variation).

To analyze the impact of sampling distances on the probability of failure, the default setting of 30 slices was used. Then the sampling distance was gradually increased until it was the full length of the slip surface, these results were also compared with the case were spatial variation was deactivated.
An additional slope was constructed in SLOPE/W to confirm the results from the spatial variation. The additional slope was constructed with simple geometry and input parameters, including only one clay layer reaching to bedrock in an undrained analysis. The input parameters are summarized in Table 4.6. A probabilistic analysis with the application of spatial variation was computed and the results were compared to the results from the more complex geometry.

Table 4.6: Material properties for clay in the simple slope

Material	Unit Weight (kN/m³)	Undrained Shear Strength (kPa)
Clay	16	15

4.4 PLAXIS

The model in PLAXIS followed the same geometry as the model in SLOPE/W. However, to ensure that the failure mechanism was the intended, the geometry was simplified, rough edges were removed and the fill at the top was angled and its cohesive strength increased to 10 kPa, the same was done for the erosion protection layer. Individual calculations were not saved, making it impossible to manually check the exact failure mechanism. Since this is pure a stability calculation, parameters affecting deformation are set sufficiently high not to affect the results. The coarseness of the mesh was set to 0.05 to properly capture the mechanism at the top of the slope where the erosion protection material is thin. Later with the additional layers, a finer mesh was applied to ensure that a single element spans an entire layer.

To ensure stability in the fill, the was sloped at a 45 degree angle, as this is not the focus of the study the fill material was assigned a friction angle of 45 degrees and cohesion of 10. This is not physically reasonable but was adopted as a simplification of the problem as it was prone to collapse, especially in PLAXIS.

The analysis was done in three phases. The first phase (initial phase) established the existing ground conditions with gravity loading. The next phase was to establish loads and structures and used plastic deformation. The third and final phase was when c/phi were reduced until the calculation failed to converge, at which point a failure had occurred.

4.4.1 Probabilistic Analysis in PLAXIS

One thing that differs PLAXIS from SLOPE/W, is that PLAXIS does not have a built in probabilistic analysis. Therefore a script is created with a loop that runs several variations of the same model.

The latest versions of PLAXIS 2D have a Python API built-in within their program, which enables remote scripting using Python [23]. Python can help create a automatization of the process to read input data, build the model, run it, take out results and plot the output.
The script is divided into nine sections called cells in Jupyter Lab, each cell has a main purpose. To run the script, the user needs to adapt the parameters in cell 1 and cell 2. After that run all cells in order from top to bottom. The code was structured in such a way that the first section established contact with the server, defines the mesh and the number of simulations conducted, as well as if Monte Carlo or Latin hypercube will be used. The output file paths are defined and how often

backups will be saved.

The second cell defines the soil parameters, where statistical distributions are assigned to each parameter with the mean μ, standard deviation σ, and min/max limits as input. The scripts requires that the names of the materials are exactly the same as in PLAXIS for it to work, the names of the materials can also be changed in cell 2. The saturated unit weight was not treated as an independent random variable but was defined as $\gamma_{sat} = \gamma_{unsat} + 1.0 \quad \mathrm{kN/m^3}$, this can also be changed in cell 1.
All calculation phases (InitialPhase, Phase_1 and Phase_3) were recomputed for every simulation run. This is necessary because the unit weight γ affects the stress state computed in the initial phases, meaning that only rerunning the Safety phase would introduce an inconsistency between the stress conditions and the sampled material parameters, this can be changed automatically if the unit weight is defined as "none" in the distribution. Then the script would only calculate the initial conditions once and then only calculate the safety phase, drastically improving the runtime.

The third cell loads all necessary libraries listed and described in 4.7. The Monte-Carlo and Latin hypercube functions are also defined. The fourth cell generates the mesh, the same mesh will be used for every simulation to keep the geometry and calculations consistent. Before running the script it is beneficial to conduct a mesh-dependency study as this is not included in the script.

Table 4.7: Python libraries used in the Monte Carlo simulation framework.

Library	**Purpose**
`plxscripting`	PLAXIS Remote Scripting API — controls PLAXIS 2D from Python
`numpy`	Numerical computations and random sampling
`scipy`	Kernel Density Estimation for empirical parameter distributions
`pandas`	Tabular storage and Excel export of simulation results
`matplotlib`	Visualisation of FoS distributions and sensitivity plots
`openpyxl`	Backend for writing `.xlsx` files via Pandas
`pyautogui`	Prevents screen lock by moving the mouse cursor one pixel every 30 seconds via a background thread.
`psutil`	Operating system process management

The fifth cell is the main computational part, this is where the previously generated parameters are paired and the calculations are preformed. The $\sum M_{sf}$ is extracted from the Output server and saved in excel at the specified intervals defined in cell 1.

The next cell sorts the output, a separate sheet in the excel file is created where all values below 1.0 is stored, this makes it easier to validate the results and investigate the failure mechanism through rerunning individual simulations. This is also where

the graphs of cumulative density functions and probability density functions are generated. The smooth curve is estimated using Kernel Density Estimation (KDE), where a Gaussian kernel is placed over each simulated $\sum M_{sf}$ value and the resulting curves are summed to produce a continuous approximation of the underlying probability distribution

To gain a deeper understanding of which parameters have the greatest influence on the Factor of Safety, correlation diagrams were produced for each randomized parameter. The Pearson correlation coefficient r is used to quantify the strength of the linear relationship between each input parameter and the resulting $\sum M_{sf}$, where values close to ± 1 indicate a strong influence and values close to zero indicate little influence.

To verify that the sampling procedure produced the intended distributions, histograms of the sampled parameter values are presented alongside the theoretical probability density function. This confirms that the sampling covered the full range of each defined distribution.

4.5 Probability of failure

In SLOPE/W the probability of failure can be defined as the the number of simulations that resulted in FoS < 1.0 divided with the total number of simulations. This approach limits the resolution of the probability of failure to 1/N, where N is the total number of simulations conducted. Failure probabilities below this threshold cannot be resolved.

Rather than counting simulation outcomes, a more statistical approach was be utilized, the β-function, or reliability index as it called in SLOPE/W. This is a method used to calculate probability based on the mean and standard deviation. It represents how many standard deviations the mean Factor of Safety (FoS) lies above the failure threshold of FoS $= 1.0$. A higher β corresponds to a lower probability of failure, related through the standard normal distribution as:

$$P_f = \Phi(-\beta) \tag{4.4}$$

where P_f is the probability of failure.

In this report, the β-function was implemented for both limit equilibrium and finite element analyses. The β-function can be directly applied in Excel through the function: *1-NORM.S.DIST(β-value;TRUE)* (where TRUE returns the cumulative function) and this was used for the Point Estimate Method and when the value of β was extracted from SLOPE/W. The script in PLAXIS used the command *Scipys* which utilizes the same β-function to extract the probability of failure from the value of β. However, in Excel very low probabilities are presented as 0. To address this problem, the Excel function *NORM.DIST(x;mean;standard_dev;TRUE)* was utilized. With the last argument set to TRUE, the function returns the cumulative

normal distribution, i.e. the probability that the variable is less than or equal to x (here x=1).

According to the Swedish standard SS-EN 1997-1:2024 "Geotechnical Design", part of the second generation Eurocode, any calculation model for geotechnical analysis shall be validated to confirm that it satisfies the requirements of the level of reliability specified in EN 1990 [24]. The recommended minimum values defined in the Swedish standard SS-EN 1990, see Table 4.8.

Reliability Class	**Minimum values for β**		**Approximate P_f**	
	Reference period 1 year	**Reference period 50 years**	**Reference period 1 year**	**Reference period 50 years**
RC3	5.2	4.3	10^{-7}	10^{-5}
RC2	4.7	3.8	10^{-6}	10^{-4}
RC1	4.2	3.3	10^{-5}	$\approx 5 \times 10^{-4}$

Table 4.8: Recommended minimum reliability index β and corresponding failure probability P_f per reliability class according to EN 1990.

The related values of Pf according to equation 4.4 are also listed. The reliability class are related to their own consequence classes, ranging from low to high risk of loss of life, and small to very large economic, social or environmental consequences [25]. Higher reliability classes imposes stricter requirements on the probability of failure. For the case of Nolhaga, where a new kindergarten is planned, the project would most probably fall within the reliability class RC2. According to the Swedish standard, RC2 applies to residential and office buildings, as well a public buildings where the consequences of failure are medium [25]. This description is considered applicable to a kindergarten.
Eurocode assigns common building structures a design working life of 50 years [24] and this was used as the relevant design horizon fo the kindergarten.

5
Results

All three probabilistic tools yield similar results for the mean factor of safety (FoS), while some differences in the probability of failure (P_f) are observed and will be compared. The comparison between the tools is based on normally distributed input parameters. In all figures, the minimum P_f displayed is set to 10^{-9}; values below this threshold are instead reported in tables.In all three cases, P_f is higher than the guideline value defined in 4.8. However, when using three layers, the value is consistently substantially lower than the recommended value for RC2.

5.1 Results from Point Estimate Method

Following tables 5.1 and 5.2 show an example of how the probability of failure could be derived from the different combinations of the +/- 1 standard deviation from the mean of, in this case, the friction angle in the 3 Layers subdivision.

Table 5.1: Combinations of +/- 1 standard deviation from the mean and the FoS calculated for these.

Cases	SF_{Morg}
+ + +	1.55
+ + −	1.43
+ − +	1.48
− + +	1.53
− − +	1.47
− + −	1.42
+ − −	1.39
− − −	1.38

The mean and standard deviation could then be derived from the above calculated FoS. Table 5.2 show the results from the calculated probability of failure according to Equation 2.11.
In Figure 5.1 the cumulative density functions from the PEM-calculations are presented, including the method combining different water levels, unit weight and friction angle in the sand layer, and varying friction angles in the three layers. The "Unit weight and friction angle" together with the "3 Layers" were modeled with the water table at +57.6 m.

Table 5.2: Calculation of Probability of Failure from the previously calculated FoS.

Results	
Mean	1.46
$(E(y))^2$	2.1
$E(y^2)$	2.1
STDV	0.06
VIP	4%
$P(f)$	6×10^{-15}

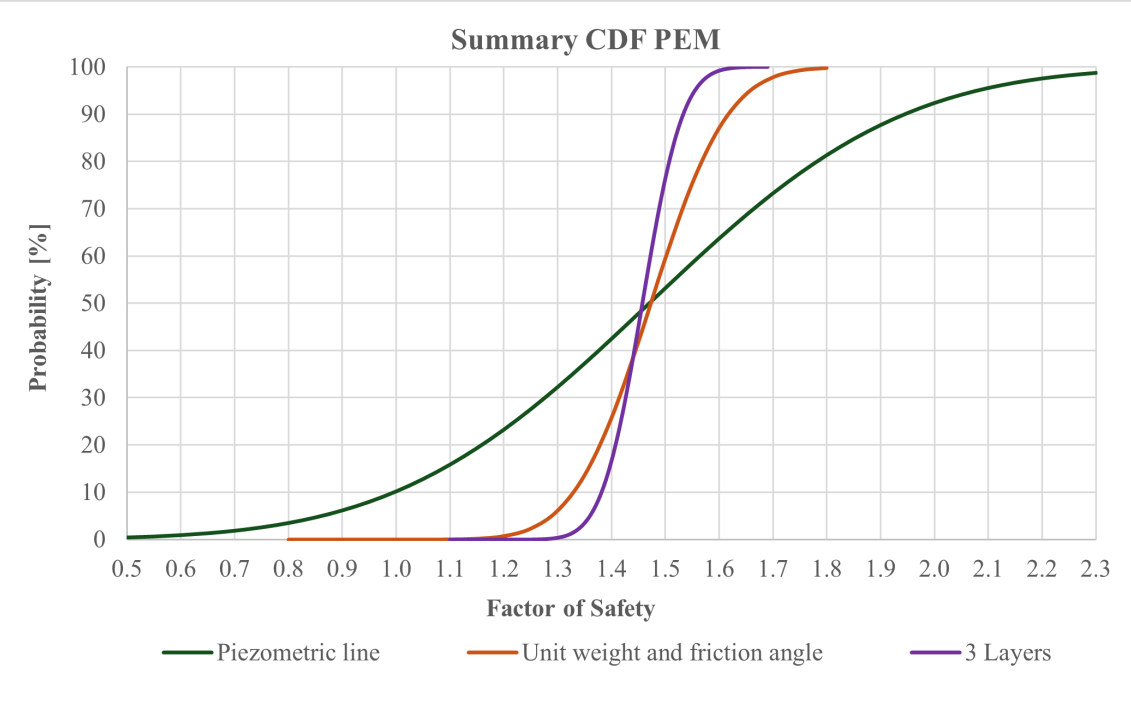


Figure 5.1: Results from the Point Estimate Method.

The results from the first simulation combining the + and - values around the mean for the friction angle in the sand and clay layer showed that the friction angle in the clay does not have any impact on the FoS. Consequently it is not suitable as a variable in PEM and is hence not included in the graph.
Table 5.3 summarizes the statistical data that the graph is based on. Equations from Chapter 2.3.3 are utilized to calculate the expected value and the standard deviation. The calculated mean, coefficient of variation and probability of failure are also presented.

Table 5.3: PEM results with the water level +57.6 m for "Unit wiegth & Friction anlge" and "3 Layers".

	Piezometric line	**Unit weight & Friction angle**	**3 Layers**
Mean FoS	1.47	1.47	1.46
σ	0.32	0.11	0.06
CV	21.9%	7.6%	4.1%
β	1.5	4.2	7.7
P_f	7×10^{-2}	1×10^{-5}	6×10^{-15}

5.2 Results from SLOPE/W

The deterministic result gave a factor of safety of 1.096 (ODF), which matches the Sweco model of 1.1 (ODF). See Appendix A for the reference model from Sweco and the model constructed based on the reference model.
Figure 5.2 shows the cumulative distribution function for the simulations conducted, including 1000, 2000, 10000 runs using Latin Hypercube and 2000 runs using Monte Carlo simulations. The curves show that the result converges relatively quickly and that there is no real difference between 1000 10 000 LH runs. With this, only results from the Latin Hypercube would be reported going forward.

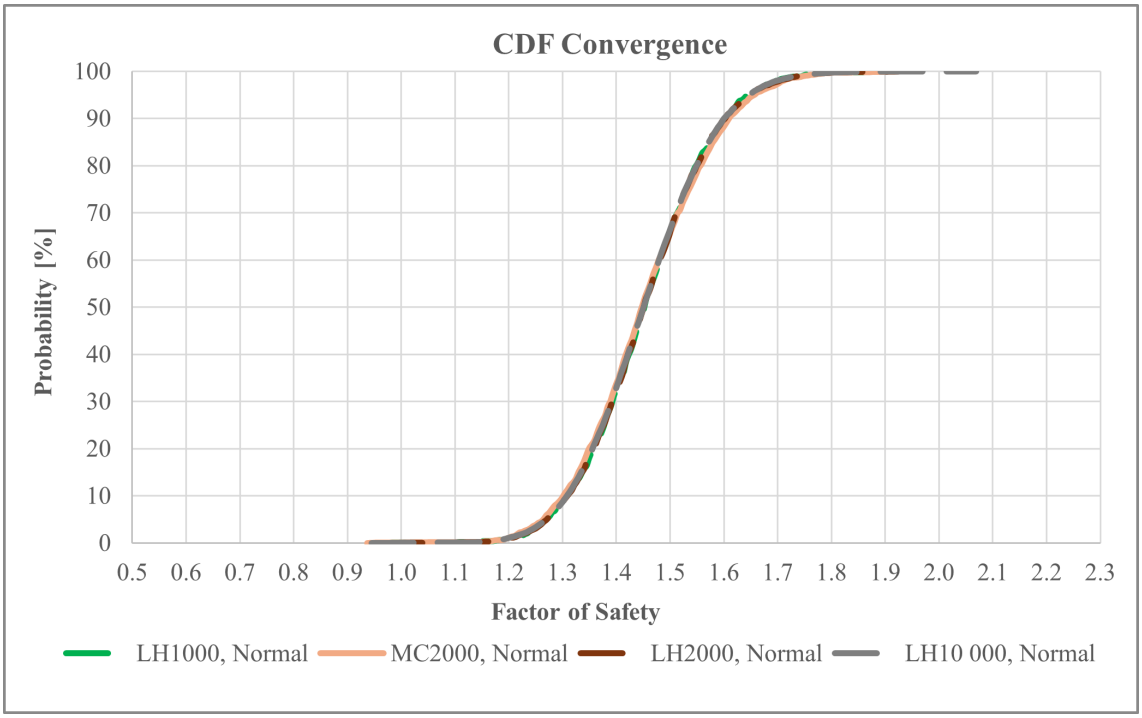


Figure 5.2: Convergence of simulations done in SLOPE/W: Monte Carlo and Latin Hypercube, 1000, 2000 & 10 000 runs (water level at +57.6 m)

The simulations done with Latin Hypercube and 1000 runs are summarized in Figure 5.3 where the probability is plotted against the factor of safety. The results show

that the simulation with three layers is slightly shifted from the other simulations, where the spread of the FoS is lower. However, the differences are minor. Figure 5.4 shows the results from the simulations done with a higher water level at +58.2 m. The results are slightly shifted to the right, naturally decreasing the P_f.

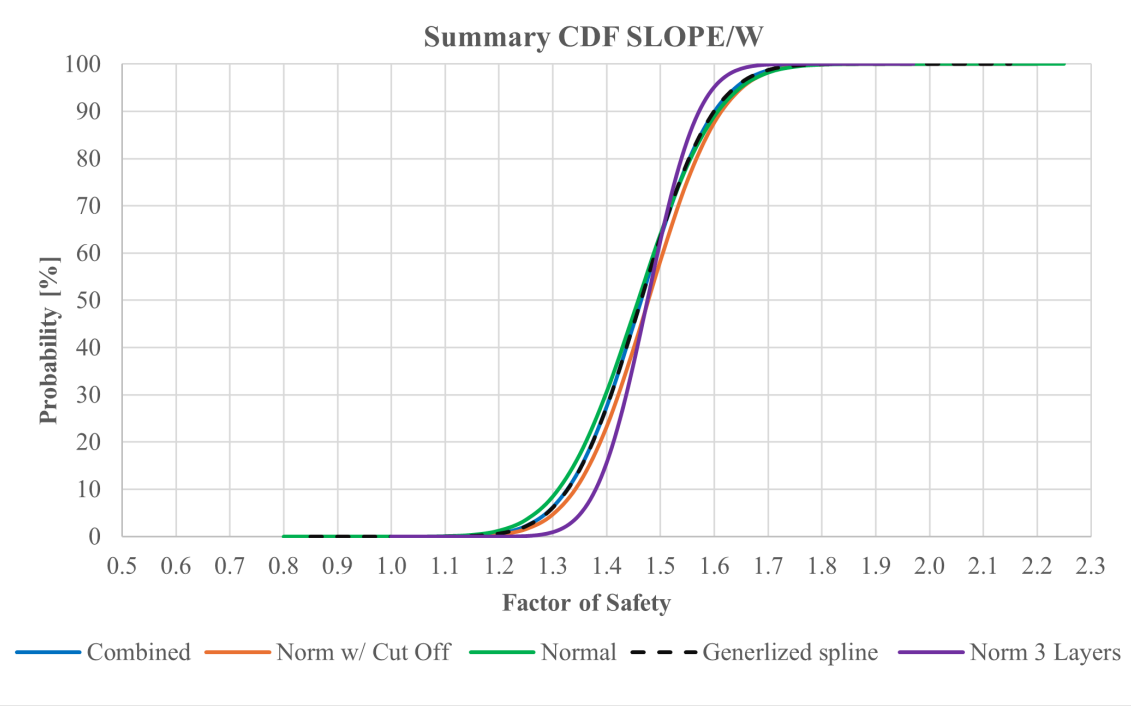


Figure 5.3: Results from SLOPE/W using water level of +57.6 m

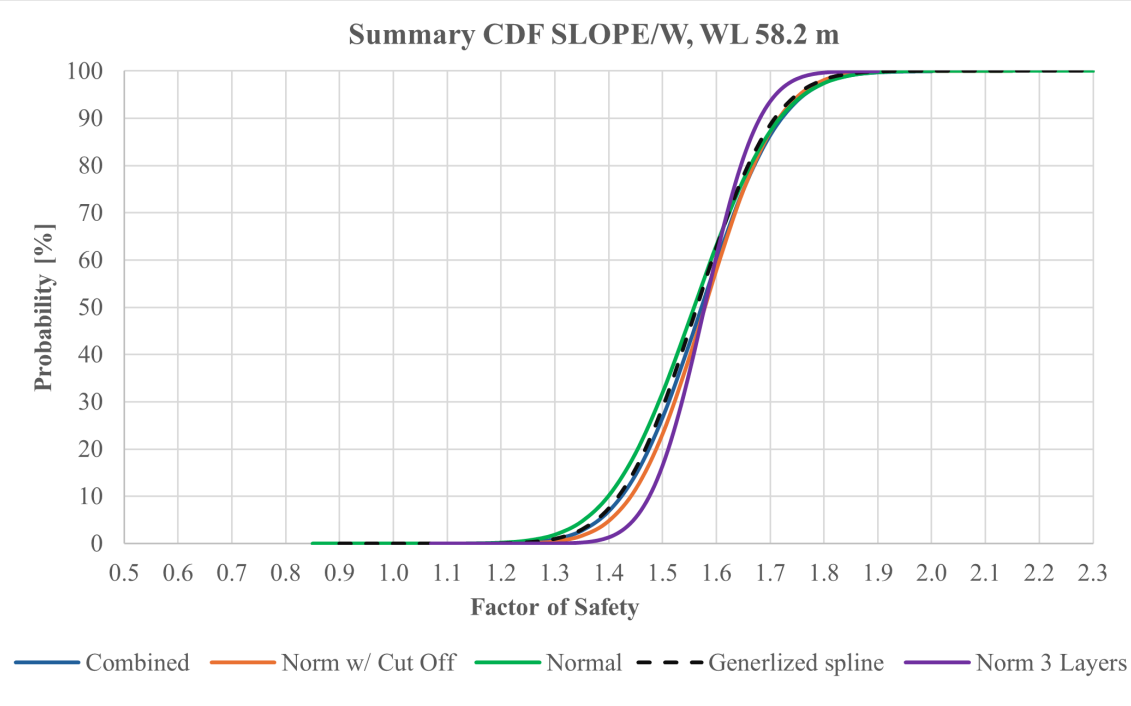


Figure 5.4: Results from SLOPE/W using water level of +58.2 m

Table 5.4 and 5.5 summarizes the results from the Latin Hypercube simulations with 1000 runs. The influence of the elevated water level on P_f is reflected in the results, consistent with the trend discussed previously. It can be noted that the statistical results are relatively similar except for the simulation "Normal 3 Layers" which yields significantly lower probability of failure.

Table 5.4: SLOPE/W results with water level +57.6 m.

	Combined	**Norm w/ Cut Off**	**Normal**	**Generalized Spline**	**Norm 3 Layers**
Mean FoS	1.46	1.48	1.46	1.46	1.48
σ	0.11	0.11	0.12	0.11	0.075
CV	7.3%	7.2%	7.9%	7.2%	5.0%
β	4.3	4.8	3.97	4.4	6.3
P_f	7×10^{-6}	8×10^{-7}	4×10^{-5}	6×10^{-6}	1×10^{-10}

Table 5.5: SLOPE/W results with water level +58.2 m.

	Combined	**Norm w/ Cut Off**	**Normal**	**Generalized Spline**	**Norm 3 Layers**
Mean FoS	1.57	1.58	1.56	1.56	1.58
σ	0.12	0.11	0.12	0.11	0.080
CV	7.4%	6.8%	8.0%	7.2%	5.1%
β	5.0	5.4	4.5	5.0	7.2
P_f	3×10^{-7}	4×10^{-8}	4×10^{-6}	3×10^{-7}	3×10^{-13}

One important observation is the choice of slip surface when looking at the statistical parameters derived from SLOPE/W. The critical slip surface is the one with the lowest factor of safety, not necessarily aligning with the one with the highest probability of failure. As shown in Table 5.6, the simulation with the combined input parameters showcase different values for the critical slip surface (noted "critical") and the one with the highest probability of failure.

Table 5.6: SLOPE/W results with water level +58.2 m, combined critical slip surface vs combined lowest beta index.

	Combined "Critical"	**Combined "Lowest beta index"**
Mean	1.57	1.57
σ	0.060	0.12
CV	4.0%	7.4%
β	9.5	5.0
P_f	8×10^{-22}	3×10^{-7}

The results from remaining simulations, including Monte Carlo simulations and Latin Hypercube for more than 1000 runs, are summarized in Appendix B.

5.2.1 Spatial Variation

The correlation between strength parameters along slip surfaces is not quantified and cannot be changed by the user. By default, the parameters are assumed to be fully correlated (no spatial variation). When spatial correlation is enabled, the opposite extreme is applied then the slope if parameters are totally uncorrelated over the slip surface instead. The difference of these extremes can be seen in 5.5.

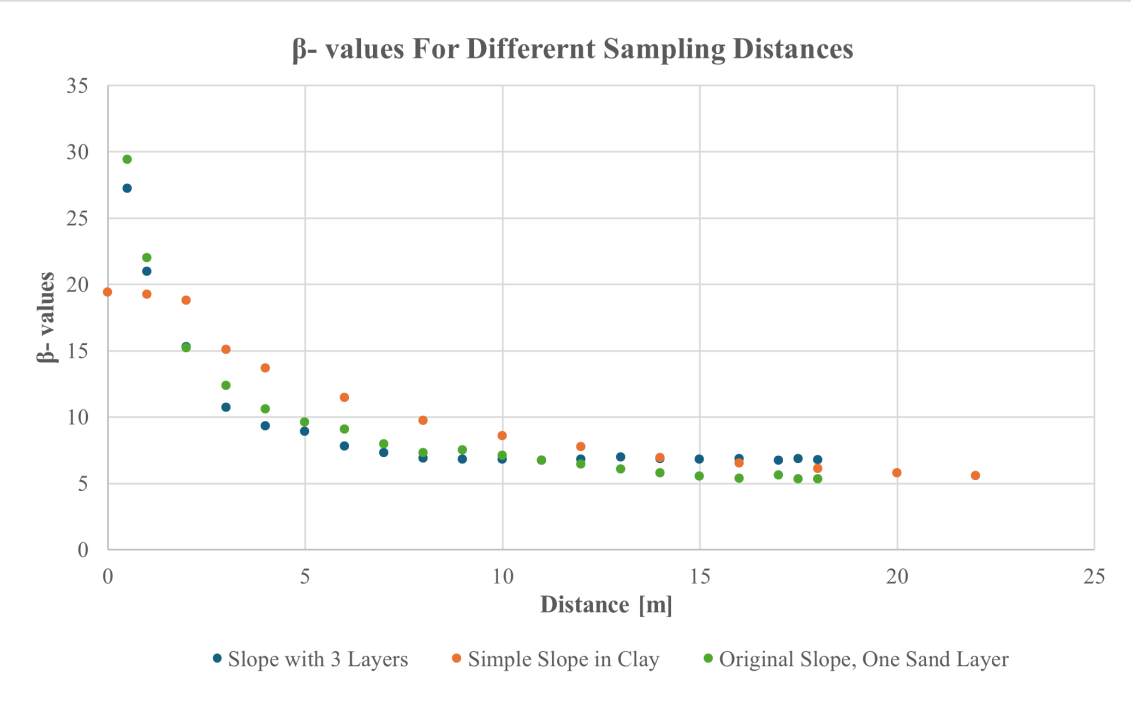


Figure 5.5: Beta values for different sampling distances with three different geometries; slope with 3 layers, simple slope in clay, original slope with one sand layer.

The same slip surface ranges from values at approximately $\beta = 5$ to $\beta = 30$ for the slope with three layers, corresponding to failure probabilities of $P_f = 1.0 \times 10^{-4}$ and $P_f \to 0$ respectively. The effects are similar in the original slope proving little to none correlation is involved in the sampling along the slip surface. The simple slope in clay also follows the same pattern but with lower beta values.

5.3 Results from PLAXIS

The results from PLAXIS include additional information beyond the standard results presented in the other sections, namely Pearson's correlation coefficients and input parameter distributions, to provide deeper insight into the analysis.

The results from PLAXIS show the same trend regarding P_f for the cases with one and three layers as seen in Figure 5.6. But with slightly higher mean FoS as seen in and table 5.7.

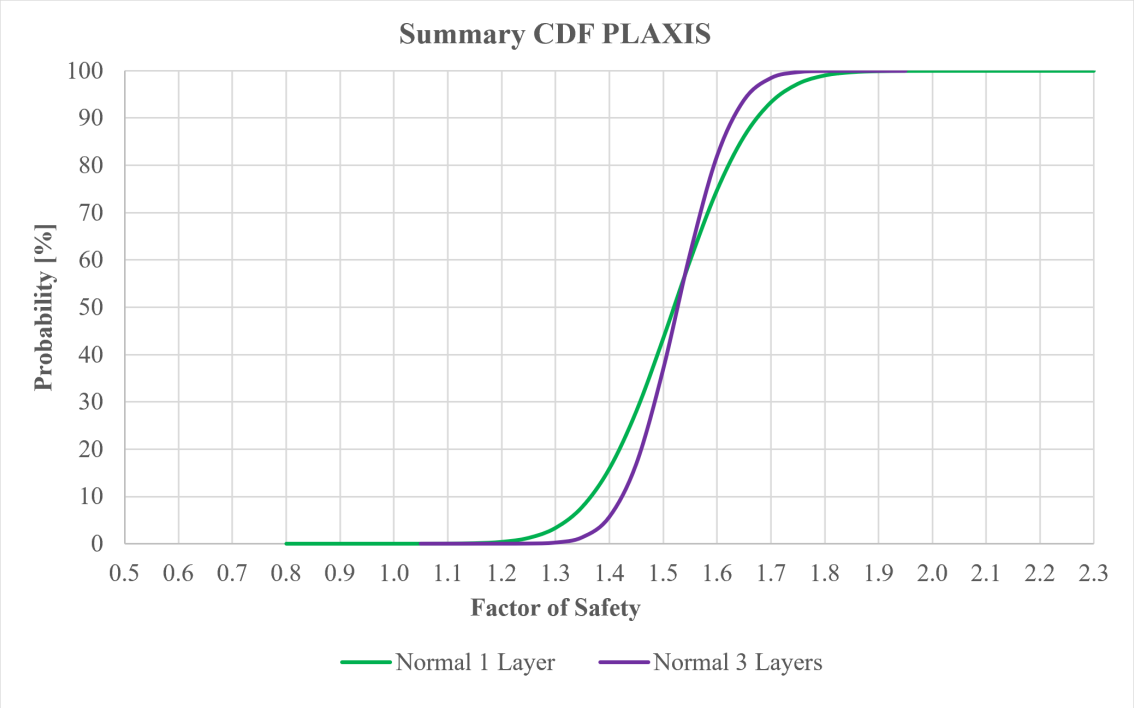


Figure 5.6: Results from PLAXIS using 3 Layers and normal distribution (water level +57.6 m).

Table 5.7: PLAXIS results with water level +57.6 m

	1 Layer	**3 Layers**
Mean FoS	1.52	1.53
σ	0.12	0.08
CV	7.7%	5.2%
β	4.4	6.6
P_f	5×10^{-6}	2×10^{-11}

The Pearson's correlation coefficients for all parameters, including the clay layer, are presented in Appendix C. The clay parameters show a correlation close to zero, confirming that the critical slip surface does not reach the clay layer and that its properties have a negligible influence on the factor of safety. The most influential parameters are shown in Figure 5.7, where the third sand layer exhibits the highest correlation with the factor of safety compared to the other layers, as a greater proportion of the critical slip surface passes through it. The unit weight also shows a slight positive correlation, suggesting that it acts as a resisting force.

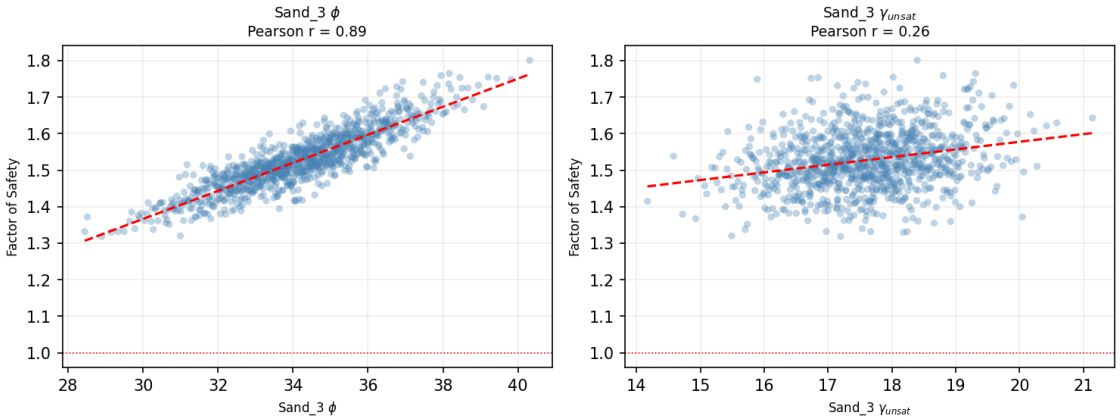


Figure 5.7: Correlation of the friction angle and unit weight with FoS in the third sand layer.

To confirm that the sampling procedure produced the intended distributions, histograms of the sampled parameter values are presented alongside the theoretical probability density function in Figure 5.8. A larger number of simulation runs will produce a histogram more closely following the theoretical distribution, thereby serving as a visual indicator of convergence.

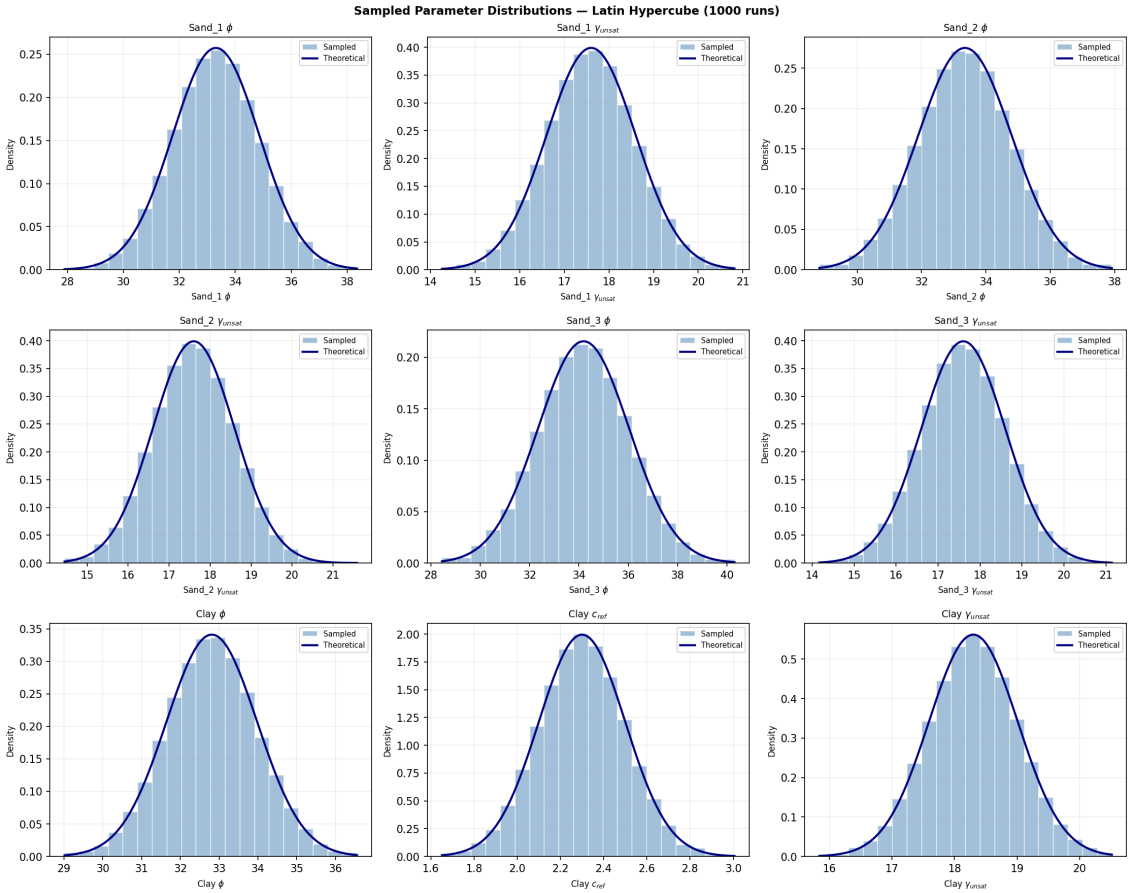


Figure 5.8: Input distribution for 3 sand layers in PLAXIS

The correlation analysis and input parameters for the simulation conducted with one sand layer is presented in Appendix C and D.

5.4 Comparison of the results

For clarity the normal distribution with one and three sand layers was chosen for comparison and the results are presented in Table 5.8. In the column for the PEM calculations, "Normal" refers to the calculations combining the unit weight & friction angle. The results are similar with some exceptions. The mean FoS is 3-5% higher in PLAXIS compared to the mean from SLOPE/W and the mean obtained with PEM.

Table 5.8: Summary of β and P_f for PEM, SLOPE/W simulations, and PLAXIS simulations (water level at + 57.6 m).

	PEM		**SLOPE/W**		**PLAXIS**	
	Normal	**3 Layers**	**Normal**	**3 Layers**	**Normal**	**3 Layers**
Mean FoS	1.47	1.46	1.46	1.48	1.52	1.53
σ	0.11	0.06	0.12	0.075	0.12	0.08
CV	7.6%	4.1%	7.9%	5.0%	7.7%	5.2%
β	4.2	7.7	4	6.3	4.4	6.6
P_f	1×10^{-5}	6×10^{-15}	4×10^{-5}	1×10^{-10}	5×10^{-6}	2×10^{-11}

5.5 Model uncertainty

The results from the accounted model uncertainty of 3 % and 5 % are summarized in Table 5.9. As expected, the β-value decreases with the introduction of model uncertainty and consequently the probability of failure increases. Despite this reduction, all simulations conducted with three layers still yield β-values well above the accepted target value of 3.8 specified in EN 1990 for consequence class CC2.

Table 5.9: Summary of the results from PEM, SLOPE/W, and PLAXIS including model uncertainty for normal distribution with one and three sand layers (water level at 57.6 m).

	PEM		**SLOPE/W**		**PLAXIS**	
	Normal	**3 Layers**	**Normal**	**3 Layers**	**Normal**	**3 Layers**
Mean FoS	1.47	1.46	1.46	1.48	1.52	1.53
CV	7.6%	4.1%	7.9%	5.1%	7.7%	5.2%
$CV_{0.03}$	7.7%	4.4%	8.5%	5.9%	8.3%	6.0%
$CV_{0.05}$	7.8%	5.0%	9.4%	7.1%	9.2%	7.2%
β	4.2	7.7	4.0	6.3	4.4	6.6
$\beta_{0.03}$	4.0	7.0	3.8	5.9	4.1	5.7
$\beta_{0.05}$	3.8	6.0	3.6	5.3	3.7	4.8
P_f	1×10^{-5}	6×10^{-15}	4×10^{-5}	1×10^{-10}	5×10^{-6}	2×10^{-11}
$P_{f,0.03}$	2×10^{-5}	3×10^{-12}	6×10^{-5}	2×10^{-9}	2×10^{-5}	5×10^{-9}
$P_{f,0.05}$	6×10^{-5}	2×10^{-9}	1×10^{-4}	6×10^{-8}	1×10^{-4}	9×10^{-7}

6
Discussion

In the geotechnical field in Sweden the probabilistic approach is not the norm, this is due to a long tradition of deterministic calculations. The way slope stability traditionally have been determined is through quotient of forces acting to mobilize the slope and forces supporting the slope called the Factor of Safety (FoS). Another way to look at slope stability is the probability of failure i.e the probability that the quotient is below 1.0. The aim of this project has been to promote a more statistical approach to slope stability by making it easier to understand and by applying three different tools. These tools are the Point Estimate Method, existing probabilistic functions in SLOPE/W, and a custom code that serves as a statistical tool for PLAXIS. These three have been compared and drawbacks have been highlighted especially considering spatial variation.

6.1 Distributions

Every distributions involves trade-offs between realism and simplicity. In this report a cut off is introduced to the normal distribution, the cut off is determined by the highest and lowest values observed in the field. Another way to do it 5-95 percentage i.e $\pm$ 1.64 standard deviations or 1-99 % i.e. $\pm$ 2.57 Stdev. When using cut offs, Pf naturally decreases, however this can introduce a false sense of security if the lower limit is set too high. Generalized spline have a similar drawback where only observed values are included with the additional drawback that it is time-consuming and not well defined how to order the data in "bins" and then assign a suitable probability density function.

Lognormal, uniform and triangular distributions have not been applied in this thesis but are an option to include in SLOPE/W and the python script. Uniform and triangular are intuitive and easy to use. This makes them appropriate to use for educational purposes but not more advanced applications as the complexity of natural soils cannot be defined in such simple terms. Lognormal on the other spectrum of this, it is more complex but excludes values below zero which is beneficial when applying distributions to strength parameters.

It is important to be transparent with the assumptions that are made when assigning distributions to parameters. Combing theoretical distributions with engineering judgment produces the best results. Geotechnical experience allows for variance reduction through clever layering of the soil. The probability of failure for the model

with one layer was 4×10^{-5} compared to the model with three layers which was 1×10^{-10} (at a water level of +57.6 m). This highlights the importance of not solely relying on statistics. The other distributions have a much narrower span of probability of failure, roughly a factor of 100 which still is significant.

6.2 Point Estimate Method

Point Estimate Method works well when the variance of the parameters is low. This is the case for the western part of Sweden where the horizontal spread is usually very low which makes the method useful. It becomes even more powerful when combing it with N-sigma rule, this helps to reduce the variance in the soil by dividing more layers. These layers enable more combinations of variables, so more layers gives a greater value for 2^N which in return provides a higher confidence. This is equivalent to increasing the number of runs with Monte Carlo or Latin Hypercube, but with the drawback that the workload increases exponentially. PEM is an appropriate method to use as a starting point.

In this thesis, PEM was applied together with SLOPE/W to calculate the FoS. An alternative could be the PEM used in combination with Janbus direct method, which can provide a quick overview of the probability of failure without the need to build a full model. However, to increase the confidence in the results, more advanced software such as SLOPE/W should be consulted.

From the PEM varying the water levels, a higher probability of failure of 7% and a CV of 21% is obtained. These results are significantly higher compared to the variation of the other parameters. This fact emphasizes the importance of being able to control water levels in a realistic way. It should be noted that for the PEM calculations, the extremes were used as + and - respectively. These values are time-dependent and have a really low probability of occurring simultaneously, and should hence be taken as an indication of the water level's influence on slope stability rather than a realistic estimate. This compromise was made as no standard deviation for the water level was obtained in this study. For a more comprehensive treatment of water level uncertainty, see Section 6.7.

6.3 SLOPE/W

The results converge faster than expected. The simulations with 4000 and 8000 runs show no meaningful difference, and an additional simulation with 10 000 runs confirmed the same trend.

SLOPE/W defines the probability of failure in two ways in its output, which can be confusing for the user. The $P(F)$ presented in the output represents the ratio of failed runs to total runs, which becomes unreliable when a low number of simulations are conducted. For users unfamiliar with the relationship between the reliability in-

dex β and P_f this can become an issue. $P(F)$ may be mistakenly interpreted as the true probability of failure, which leads to a significant underestimation of the true P_f.

Furthermore, the critical slip surface is, in SLOPE/W, defined as the slip surface with the lowest deterministic factor of safety, not the surface with the highest probability of failure. This is not intuitive and should be verified before drawing any conclusions. In most cases the slip surface with the lowest β coincided with the critical slip surface. However, this was not the case for the combined simulations in SLOPE/W, where the critical slip surface yielded a reliability index of $\beta = 9.5$ corresponding to $P_f = 8 \times 10^{-22}$, compared to the lowest observed value of $\beta = 5.0$ corresponding to $P_f = 3 \times 10^{-7}$. A large difference is noted and reporting only the critical slip surface could therefore potentially lead to a significant underestimation of the probability of failure.

6.4 Spatial Variability

Spatial variability has proven to have a significant impact on the probability of failure, none of the three probabilistic tools used in this thesis provides a good solution to this problem. What can be said is that SLOPE/W uses the extremes as the default settings which is not well explained in the user interface. With the press of a button the Pf goes from $\beta = 6.77$ to $\beta = 27$ representing a slope with relatively low Pf to a non existing probability of failure. This indicates that there is no correlation between the sampling points along the slip surface. An analogy for this is rolling a dice and assuming the slope fails if the result is a 1. With a single roll, the probability of failure is $\frac{1}{6}$. However, if the failure criterion is instead based on the average of 30 rolls being equal to 1, every single roll would have to show a 1, making the outcome virtually impossible. The analogy illustrates how assuming no correlation between sampling points along the slip surface leads to an unrealistically low probability of failure, as the average will always tend toward the mean value rather than the extreme.

The other extreme is also problematic, but in a different way. It overestimates the probability of failure. Continuing with the dice analogy, a single unlucky roll could lead to the entire slip surface being assigned unreasonably low strength parameters. For the soil to have exactly the same properties along a 20 meter long slip surface is highly unlikely. The answer likely lies somewhere between these two extremes, but exactly where, remains difficult to quantify. Moving forward, it is advised to avoid using the spatial variation function in SLOPE/W, as it tends to yield overly optimistic estimates of the probability of failure.

6.5 PLAXIS

The script fulfills its primary purpose of enabling probabilistic slope stability analysis in PLAXIS 2D with minimal programming prerequisites. However, the script requires some knowledge of installing plugins and managing file paths. The script

is designed to be accessible to users with a basic understanding of geotechnical engineering, requiring only the definition of statistical distributions in a dedicated cell before execution.

The main limitation of the implementation is the computational demand. Each simulation run requires both the PLAXIS Input and Output modules to operate simultaneously, and the calculations are inherently slow due to the complexity of the finite element model. To prevent the operating system from locking the screen and to manage the accumulation of open Output tabs, the script continuously moves the mouse cursor and closes tabs automatically. As a consequence, the computer is practically unusable during the entire simulation, which is a significant practical limitation for longer runs.

The implementation was verified by reproducing selected simulation runs manually in PLAXIS 2D using the parameter values recorded in the results file, yielding identical Factors of Safety.

From the correlation analysis, it is clear that the unit weight does not exhibit a strong correlation with $\sum M_{sf}$ for the analyzed section. Excluding unit weight as a random variable reduces the number of phases calculated per run from three to one, as the initial stress conditions remain unchanged between runs. Consequently, the computation time was reduced by approximately 50%.

Furthermore, the correlation coefficient close to zero for the clay parameters, as presented in Appendix C, confirm that the critical slip surface does not reach the clay layer, thereby validating the model behaviour.

The saturated unit weight was not treated as an independent variable but defined as $\gamma_{sat} = \gamma_{unsat} + 1.0$ kN/m^3. This is a simplification that may not reflect the true relationship between saturated and unsaturated unit weight in all soil types and can thus be changed manually in the script.

Given the above limitations, the script should be regarded as a tool for educational purposes rather than a production-ready framework. Latin Hypercube Sampling is recommended over Monte Carlo for this application, as it achieves comparable accuracy with significantly fewer simulation runs, which is particularly valuable given the slow runtime.

The issue of the critical slip surface not aligning with the slip surface that yields the lowest beta index does not occur in PLAXIS. Unlike SLOPE/W, which evaluates stability along a predefined slip surface, PLAXIS progressively reduces the shear strength parameters until failure occurs naturally. Hence, the results from PLAXIS may be more straightforward to interpret.

6.6 Reliability Assessment and Model Uncertainty

The required reliability index depends on the reliability class assigned to the structure. In the case of Nolhaga, a reliability class RC2 was considered. This applies to common residential, office and public buildings where the consequences of collapse are moderate. However, an argument could be made for the stricter class RC3 given the vulnerable nature of the occupants. For the Reliability Class RC2, the minimum required reliability index is β=4.7 for a 1- year reference period and β=3.8 for a 50-year reference period. On the other hand, RC3 requires β=5.2 and β= 4.3 respectively.

Comparing these thresholds with the calculated β-values from the different simulations done with normal distributions in Table 5.8, some does not meet the requirements for a 1-year reference time period. These include the "Normal" (one sand layer) from PEM (β=4.2) and SLOPE/W (β=3.97). However, in this study the 50 year reference period (requiring β=3.8 for RC2) was chosen as reference for the slope in Nolhaga which all calculated β-values exceeds. The β-values from one sand layer in PEM and SLOPE/W does not meet the RC3 requirements of β= 4.3 and indicates insufficient reliability when the soil variability is described with a single sand layer. When the sand layer is subdivided into three layers, the β-values increase, exceeding the not only the RC2 requirements but also the stricter RC3 requirements. This highlights the sensitivity of the reliability assessment to the level of detail in the soil model. A more refined layering reflects the natural variation with depth and leads to more favorable estimate of slope reliability.

The results also highlights the importance of accounting for model uncertainties. Table 5.9, where a coefficient of variation of 3 % and 5 % represents the model uncertainty, show that this introduction reduces the *beta*-values and increases the P_f. For example, introducing a model uncertainty of 5 % to the three-layer model in SLOPE/W decreases the *beta*-values from 6.3 to 5.3. This suggests that model uncertainty can have a significant impact on the estimated probability of failure and should be considered in any probabilistic slope stability assessment to provide an extra layer of safety. A value of 5% is considered high and should be applied to slopes with greater uncertainties in height and geometry. The Nolhaga slope is well-documented with extensive site investigations, and a model uncertainty of 3% is therefore considered more appropriate for this case. The β- values for the 3 % are sufficient for all cases when comparing it to the RC2 requirements with a 50 year time period.

6.7 Hydraulic Conditions

As mentioned, the minimum beta index defined in Eurocode has reference periods of 1 and 50 years (see table 4.8) and the 50 years was chosen as reference since it Eurocode assigns common building structures a design working life of 50 years. However, introducing a time aspect to the simulations is not straightforward, since

it requires that the time-dependent governing loads are described statistically. The time-dependent variable here is the water level, which varies with time and influences the stability. In the study, the water levels are based on provided hydrological data for the area, where the lowest low water (+57.6 m) and the mean flow (+58.2 m) represents characteristic values rather than continuous distributions over time. Because of this the time-dependency could not be properly captured with the available measurements. This is important to keep in mind when comparing the results against the Eurocode values, as it is not clear which reference period the calculated β-values corresponds to.

Throughout the project it is evident that the water level in the Säve-river has a great impact on the probability of failure. The lowest low water (+57.6 m) was selected as the reference water level since it reflects standard industry practice. However, a static water level is a simplification that does not reflect the natural variation of the river, and as shown by the PEM simulations with varying water levels, the choice of water level has a substantial effect on P_f. This is also confirmed by the additional analysis in SLOPE/W where the higher water level of +58.2 m yielded lower values of P_f. For example, P_f is reduced from 10^{-10} to 3×10^{-13} for the case of 3 Layers. This suggests that relying solely on LLW, overestimates the true probability of failure, and should therefore be revised in future applications.

For a more comprehensive study, more data of the piezometric line should be collected and standard deviations should be included, which would also allow the water level to be treated as a stochastic variable with an associated time. This could also help make a meaningful comparison against the 1- and 50-year reference periods. An optional modeling software such as SEEP/W could be used to model the transient flow conditions as an additional parameter. Finally, as the slip surface never reaches the clay layer in the current model, the undrained condition remains irrelevant. Future studies could explore slopes where the clay is critical to stability, and where a proper treatment of varying water levels are implemented.

6.8 Probability of Failure

By reducing the spread of the input data, the probability of failure was lowered to well within acceptable levels. The Nolhaga slope is therefore considered unlikely to fail. The difference between the one-layer and three-layer models is substantial, highlighting the sensitivity of the reliability assessment to the level of detail in the soil model. The variance could theoretically be reduced further by subdividing into additional layers, however this requires careful engineering judgment regarding the representativeness of each sublayer.

All three probabilistic methods show consistent trends, although some differences exist. It is unrealistic to expect identical results from three different computational approaches. With that said, the mean factor of safety obtained from PLAXIS is approximately 3–5% above the values from PEM and SLOPE/W, resulting in a correspondingly lower probability of failure. The cause of this difference has not

been identified and warrants further investigation. It should be noted that the custom Python script was developed for academic purposes and is not intended for use in the industry. For commercial use, we recommend that PLAXIS develops dedicated built-in probabilistic functionality.

7
Conclusion

A probabilistic approach to slope stability analysis has the potential to enable more optimized designs with reduced costs and environmental impact. However, it requires a thorough understanding of statistics as some sources of uncertainty are difficult to quantify. Furthermore, small changes in the assumed distributions or input parameters can have a disproportionately large effect on the estimated probability of failure, placing greater demands on the engineers conducting the analysis. This, in combination with the limited transparency of the built-in probabilistic tools available in the widely used software SLOPE/W, likely contributes to the skepticism towards probabilistic analysis in the industry.

The Point Estimate Method has proven to be a powerful tool, yielding results comparable to Monte Carlo and Latin Hypercube while being considerably more user-friendly. For practitioners new to probabilistic slope stability analysis, PEM is recommended as a starting point, and can be complemented by the PLAXIS simulation framework developed in this thesis. For probabilistic methods to become standard practice, clearer regulatory guidelines must be established. This study also highlights that pore pressure variability is an important source of uncertainty, and must be incorporated as a stochastic variable in a realistic way for the model to reach its full potential.

The Nolhaga slope satisfies the requirements of Eurocode when analyzed using partial coefficients as originally modeled. However, as demonstrated through the PEM, SLOPE/W and PLAXIS simulations, the choice of parameter distribution has a significant influence on the results. Simply assigning theoretical distributions to soil parameters without engineering judgment risks overestimating the probability of failure, potentially leading to over-design with unnecessary costs and environmental impact. By applying well-reasoned engineering assumptions that remain consistent with reality, the variance of the input parameters can be reduced and the requirements can be met. This highlights that probabilistic slope stability analysis requires informed judgment which remains essential throughout the process.

Bibliography

[1] D. V. Griffiths and Gordon A. Fenton. *Probabilistic methods in geotechnical engineering*. Ed. by D. V. Griffiths and Gordon A. Fenton. Udine: Springer Verlag, 2007. ISBN: 9783211733653.

[2] Lovisa Moritz and Magnus Karlsson. *Trafikverkets tekniska råd för geokonstruktioner-TR Geo 13*. Tech. rep. Trafikverket, Feb. 2016. DOI: TDOK2013:0667.

[3] B van Den Eijnden et al. *Reliability-based verification of limit states for geotechnical structures*. English. Tech. rep. Luxembourg: European Commission, Joint Research Centre, Nov. 2024. DOI: 10.2760/1342542. URL: https://data.europa.eu/doi/10.2760/1342542.

[4] Douglas C. Montgomery and George C Runger. *Applied Statistics and Probability for Engineer*. Ed. by Linda Ratts. 6th ed. John Wiley & Sons, 2014. URL: https://kolegite.com/EE_library/books_and_lectures/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0/Applied%20Statistics%20and%20Probability%20for%20Engineering.pdf.

[5] David Johnson. "The triangular distribution as a proxy for the beta distribution in risk analysis". In: (1994). DOI: 10.1111/1467-9884.00091. URL: https://rss.onlinelibrary.wiley.com/doi/10.1111/1467-9884.00091.

[6] Yu Wang, Tengyuan Zhao, and Zijun Cao. "Site-specific probability distribution of geotechnical properties". In: *Computers and Geotechnics* 70 (2015), pp. 159–168. ISSN: 0266-352X. URL: https://www.sciencedirect.com/science/article/pii/S0266352X15001792?ref=pdf_download&fr=RR-2&rr=9cba81149de2eb48.

[7] Gregory B. Baecher and John T. Christian. *Reliability and Statistics in Geotechnical Engineering*. Tech. rep. West Sussex: Wiley, 2003. URL: https://www.researchgate.net/profile/Gregory-Baecher/publication/247385445_Reliability_and_Statistics_in_Geotechnical_Engineering/links/603bef4292851c4ed5a4e861/Reliability-and-Statistics-in-Geotechnical-Engineering.pdf.

[8] Seequent. *Stability Modeling with GeoStudio*. Tech. rep. Seequent Limited, 2022.

[9] Claes Alén. *On Probability in Geotechnics: Random Calculation Models Exemplified On Slope Stability Analysis and Ground-Superstructure Interaction*. Tech. rep. Gothenburg: Chalmers University of Technology, Apr. 1998. DOI: https://publications.lib.chalmers.se/records/fulltext/927/927.pdf. URL: https://publications.lib.chalmers.se/records/fulltext/927/927.pdf.

[10] Göran Sällfors. *Punktskattningsmetoden: En statistisk metod användbar på geotekniska problem*. Tech. rep. Gothenburg: Chalmers Tekniska Högskola, 1990.

[11] J. M.. Duncan, Stephen G.. Wright, and Thomas L.. Brandon. *Soil strength and slope stability*. 3rd ed. New Jersey: John Wiley & Sons Inc., 2014. ISBN: 9781118917954.

[12] SEQUENT. *PLAXIS 2025.1 Scientific Manual PLAXIS 2D*. Tech. rep. The Bentley Subsurface Company, Sept. 2025. URL: https://files.seequent.com/PLAXIS/Manuals/PLAXIS_2D/English/PLAXIS_2D_4_Scientific%20Manual.pdf.

[13] G. R. Liu and S. S. Quek. *The Finite Element Method: A Practical Course*. Oxford: Butterworth-Heinemann, 2003. ISBN: 0750658665. URL: https://www.academia.edu/26052570/Finite_Element_Method_a_practical_course.

[14] D V Griffiths et al. "Probabilistic Slope Stability Analysis by Finite Elements". In: *Journal of Geotechnical and Geoenvironmental Engineering* 130.5 (May 2004), pp. 507–518. ISSN: 1090-0241. DOI: 10.1061/(asce)1090-0241(2004)130:5(507). URL: /doi/pdf/10.1061/%28ASCE%291090-0241%282004%29130%3A5%28507%29?download=true.

[15] Jennifer Nyström. *Markteknisk undersökningsrapport/Geoteknik: Nolhaga, Alingsås*. Tech. rep. Jönköping: Sweco Sverige AB, 2024. URL: www.sweco.se.

[16] Chapman Annlouise and Luke Elliot. *Geoteknisk PM- Alingsås Nolhaga detaljplan*. Tech. rep. Jönköping, July 2022. URL: www.sweco.se.

[17] Göran Sällfors and Lars Rolfsson. *Föreläsning-Statistik för Geotekniker*. Gothenburg, 2026. URL: http://geoforce.argonovautveckling.se/files/user/4.b.%20Punktskattningsmetoden_2.pdf.

[18] Statens Geotekniska Institut. *Användarmanual: CONRAD version 3.1*. Tech. rep. Linköping: Statens Geotekniska Institut, 2010.

[19] Jennifer Nyström et al. *Rev C: MARKTEKNISK UNDERSÖKNINGSRAPPORT/GEOTEKNIK, MUR*. Tech. rep. Jönköping: Sweco Sverige AB, 2026.

[20] Rolf Larsson. *CPT-sondering utrustning-utförande-utvärdering*. Tech. rep. Linköping: Statens Geotekniska Institut, 2015.

[21] Annlouise Elliot, Luke Chapman, and Per Nylander. *Rev D: GEOTEKNISK PM, Alingsås Nolhaga detaljplan*. Tech. rep. Jönköping: Sweco Sverige AB, Feb. 2026. URL: www.sweco.se.

[22] Yongqing Zeng et al. "A case study on soil slope landslide failure and parameter analysis of influencing factors for safety factor based on strength reduction method and orthogonal experimental design". In: *PLOS ONE* 19.5 (May 2024), e0300586. ISSN: 19326203. DOI: 10.1371/JOURNAL.PONE.0300586. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC11095681/.

[23] B.A. Yogatama and B.A Tirta. *Python Application in Geotechnical Engineering Practices*. Tech. rep. Yogyakarta: Simposium Nasional Teknologi Infrastruktur, Jan. 2021.

[24] Swedish Standards Institute (SIS). *SVENSK STANDARD SS-EN 1997-1:2004. Eurocode 7 - Geotechnical design - Part 1: General rules*. Tech. rep. Stockholm: SIS (Svenska Institutet för Standarder), Nov. 2024. URL: https://

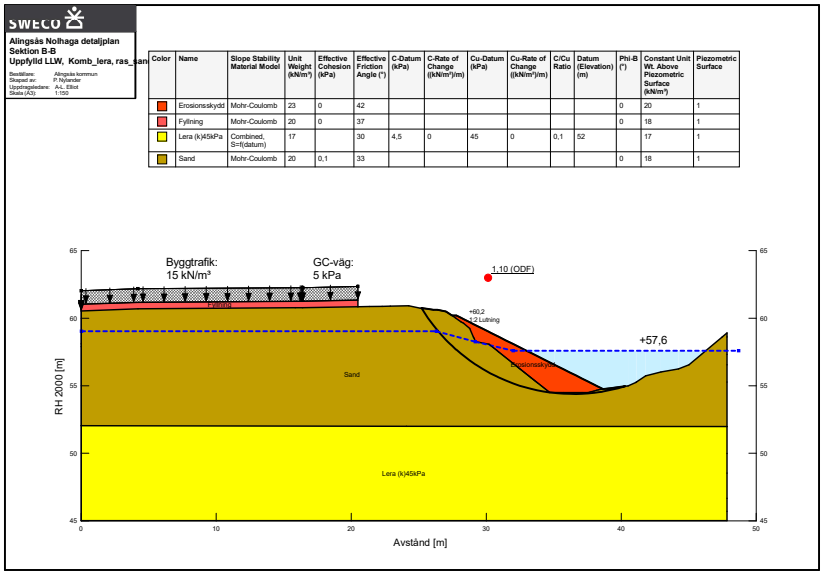
www.sis.se/produkter/anlaggningsarbete/markarbete-utgravning-grundlaggning-arbete-under-jord/ss-en-1997-12024/.

[25] Swedish Standards Institute (SIS). *SVENSK STANDARD SS-EN 1990:2002 Eurocode - Basis of structural design*. Tech. rep. Stockholm: Swedish Standards Institute (SIS), 2002.

A

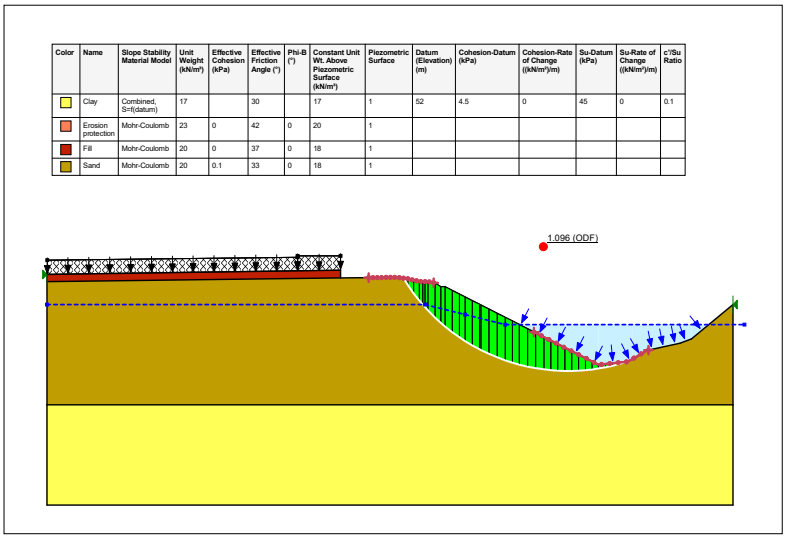
SLOPE/W sections

Reference model: Slope stability analysis of section B-B derived from Sweco PM, SF = 1.10.



Color	Name	Slope Stability Material Model	Unit Weight (kN/m³)	Effective Cohesion (kPa)	Effective Friction Angle (°)	C-Datum (kPa)	C-Rate of Change ((kN/m²)/m)	Cu-Datum (kPa)	Cu-Rate of Change ((kN/m²)/m)	C/Cu Ratio	Datum (Elevation) (m)	Phi-B (°)	Constant Unit Wt. Above Piezometric Surface (kN/m³)	Piezometric Surface
	Erosionsskydd	Mohr-Coulomb	23	0	42							0	20	1
	Fyllning	Mohr-Coulomb	20	0	37							0	18	1
	Lera (k)45kPa	Combined, S=f(datum)	17		30	4,5	0	45	0	0,1	52		17	1
	Sand	Mohr-Coulomb	20	0,1	33							0	18	1

Slope stability analysis of section B-B created from the reference model, SF = 1.096



Color	Name	Slope Stability Material Model	Unit Weight (kN/m³)	Effective Cohesion (kPa)	Effective Friction Angle (°)	Phi-B (°)	Constant Unit Wt. Above Piezometric Surface (kN/m³)	Piezometric Surface	Datum (Elevation) (m)	Cohesion-Datum (kPa)	Cohesion-Rate of Change ((kN/m²)/m)	Su-Datum (kPa)	Su-Rate of Change ((kN/m²)/m)	c'/Su Ratio
	Clay	Combined, S=f(datum)	17		30		17	1	52	4.5	0	45	0	0.1
	Erosion protection	Mohr-Coulomb	23	0	42	0	20	1						
	Fill	Mohr-Coulomb	20	0	37	0	18	1						
	Sand	Mohr-Coulomb	20	0.1	33	0	18	1						

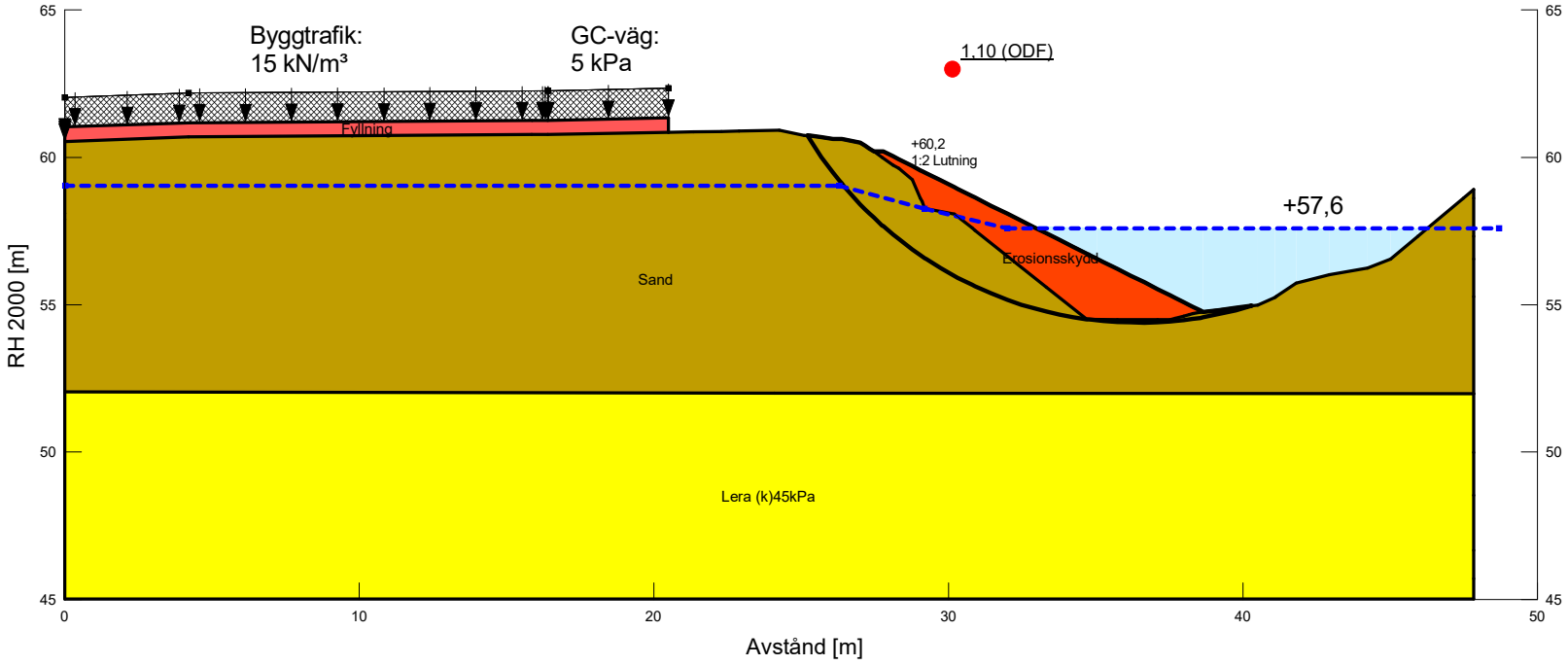
A. SLOPE/W sections

SWECO

Alingsås Nolhaga detaljplan
Sektion B-B
Uppfylld LLW, Komb_lera, ras_san

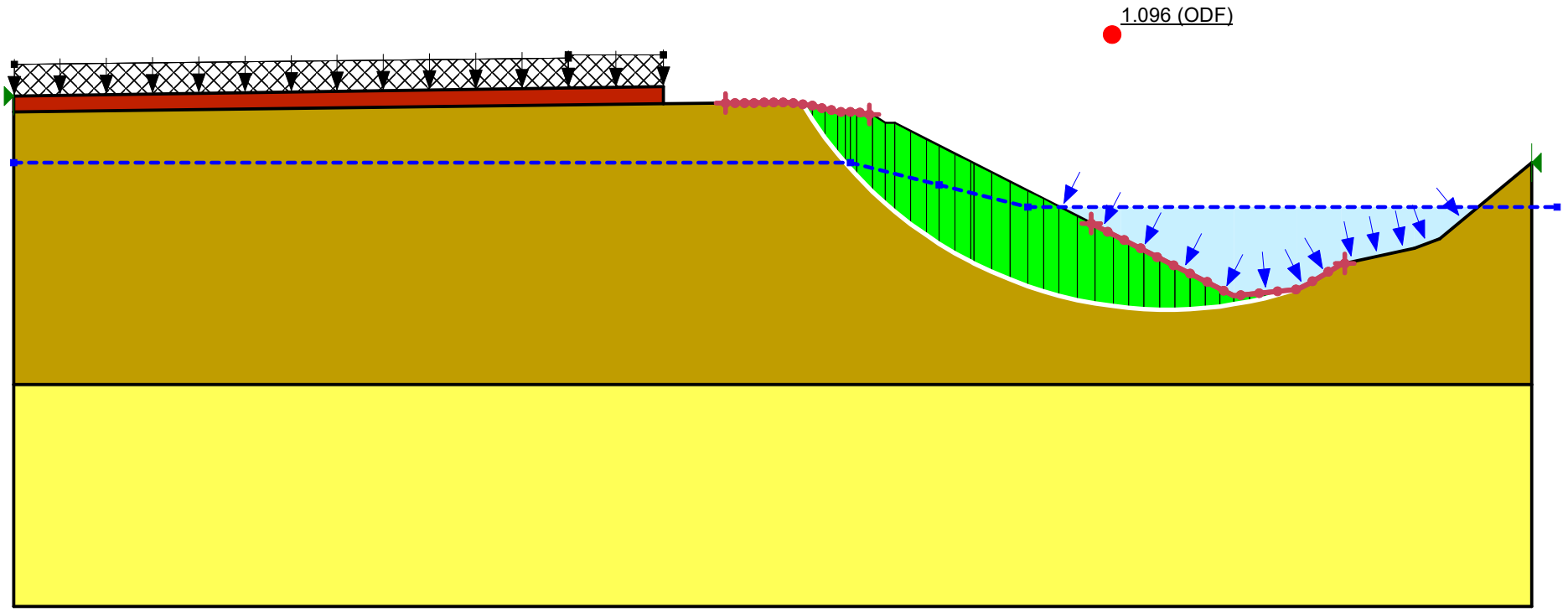
Beställare: Alingsås kommun
Skapad av: P. Nylander
Uppdragsledare: A-L. Elliot
Skala (A3): 1:150

Color	Name	Slope Stability Material Model	Unit Weight (kN/m³)	Effective Cohesion (kPa)	Effective Friction Angle (°)	C-Datum (kPa)	C-Rate of Change ((kN/m²)/m)	Cu-Datum (kPa)	Cu-Rate of Change ((kN/m²)/m)	C/Cu Ratio	Datum (Elevation) (m)	Phi-B (°)	Constant Unit Wt. Above Piezometric Surface (kN/m³)	Piezometric Surface
	Erosionsskydd	Mohr-Coulomb	23	0	42							0	20	1
	Fyllning	Mohr-Coulomb	20	0	37							0	18	1
	Lera (k)45kPa	Combined, S=f(datum)	17		30	4,5	0	45	0	0,1	52		17	1
	Sand	Mohr-Coulomb	20	0,1	33							0	18	1



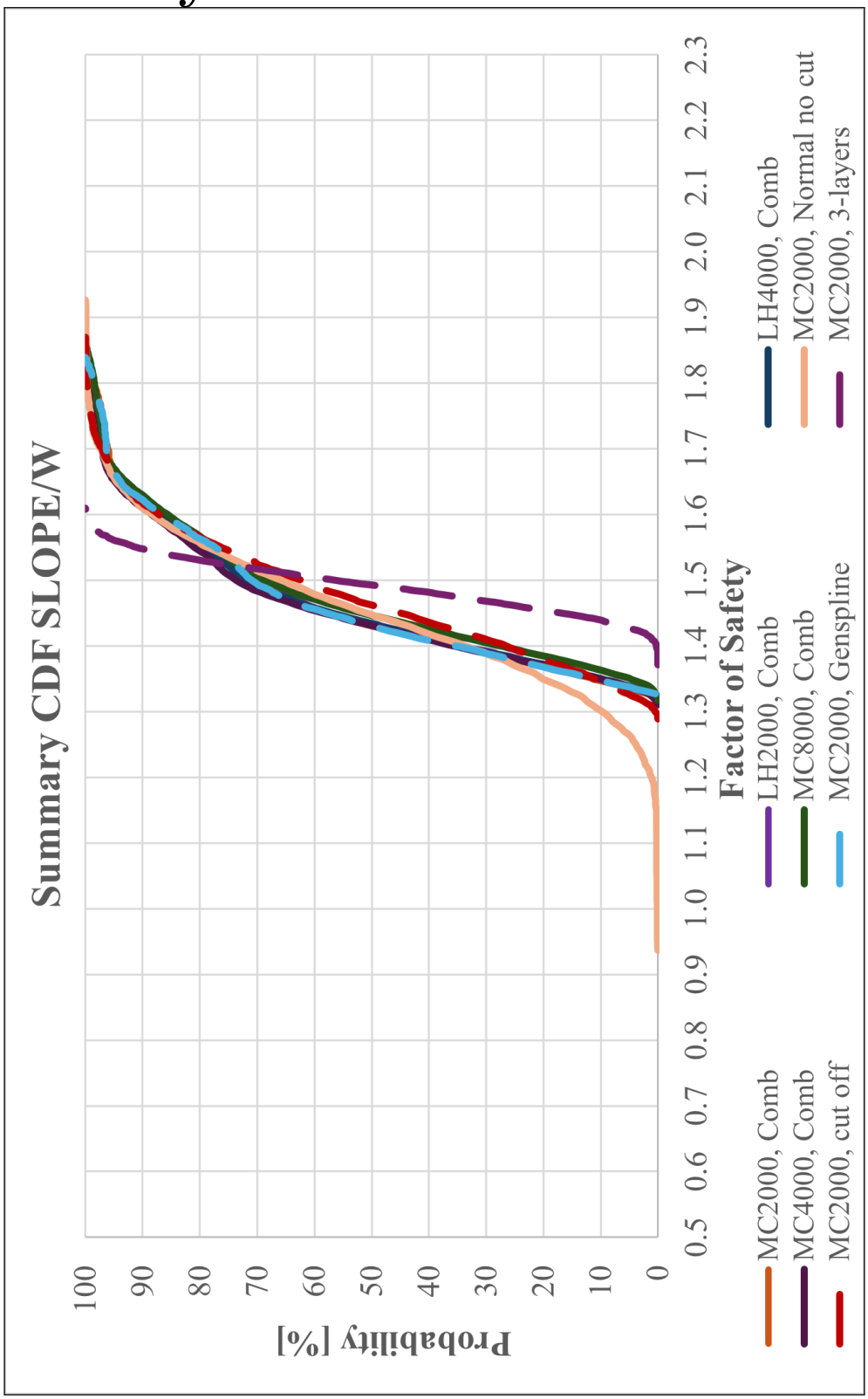
Slope stability analysis of section B-B created from the reference model, SF = 1.096

Color	Name	Slope Stability Material Model	Unit Weight (kN/m³)	Effective Cohesion (kPa)	Effective Friction Angle (°)	Phi-B (°)	Constant Unit Wt. Above Piezometric Surface (kN/m³)	Piezometric Surface	Datum (Elevation) (m)	Cohesion-Datum (kPa)	Cohesion-Rate of Change ((kN/m²)/m)	Su-Datum (kPa)	Su-Rate of Change ((kN/m²)/m)	c'/Su Ratio
	Clay	Combined, S=f(datum)	17		30		17	1	52	4.5	0	45	0	0.1
	Erosion protection	Mohr-Coulomb	23	0	42	0	20	1						
	Fill	Mohr-Coulomb	20	0	37	0	18	1						
	Sand	Mohr-Coulomb	20	0.1	33	0	18	1						



B

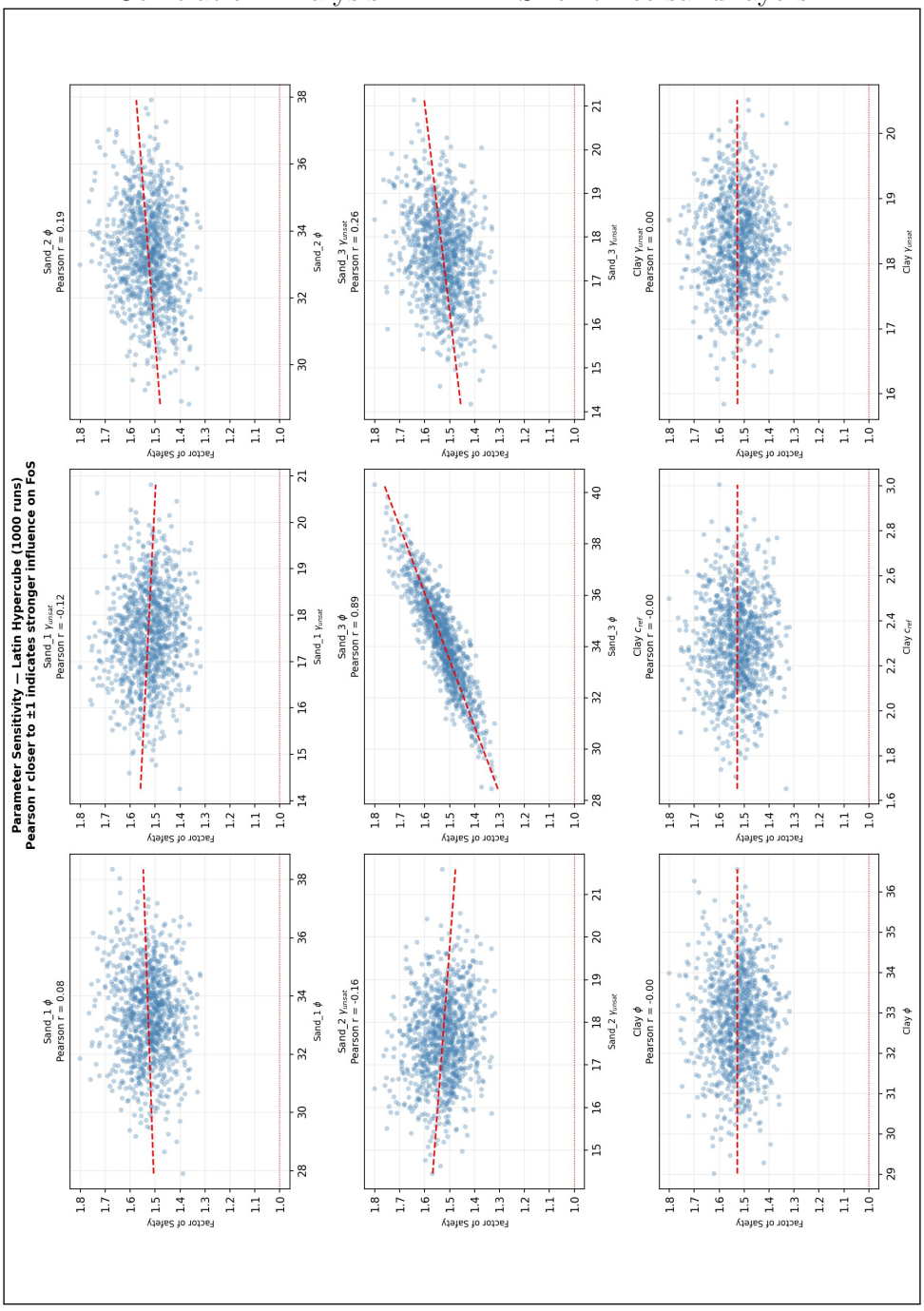
Summary Cumulative Distribution



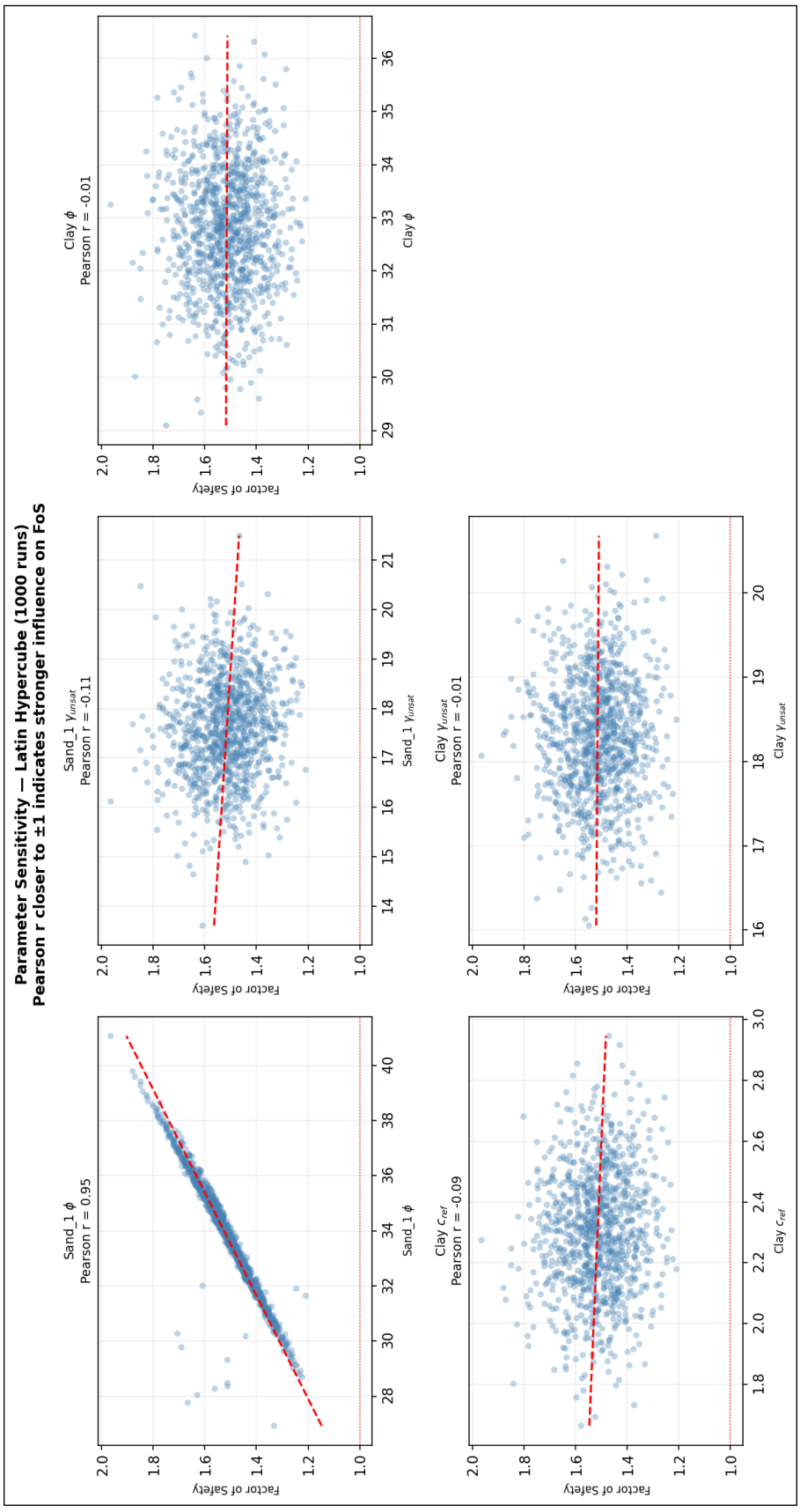
C

Results from Correlation Analysis in PLAXIS

Correlation Analysis in PLAXIS for three sand layers

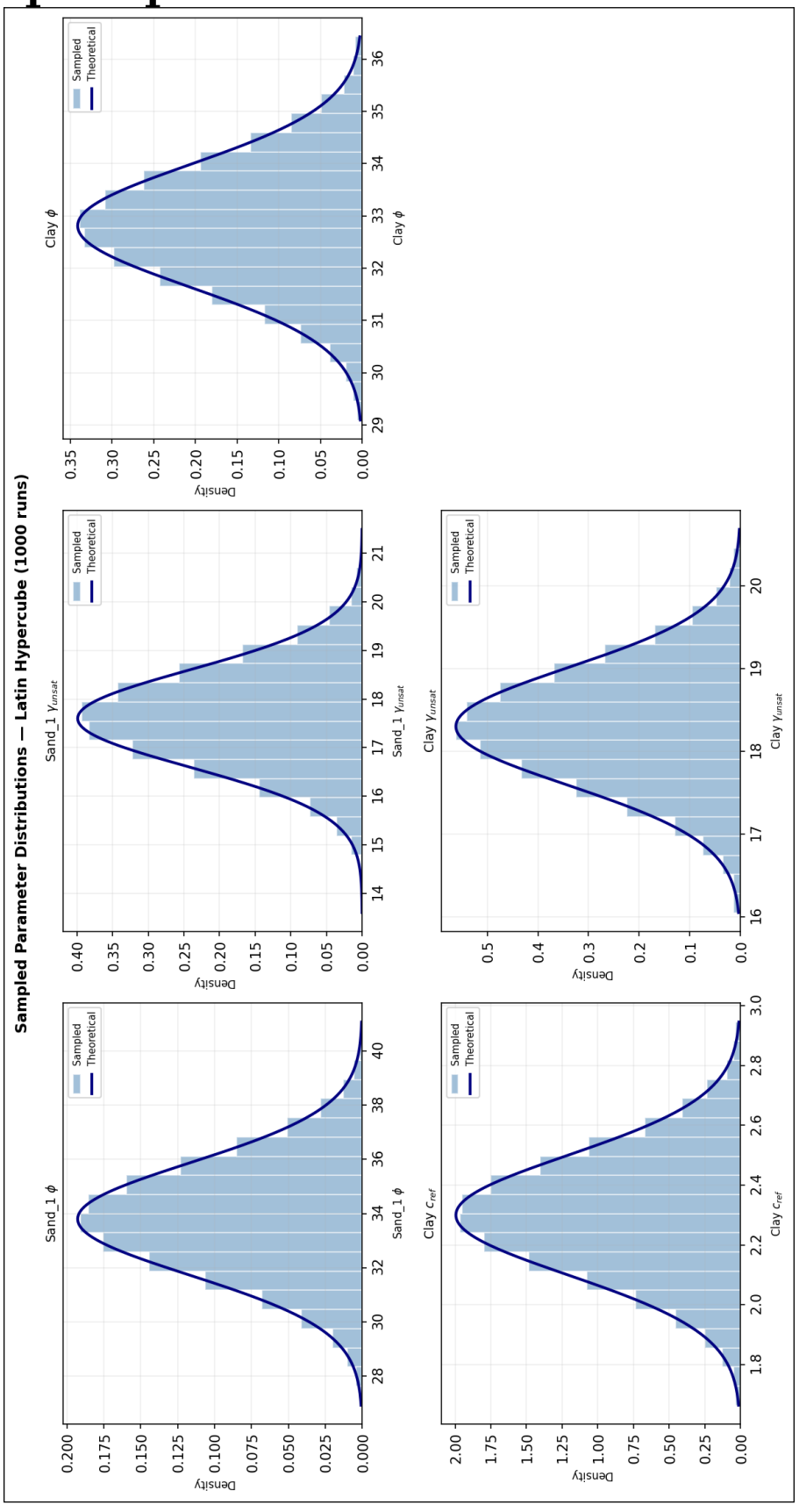


Correlation Analysis in PLAXIS for one sand layer



D

Input parameters for one sand



E

Python script — simple version

Probabilistic Slope Stability Analysis — PLAXIS 2D

Teaching version (simple)

This notebook runs a small probabilistic slope stability analysis with PLAXIS 2D. It is written for students who are **new to Python**. You do not need to write any code — you only fill in a few values in **Cell 1** and **Cell 2**, then run every cell from top to bottom.

What the notebook does: for each simulation run it picks a random combination of soil parameters (drawn from the distributions you define), runs the PLAXIS calculation, and reads the Factor of Safety (FoS). After all runs it shows the distribution of FoS values and estimates the probability of failure and the reliability index β.

This simple version is meant for a **small number of runs (up to about 50)** so it finishes quickly and is easy to follow. For large production runs (1000+), use the **full version** of the notebook instead.

Before you begin

1. **Open PLAXIS 2D** and load the slope model you want to analyse.
2. **Turn on the scripting server** in PLAXIS 2D under `Expert → Configure remote scripting server`. Write down the input port, output port, and password shown there.
3. **Run this notebook on the same computer** as PLAXIS 2D. (PLAXIS runs on Windows.)
4. **Edit Cell 1 and Cell 2** with your settings. These are the only two cells you need to change.
5. **Replace every placeholder** written in capitals or angle brackets — for example `YOUR_PASSWORD`, `C:\Path\To\...`, `<version>` — with the values for your own installation.

How to run

Click the first cell, then press `Shift + Enter` to run it and move to the next. Repeat all the way down. **Run the cells in order.** If something goes wrong, the most common cause is a placeholder that was not replaced, or PLAXIS not being open with the scripting server turned on.

Cell 1 — Settings

Fill in the values below, then run the cell (`Shift + Enter`). Lines you normally need to change are marked with `# <-- CHANGE THIS`.

In [ ]:
```
from pathlib import Path
from datetime import datetime as _dt

# ============================================================================
#  PLAXIS CONNECTION
#  These three values come from PLAXIS 2D:
#  Expert -> Configure remote scripting server
# ============================================================================
PORT_IN   = 10000
PORT_OUT  = 10001
PASSWORD  = "YOUR_PASSWORD"        # <-- CHANGE THIS (copy from PLAXIS)

# ============================================================================
#  PHASES
#  SAFETY_PHASE     : the name of the phi/c reduction (Safety) phase
#  FULL_PHASE_CHAIN : every phase, in the order PLAXIS calculates them
#  These names must match your model exactly (case-sensitive).
# ============================================================================
SAFETY_PHASE     = "Phase_3"                              # <-- CHANGE if needed
FULL_PHASE_CHAIN = ["InitialPhase", "Phase_1", "Phase_3"] # <-- CHANGE if needed

# ============================================================================
#  MESH
#  How fine the finite element mesh is.
#  0.1 = very coarse (fast) | 0.05 = medium | 0.01 = fine (slow)
# ============================================================================
MESH_COARSENESS = 0.05

# ============================================================================
#  SIMULATION
#  N_SIMULATIONS   : how many runs (keep this small in the simple version,
```

```
#                   e.g. 20-50, so it finishes quickly)
#  SAMPLING_METHOD : "monte_carlo"     = plain random sampling
#                    "latin_hypercube" = better coverage with fewer runs
#  N_SAND_LAYERS   : number of sand layers (used only for naming the output folder)
#  RANDOM_SEED     : None = different result each time
#                    a number (e.g. 42) = same result every time (reproducible)
# ============================================================================
N_SIMULATIONS   = 30                  # <-- CHANGE THIS (max ~50 in this version)
SAMPLING_METHOD = "latin_hypercube"   # <-- "monte_carlo" or "latin_hypercube"
N_SAND_LAYERS   = 3                   # <-- CHANGE if needed
RANDOM_SEED     = None                # <-- set a number for reproducible results

# ============================================================================
#  TIMING
#  SLEEP_AFTER_VIEW : seconds to wait after opening Output before reading FoS.
#                     Increase if Output is slow to load on your computer.
#  SETTLE_SLEEP     : short pause between runs.
# ============================================================================
SLEEP_AFTER_VIEW = 5
SETTLE_SLEEP     = 2

# ============================================================================
#  OUTPUT READ RETRY
#  If PLAXIS Output is briefly unresponsive, the FoS read is tried once more
#  after waiting this many seconds. This keeps a single slow read from
#  stopping the whole simulation.
# ============================================================================
FOS_RETRY_WAIT = 120

# ============================================================================
#  WHERE TO SAVE RESULTS
#  One Excel file and the figures are written here at the very end.
# ============================================================================
_base_dir     = Path(r"C:\Path\To\Output")   # <-- CHANGE THIS
RUN_TIMESTAMP = _dt.now().strftime("%Y-%m-%d_%H-%M")

print("Settings loaded.")
print(f"  Simulations     : {N_SIMULATIONS}")
print(f"  Sampling method : {SAMPLING_METHOD}")
print(f"  Output folder   : {_base_dir}")
```

Cell 2 — Soil Parameter Distributions

Define the statistical distribution for each soil parameter. Set a parameter to `None` to keep it fixed at the model value.

The `GAMMA_SAT_OFFSET` defines the difference between saturated and unsaturated unit weight. If `gammaUnsat` is randomised, `gammaSat` is updated automatically.

If `gammaUnsat` is set to `None` for all materials, the initial stress phases are calculated only once before the loop. This is valid because the stress distribution does not change when unit weight is fixed, and significantly reduces the total runtime.

Available distribution types

```
# Normal
{"type": "normal", "mean": 33.8, "std": 2.07}

# Normal with truncation (values outside range are rejected and redrawn)
{"type": "normal", "mean": 33.8, "std": 2.07, "min": 25.0, "max": 42.0}

# Normal with 90% confidence interval (cutoff at mean ± 1.645 * std)
{"type": "normal", "mean": 33.8, "std": 2.07, "confidence": 0.90}

# Lognormal (enter real-space mean and std)
{"type": "lognormal", "mean": 2.3, "std": 0.5}

# Triangular
{"type": "triangular", "low": 28.0, "mode": 33.8, "high": 40.0}

# Uniform
{"type": "uniform", "low": 28.0, "high": 40.0}

# Custom (fitted KDE from measured data points)
{"type": "custom", "data": [32.1, 33.4, 34.2, 35.0, 33.8]}

# Fixed at model value
None
```

In [ ]:

```
# ============================================================================
#  UNIT WEIGHT OFFSET
#  gammaSat = gammaUnsat + GAMMA_SAT_OFFSET  (kN/m3)
#  Applied automatically when gammaUnsat is randomised.
# ============================================================================
```

```
GAMMA_SAT_OFFSET = 1.0

# ============================================================================
#  SAND_1
#  Material name must match exactly the name in the PLAXIS model (case-sensitive)
# ============================================================================
SAND_1 = {
    "phi": {
        "type": "normal",
        "mean": 33.3,
        "std":   1.58,
        "min":  23.5,
        "max":  44.1,
    },
    "cRef":       None,
    "gammaUnsat": None,
}

# ============================================================================
#  SAND_2
# ============================================================================
SAND_2 = {
    "phi": {
        "type": "normal",
        "mean": 33.4,
        "std":   1.50,
        "min":  23.5,
        "max":  44.1,
    },
    "cRef":       None,
    "gammaUnsat": None,
}

# ============================================================================
#  SAND_3
# ============================================================================
SAND_3 = {
    "phi": {
        "type": "normal",
        "mean": 34.2,
        "std":   1.92,
        "min":  23.5,
        "max":  44.1,
    },
    "cRef":       None,
    "gammaUnsat": None,
}

# ============================================================================
#  CLAY
# ============================================================================
CLAY = {
    "phi": {
        "type": "normal",
        "mean": 32.9,
        "std":   1.28,
        "min":  26.5,
        "max":  39.3,
    },
    "cRef": {
        "type": "normal",
        "mean": 2.3,
        "std":  0.1,
        "min":  1.8,
        "max":  2.8,
    },
    "gammaUnsat": None,
}

# Bundle all materials -- do not edit
MATERIALS = {
    "Sand_1": SAND_1,
    "Sand_2": SAND_2,
    "Sand_3": SAND_3,
    "Clay":   CLAY,
}

# ============================================================================
#  RESOLVE CONFIDENCE INTERVAL CUTOFFS
#  Converts {"confidence": 0.90} to explicit min/max at mean ± 1.645 * std
# ============================================================================
import math
_CONFIDENCE_Z = {0.90: 1.645, 0.95: 1.960, 0.99: 2.576}
```

```
for _mp in MATERIALS.values():
    for _cfg in _mp.values():
        if isinstance(_cfg, dict) and "confidence" in _cfg:
            _z = _CONFIDENCE_Z.get(_cfg["confidence"], 1.645)
            _cfg["min"] = _cfg["mean"] - _z * _cfg["std"]
            _cfg["max"] = _cfg["mean"] + _z * _cfg["std"]
            del _cfg["confidence"]

# ============================================================================
#  BUILD OUTPUT FOLDER NAME
#  The folder name encodes the sampling method, number of runs, number of
#  sand layers, and the distribution type used for each randomised parameter.
#  Example: lhs_200runs_3sand_N_phi_N_cRef
# ============================================================================
_DIST_SHORT = {"normal": "N", "normal_ci": "NCI", "lognormal": "LN",
               "triangular": "T", "uniform": "U", "custom": "KDE"}
_METH_SHORT = {"latin_hypercube": "lhs", "monte_carlo": "mc"}

_meth_s      = _METH_SHORT.get(SAMPLING_METHOD, SAMPLING_METHOD)
_params_seen = {}
for _mp in MATERIALS.values():
    for _pname, _cfg in _mp.items():
        if _cfg is not None and _pname not in _params_seen:
            _params_seen[_pname] = _DIST_SHORT.get(_cfg["type"], _cfg["type"])
_param_str   = "_".join(f"{v}_{k}" for k, v in _params_seen.items())
_tag         = f"{_meth_s}_{N_SIMULATIONS}runs_{N_SAND_LAYERS}sand_{_param_str}"
RESULTS_DIR  = _base_dir / _tag
RESULTS_XLSX = RESULTS_DIR / f"mc_results_{RUN_TIMESTAMP}.xlsx"
LOG_FILE     = RESULTS_DIR / f"mc_log_{RUN_TIMESTAMP}.log"

# ============================================================================
#  SUMMARY PRINTOUT
# ============================================================================
print(f"Output folder    : {RESULTS_DIR}")
print(f"gammaSat offset  : gammaUnsat + {GAMMA_SAT_OFFSET} kN/m3")
print()
print("Soil parameter distributions:")
for mat_name, params in MATERIALS.items():
    print(f"  {mat_name}:")
    for param, cfg in params.items():
        if cfg is None:
            print(f"    {param:<14}: Fixed at model value")
        elif cfg["type"] == "normal":
            lo = f"{cfg['min']:.2f}" if "min" in cfg else "-inf"
            hi = f"{cfg['max']:.2f}" if "max" in cfg else "+inf"
            print(f"    {param:<14}: Normal       mean={cfg['mean']:>6}  std={cfg['std']:>5}  [{lo}, {hi}]")
        elif cfg["type"] == "lognormal":
            print(f"    {param:<14}: Lognormal    mean={cfg['mean']:>6}  std={cfg['std']:>5}")
        elif cfg["type"] == "triangular":
            print(f"    {param:<14}: Triangular   low={cfg['low']}  mode={cfg['mode']}  high={cfg['high']}")
        elif cfg["type"] == "uniform":
            print(f"    {param:<14}: Uniform      low={cfg['low']}  high={cfg['high']}")
        elif cfg["type"] == "custom":
            print(f"    {param:<14}: Custom KDE   {len(cfg['data'])} data points")
    print()
```

Cell 2b — Output file names (no editing needed)

Builds the output folder name automatically from your settings and creates the folder. Just run it.

In [ ]:
```
# Build an automatic, descriptive folder name from the settings above.
_tag = (f"{'lhs' if SAMPLING_METHOD == 'latin_hypercube' else 'mc'}"
        f"_{N_SIMULATIONS}runs_{N_SAND_LAYERS}sand")

RESULTS_DIR  = _base_dir / _tag
RESULTS_XLSX = RESULTS_DIR / f"results_{_tag}_{RUN_TIMESTAMP}.xlsx"
LOG_FILE     = RESULTS_DIR / f"log_{_tag}_{RUN_TIMESTAMP}.txt"

RESULTS_DIR.mkdir(parents=True, exist_ok=True)
print(f"Results will be saved in:\n  {RESULTS_DIR}")
```

Cell 3 — Load libraries and helper functions (no editing needed)

This cell loads the Python libraries and defines the helper functions used later (sampling the parameters, reading the Factor of Safety from PLAXIS, and saving the results). You do not need to change anything here — just run it.

In [ ]:

```
import sys, time, logging, subprocess
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
from scipy import stats
from scipy.stats import norm, lognorm, qmc

# Install any missing packages automatically
for pkg in ["pandas", "matplotlib", "scipy", "openpyxl"]:
    try:
        __import__(pkg)
    except ImportError:
        print(f"Installing {pkg} ...")
        subprocess.check_call([sys.executable, "-m", "pip", "install", pkg])

# Add the PLAXIS scripting library to the Python path
PLAXIS_PATH = r"C:\ProgramData\Seequent\PLAXIS Python Distribution V<n>\python\Lib\site-packages"  # <-- CHANGE version number
if PLAXIS_PATH not in sys.path:
    sys.path.insert(0, PLAXIS_PATH)
from plxscripting.easy import new_server

# Write log messages to both the screen and a log file
for h in logging.root.handlers[:]:
    logging.root.removeHandler(h)
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s  %(levelname)-8s  %(message)s",
    handlers=[logging.FileHandler(LOG_FILE, encoding="utf-8"),
              logging.StreamHandler(sys.stdout)],
)

# ============================================================================
#  SAMPLING FUNCTIONS
#  Turn the distributions you defined in Cell 2 into random parameter values.
#  Supports: normal (optionally truncated or with a confidence interval),
#            lognormal, triangular, uniform, and custom (fitted from data).
# ============================================================================

def _lognormal_params(mean, std):
    """Convert real-space mean and std to lognormal mu and sigma."""
    v = std**2
    return np.log(mean**2 / np.sqrt(v + mean**2)), np.sqrt(np.log(1 + v / mean**2))

def _get_active_params(materials):
    """Return ordered list of (material, parameter, config) for active parameters."""
    return [(mat, p, cfg)
            for mat, params in materials.items()
            for p, cfg in params.items()
            if cfg is not None]

def _ppf(cfg, u):
    """Transform a uniform sample u in [0, 1] into a parameter value using the
    inverse CDF of the chosen distribution."""
    t    = cfg["type"]
    vmin = cfg.get("min", None)
    vmax = cfg.get("max", None)

    if t == "normal":
        m, s = cfg["mean"], cfg["std"]
        u_lo = norm.cdf(vmin, m, s) if vmin is not None else 0.0
        u_hi = norm.cdf(vmax, m, s) if vmax is not None else 1.0
        return float(norm.ppf(u_lo + u * (u_hi - u_lo), m, s))

    elif t == "lognormal":
        mu, sigma = _lognormal_params(cfg["mean"], cfg["std"])
        u_lo = lognorm.cdf(vmin, sigma, scale=np.exp(mu)) if vmin is not None else 0.0
        u_hi = lognorm.cdf(vmax, sigma, scale=np.exp(mu)) if vmax is not None else 1.0
        return float(lognorm.ppf(u_lo + u * (u_hi - u_lo), sigma, scale=np.exp(mu)))

    elif t == "triangular":
        from scipy.stats import triang
        lo, mode, hi = cfg["low"], cfg["mode"], cfg["high"]
        c = (mode - lo) / (hi - lo)
        return float(triang.ppf(u, c, loc=lo, scale=hi - lo))

    elif t == "uniform":
        return float(cfg["low"] + u * (cfg["high"] - cfg["low"]))

    elif t == "custom":
        kde = stats.gaussian_kde(np.array(cfg["data"], dtype=float))
        for _ in range(1000):
```

```
            v = float(kde.resample(1)[0][0])
            if v > 0 and (vmin is None or v >= vmin) and (vmax is None or v <= vmax):
                return v
        return float(np.mean(cfg["data"]))

    else:
        raise ValueError(f"Unknown distribution type: '{t}'.")

def generate_samples(materials, n, rng, method):
    """Generate all n parameter combinations before the simulation starts."""
    active   = _get_active_params(materials)
    n_params = len(active)

    if method == "latin_hypercube":
        lhs     = qmc.LatinHypercube(d=n_params, seed=RANDOM_SEED)
        u_array = lhs.random(n=n)
    else:
        u_array = rng.uniform(0, 1, size=(n, n_params))

    samples = []
    for i in range(n):
        sample = {mat: {} for mat in materials}
        for j, (mat_name, param, cfg) in enumerate(active):
            sample[mat_name][param] = _ppf(cfg, u_array[i, j])
        samples.append(sample)
    return samples

# ============================================================================
#  RESULTS FUNCTIONS
#  Build the summary table and save everything to one Excel file at the end.
# ============================================================================

def _build_summary_df(records):
    """Summary table: FoS statistics, reliability index beta and probability of
    failure (normal and lognormal), and the input distribution of each parameter."""
    from scipy.stats import norm as _norm
    df_all  = pd.DataFrame(records)
    df_conv = df_all[df_all["converged"]].copy()
    rows    = []

    if len(df_conv) > 0:
        fos     = df_conv["FoS"].values
        mu      = fos.mean()
        sigma   = fos.std()
        V       = sigma / mu
        beta_n  = (mu - 1.0) / sigma
        beta_ln = np.log(mu / np.sqrt(1 + V**2)) / np.sqrt(np.log(1 + V**2))
        pf_n    = _norm.cdf(-beta_n)  * 100
        pf_ln   = _norm.cdf(-beta_ln) * 100
        rows += [
            {"Category": "Results",     "Parameter": "Sampling method",   "Value": SAMPLING_METHOD},
            {"Category": "Results",     "Parameter": "Simulations",       "Value": N_SIMULATIONS},
            {"Category": "Results",     "Parameter": "Converged runs",    "Value": len(df_conv)},
            {"Category": "Results",     "Parameter": "Mean FoS",          "Value": round(mu, 3)},
            {"Category": "Results",     "Parameter": "Std FoS",           "Value": round(sigma, 3)},
            {"Category": "Results",     "Parameter": "Min FoS",           "Value": round(fos.min(), 3)},
            {"Category": "Results",     "Parameter": "Max FoS",           "Value": round(fos.max(), 3)},
            {"Category": "Results",     "Parameter": "",                  "Value": ""},
            {"Category": "Reliability", "Parameter": "beta (Normal)",     "Value": round(beta_n, 3)},
            {"Category": "Reliability", "Parameter": "P(f) Normal [%]",   "Value": f"{pf_n:.4e}"},
            {"Category": "Reliability", "Parameter": "beta (Lognormal)",  "Value": round(beta_ln, 3)},
            {"Category": "Reliability", "Parameter": "P(f) Lognormal [%]", "Value": f"{pf_ln:.4e}"},
            {"Category": "Reliability", "Parameter": "",                  "Value": ""},
        ]

    for mat_name, params in MATERIALS.items():
        for param, cfg in params.items():
            if cfg is None:
                rows.append({"Category": mat_name, "Parameter": param, "Value": "Fixed (model value)"})
            elif cfg["type"] == "normal":
                rows.append({"Category": mat_name, "Parameter": param,
                             "Value": f"Normal  mean={cfg['mean']}  std={cfg['std']}  [{cfg.get('min','-
inf')}, {cfg.get('max','+inf')}]"})
            elif cfg["type"] == "lognormal":
                rows.append({"Category": mat_name, "Parameter": param,
                             "Value": f"Lognormal  mean={cfg['mean']}  std={cfg['std']}"})
            elif cfg["type"] == "triangular":
                rows.append({"Category": mat_name, "Parameter": param,
                             "Value": f"Triangular  low={cfg['low']}  mode={cfg['mode']}  high=
{cfg['high']}"})
            elif cfg["type"] == "uniform":
                rows.append({"Category": mat_name, "Parameter": param,
                             "Value": f"Uniform  low={cfg['low']}  high={cfg['high']}"})
```

```
            elif cfg["type"] == "custom":
                rows.append({"Category": mat_name, "Parameter": param,
                             "Value": f"Custom KDE  ({len(cfg['data'])} data points)"})
    return pd.DataFrame(rows)

def save_to_excel(records, filepath):
    """Save all results to one Excel file with three sheets."""
    df_all     = pd.DataFrame(records)
    df_conv    = df_all[df_all["converged"]].copy()
    df_failed  = df_conv[df_conv["FoS"] < 1.0].copy() if len(df_conv) > 0 else pd.DataFrame()
    df_summary = _build_summary_df(records)
    with pd.ExcelWriter(filepath, engine="openpyxl") as writer:
        df_all.to_excel(writer, sheet_name="All results", index=False)
        if len(df_failed) > 0:
            df_failed.to_excel(writer, sheet_name="FoS below 1.0", index=False)
        else:
            pd.DataFrame({"Note": ["No runs with FoS < 1.0"]}).to_excel(
                writer, sheet_name="FoS below 1.0", index=False)
        df_summary.to_excel(writer, sheet_name="Summary", index=False)

# ============================================================================
#  READING THE FACTOR OF SAFETY FROM PLAXIS OUTPUT
#  After PLAXIS finishes a calculation, the result is read from the Output
#  program. The code waits until Output shows the latest calculation step
#  (so it does not read an old result), then reads SumMsf = the Factor of Safety.
#  If the read fails once, it waits and tries a second time before giving up
#  on that single run.
# ============================================================================

def _try_read_fos(g_i, safety_phase_obj):
    """One attempt to read FoS. Returns (fos, success)."""
    try:
        g_i.view(safety_phase_obj)
        time.sleep(SLEEP_AFTER_VIEW)

        target_step = f"Step_{safety_phase_obj.LastStep.value}"
        synced      = False
        for attempt in range(60):
            _, g_o    = new_server("localhost", PORT_OUT, password=PASSWORD)
            out_phase = getattr(g_o, SAFETY_PHASE)
            out_last  = out_phase.Steps[-1].Name.value
            if out_last == target_step:
                logging.info(f"    Output synced after {attempt + 1}s ({target_step})")
                synced = True
                break
            time.sleep(1)

        if not synced:
            logging.warning(f"    Output not synced (got {out_last}, expected {target_step}) -- reading
anyway")

        fos = out_phase.Reached.SumMsf.value
        return fos, True
    except Exception:
        return None, False

def read_fos_with_timeout(g_i, safety_phase_obj):
    """Read FoS, with one automatic retry after FOS_RETRY_WAIT seconds."""
    fos, ok = _try_read_fos(g_i, safety_phase_obj)
    if ok and fos is not None and fos > 0:
        return fos

    logging.warning(f"    FoS read failed -- waiting {FOS_RETRY_WAIT}s before retry ...")
    time.sleep(FOS_RETRY_WAIT)

    fos, ok = _try_read_fos(g_i, safety_phase_obj)
    if ok and fos is not None and fos > 0:
        logging.info("    Retry successful.")
        return fos

    raise RuntimeError("FoS could not be read after retry.")

print("Libraries loaded and helper functions defined.")
```

Cell 4 — Connect to PLAXIS and build the mesh

Connects to your running PLAXIS model, checks that all the materials and phases named in Cell 1 and Cell 2 really exist, and builds the finite element mesh once.

If this cell fails: make sure PLAXIS 2D is open with the model loaded and the scripting server turned on, and that the

names in Cell 1/Cell 2 match the model.

In [ ]:
```
logging.info(f"Connecting to PLAXIS 2D on localhost:{PORT_IN} ...")
s_i, g_i = new_server("localhost", PORT_IN, password=PASSWORD)
logging.info("Connection established.")

# Verify that all defined materials exist in the model
mat_lookup = {m.Name.value: m for m in g_i.Materials}
for mat_name in MATERIALS:
    if mat_name not in mat_lookup:
        raise RuntimeError(
            f"Material '{mat_name}' not found in the PLAXIS model.\n"
            f"Available materials: {list(mat_lookup.keys())}\n"
            f"Check that the name in Cell 2 matches exactly (case-sensitive)."
        )
    logging.info(f"Found material: '{mat_name}'")

# Verify that all phases exist
g_i.gotostages()
phase_lookup     = {ph.Name.value: ph for ph in g_i.Phases}
safety_phase_obj = phase_lookup.get(SAFETY_PHASE)
if safety_phase_obj is None:
    raise RuntimeError(
        f"Safety phase '{SAFETY_PHASE}' not found.\n"
        f"Available phases: {list(phase_lookup.keys())}"
    )
for name in FULL_PHASE_CHAIN:
    if name not in phase_lookup:
        raise RuntimeError(f"Phase '{name}' not found. Check FULL_PHASE_CHAIN in Cell 1.")
logging.info(f"Found all phases: {FULL_PHASE_CHAIN}")

# Generate the finite element mesh
logging.info(f"Generating mesh (coarseness = {MESH_COARSENESS}) ...")
g_i.gotomesh()
g_i.mesh(MESH_COARSENESS)
logging.info("Mesh generated. Ready to start simulation.")
```

Cell 5 — Run the simulation

This is the main loop. It first generates all the random parameter combinations, then for each run it sends the parameters to PLAXIS, calculates, and reads the Factor of Safety. Results are kept in memory and written to Excel once at the end (this is the fastest approach).

You can stop the simulation at any time with the stop button (■). With a small number of runs this should finish quickly.

In [ ]:
```
# -- Choose which phases to calculate each run ------------------------------
# If gammaUnsat is fixed for every material, the initial stress phases only need
# to be calculated once. If gammaUnsat is randomised, all phases are redone each
# run so the stress state stays consistent.
gamma_is_randomised = any(
    params.get("gammaUnsat") is not None for params in MATERIALS.values()
)

if gamma_is_randomised:
    PHASE_CHAIN = FULL_PHASE_CHAIN
    logging.info("gammaUnsat is randomised -- all phases recalculated each run.")
else:
    PHASE_CHAIN = [SAFETY_PHASE]
    logging.info("gammaUnsat is fixed -- pre-calculating initial phases once ...")
    g_i.gotostages()
    for ph_name in ["InitialPhase", "Phase_1"]:
        phase_lookup[ph_name].ShouldCalculate = True
        g_i.calculate(phase_lookup[ph_name])
        logging.info(f"  {ph_name} calculated.")
    logging.info("Initial phases complete. Only the Safety phase is redone each run.")

# -- Generate all parameter combinations before the loop --------------------
rng     = np.random.default_rng(RANDOM_SEED)
samples = generate_samples(MATERIALS, N_SIMULATIONS, rng, SAMPLING_METHOD)
logging.info(f"Generated {N_SIMULATIONS} parameter combinations ({SAMPLING_METHOD}).")

records    = []
start_time = time.time()

logging.info("=" * 60)
logging.info("Simulation started")
logging.info(f"  Runs            : {N_SIMULATIONS}")
logging.info(f"  Sampling method : {SAMPLING_METHOD}")
```

```
logging.info(f"  Phase chain     : {' -> '.join(PHASE_CHAIN)}")
logging.info("=" * 60)

for run_id in range(1, N_SIMULATIONS + 1):

    sample    = samples[run_id - 1]
    run_start = time.time()

    record = {
        "run_id":    run_id,
        "converged": False,
        "FoS":       None,
        "timestamp": datetime.now().isoformat(timespec="seconds"),
    }
    for mat_name, params in sample.items():
        for param, val in params.items():
            record[f"{mat_name.lower()}_{param}"] = round(val, 2)
            if param == "gammaUnsat" and GAMMA_SAT_OFFSET is not None:
                record[f"{mat_name.lower()}_gammaSat"] = round(val + GAMMA_SAT_OFFSET, 2)

    try:
        # Step 1: send the sampled parameters to the PLAXIS model
        g_i.gotosoil()
        for mat_name, params in sample.items():
            mat   = mat_lookup[mat_name]
            props = []
            for param, val in params.items():
                props += [param, val]
                if param == "gammaUnsat" and GAMMA_SAT_OFFSET is not None:
                    props += ["gammaSat", val + GAMMA_SAT_OFFSET]
            mat.setproperties(*props)

        # Step 2: run the calculation phases
        g_i.gotostages()
        for ph_name in PHASE_CHAIN:
            phase_lookup[ph_name].ShouldCalculate = True
        for ph_name in PHASE_CHAIN:
            g_i.calculate(phase_lookup[ph_name])

        # Step 3: read the Factor of Safety from Output
        fos = read_fos_with_timeout(g_i, safety_phase_obj)
        if fos is None or fos <= 0:
            raise RuntimeError(f"Unexpected FoS value received: {fos}")

        record["FoS"]       = round(float(fos), 3)
        record["converged"] = True

        elapsed = (time.time() - start_time) / 60
        run_sec = time.time() - run_start
        logging.info(f"  Run {run_id:>3d}/{N_SIMULATIONS}  FoS={fos:.3f}  [{run_sec:.0f}s |
{elapsed:.1f}min]")

    except Exception as exc:
        # If one run fails it is logged and skipped; the loop keeps going.
        logging.warning(f"  Run {run_id:>3d}/{N_SIMULATIONS}  FAILED -- {exc}")

    records.append(record)

    # short pause to let PLAXIS settle before the next run
    try:
        g_i.gotostages()
    except Exception:
        pass
    time.sleep(SETTLE_SLEEP)

# -- Save everything once, at the end ----------------------------------------
df_results = pd.DataFrame(records)
save_to_excel(records, RESULTS_XLSX)
total_min  = (time.time() - start_time) / 60

logging.info("=" * 60)
logging.info("Simulation complete.")
logging.info(f"  Total time : {total_min:.1f} minutes")
logging.info(f"  Converged  : {df_results['converged'].sum()} / {N_SIMULATIONS}")
logging.info(f"  Saved to   : {RESULTS_XLSX}")
logging.info("=" * 60)
```

Cell 6 — Results Summary

Prints a statistical summary of the simulation results including the reliability index β and probability of failure P(f) under

both normal and lognormal FoS assumptions.

In [ ]:
```python
from scipy.stats import norm as _norm

df_conv   = df_results[df_results["converged"]].copy()
df_failed = df_results[~df_results["converged"]].copy()

print(f"{'='*60}")
print(f"  Probabilistic Slope Stability — Results Summary")
print(f"{'='*60}")
print(f"  Sampling method   : {SAMPLING_METHOD}")
print(f"  Total runs        : {N_SIMULATIONS}")
print(f"  Converged runs    : {len(df_conv)}")
print(f"  Failed runs       : {len(df_failed)}")

if len(df_conv) > 0:
    fos     = df_conv["FoS"]
    mu      = fos.mean()
    sigma   = fos.std()
    V       = sigma / mu
    beta_n  = (mu - 1.0) / sigma
    beta_ln = np.log(mu / np.sqrt(1 + V**2)) / np.sqrt(np.log(1 + V**2))
    pf_n    = _norm.cdf(-beta_n)  * 100
    pf_ln   = _norm.cdf(-beta_ln) * 100

    print()
    print(f"  Factor of Safety")
    print(f"  {'-'*40}")
    print(f"  {'Mean':<22}: {mu:.3f}")
    print(f"  {'Standard deviation':<22}: {sigma:.3f}")
    print(f"  {'Minimum':<22}: {fos.min():.3f}")
    print(f"  {'Maximum':<22}: {fos.max():.3f}")
    print(f"  {'P(FoS < 1.0)':<22}: {(fos < 1.0).mean()*100:.2f} %")
    print()
    print(f"  Reliability Index")
    print(f"  {'-'*40}")
    print(f"  {'beta (Normal)':<22}: {beta_n:.3f}")
    print(f"  {'P(f) Normal [%]':<22}: {pf_n:.4e}")
    print(f"  {'beta (Lognormal)':<22}: {beta_ln:.3f}")
    print(f"  {'P(f) Lognormal [%]':<22}: {pf_ln:.4e}")

print(f"{'='*60}")
print(f"  Results saved to: {RESULTS_XLSX.parent.name}")
print(f"{'='*60}")

param_cols   = [c for c in df_conv.columns
                if c.startswith(("sand_", "clay_")) and "gammaSat" not in c]
df_conv[["run_id", "FoS"] + param_cols].head(10)
```

Cell 6b — Sampled Parameter Distributions

Histograms of all sampled parameters with the theoretical distribution overlaid. Confirms that the sampling covered the full range of the defined distributions.

In [ ]:
```python
# -- Parameter Distribution Check ---------------------------------------------
# Plots histograms of all sampled parameters with the theoretical
# distribution overlaid. Confirms that sampling covered the full range.

if len(df_conv) == 0:
    print("No converged runs to plot.")
else:
    from scipy.stats import norm as _norm_dist

    param_cols = [c for c in df_conv.columns
                  if any(c.startswith(m.lower() + "_") for m in MATERIALS)
                  and "gammaSat" not in c]

    _SYM = {"phi": r"$\phi$", "cRef": r"$c_{ref}$",
            "gammaUnsat": r"$\gamma_{unsat}$"}

    def _plabel(col):
        parts = col.split("_")
        mat   = "_".join(parts[:-1]).replace("sand", "Sand").replace("clay", "Clay")
        return f"{mat} {_SYM.get(parts[-1], parts[-1])}"

    ncols = 3
    nrows = (len(param_cols) + ncols - 1) // ncols
    fig, axes = plt.subplots(nrows, ncols, figsize=(5 * ncols, 4 * nrows))
    axes = np.array(axes).flatten()
```

```
    for i, col in enumerate(param_cols):
        ax   = axes[i]
        lbl  = _plabel(col)
        data = df_conv[col].values

        ax.hist(data, bins=20, density=True, color="steelblue",
                alpha=0.5, edgecolor="white", label="Sampled")

        # Overlay theoretical distribution if defined in MATERIALS
        parts    = col.split("_")
        param    = parts[-1]
        mat_key  = next((k for k in MATERIALS
                         if k.lower() == "_".join(parts[:-1]).lower()), None)
        if mat_key and isinstance(MATERIALS.get(mat_key, {}).get(param), dict):
            cfg = MATERIALS[mat_key][param]
            if cfg["type"] == "normal":
                x_th = np.linspace(data.min(), data.max(), 200)
                ax.plot(x_th, _norm_dist.pdf(x_th, cfg["mean"], cfg["std"]),
                        color="navy", lw=2, label="Theoretical")

        ax.set_title(lbl, fontsize=9)
        ax.set_xlabel(lbl, fontsize=8)
        ax.set_ylabel("Density", fontsize=8)
        ax.legend(fontsize=7)
        ax.grid(True, alpha=0.2)

    for j in range(i + 1, len(axes)):
        axes[j].set_visible(False)

    fig.suptitle(
        f"Sampled Parameter Distributions — {SAMPLING_METHOD.replace('_', ' ').title()} ({len(df_conv)}
runs)",
        fontsize=11, fontweight="bold"
    )
    plt.tight_layout()
    dist_path = RESULTS_DIR / f"parameter_distributions_{_tag}_{RUN_TIMESTAMP}.png"
    fig.savefig(dist_path, dpi=150, bbox_inches="tight")
    plt.show()
    print(f"Saved to: {dist_path}")
```

Cell 7 — FoS Distribution Plots

Generates two plots saved as separate image files:

Probability Density Function (PDF): Histogram of FoS values with a smooth Kernel Density Estimate (KDE) overlay. The KDE places a Gaussian kernel over each simulated FoS value and sums the resulting curves to produce a continuous estimate of the distribution.

Cumulative Distribution Function (CDF): For any FoS value on the x-axis, the y-axis shows the probability that the actual FoS is below that value.

In [ ]:
```
if len(df_conv) == 0:
    print("No converged runs to plot.")
else:
    fos   = df_conv["FoS"].values
    _meth = SAMPLING_METHOD.replace("_", " ").title()

    # -- Probability Density Function ----------------------------------------
    fig1, ax1 = plt.subplots(figsize=(8, 5))
    ax1.hist(fos, bins=30, density=True, color="steelblue", alpha=0.5,
             edgecolor="white", label="Histogram")
    kde_x = np.linspace(fos.min() - 0.2, fos.max() + 0.2, 300)
    ax1.plot(kde_x, stats.gaussian_kde(fos)(kde_x), color="navy", lw=2, label="KDE")
    ax1.axvline(1.0, color="red",   lw=2, ls="--", label="FoS = 1.0 (failure)")
    ax1.axvline(fos.mean(), color="green", lw=2, label=f"Mean = {fos.mean():.3f}")
    ax1.set_xlabel("Factor of Safety", fontsize=11)
    ax1.set_ylabel("Probability Density", fontsize=11)
    ax1.set_title(
        f"Probability Density Function — {_meth} ({len(df_conv)} runs)\n"
        f"Mean = {fos.mean():.3f}  |  Std = {fos.std():.3f}",
        fontsize=11, fontweight="bold")
    ax1.legend(fontsize=9)
    ax1.grid(True, alpha=0.3)
    fig1.tight_layout()
    pdf_path = RESULTS_DIR / f"pdf_{_tag}_{RUN_TIMESTAMP}.png"
    fig1.savefig(pdf_path, dpi=150, bbox_inches="tight")
    plt.show()
```

```
    print(f"PDF saved to: {pdf_path}")

    # -- Cumulative Distribution Function -----------------------------------
    fig2, ax2 = plt.subplots(figsize=(8, 5))
    fos_s = np.sort(fos)
    cdf   = np.arange(1, len(fos_s) + 1) / len(fos_s)
    ax2.plot(fos_s, cdf * 100, color="steelblue", lw=2)
    ax2.axvline(1.0, color="red", lw=2, ls="--", label="FoS = 1.0 (failure)")
    pf = (fos < 1.0).mean()
    if pf > 0:
        ax2.fill_between(fos_s[fos_s <= 1.0], cdf[fos_s <= 1.0] * 100,
                         alpha=0.3, color="red",
                         label=f"P(FoS < 1.0) = {pf*100:.2f}%")
    ax2.set_xlabel("Factor of Safety", fontsize=11)
    ax2.set_ylabel("Probability [%]", fontsize=11)
    ax2.set_title(
        f"Cumulative Distribution Function — {_meth} ({len(df_conv)} runs)\n"
        f"Mean = {fos.mean():.3f}  |  Std = {fos.std():.3f}",
        fontsize=11, fontweight="bold")
    ax2.legend(fontsize=9)
    ax2.grid(True, alpha=0.3)
    fig2.tight_layout()
    cdf_path = RESULTS_DIR / f"cdf_{_tag}_{RUN_TIMESTAMP}.png"
    fig2.savefig(cdf_path, dpi=150, bbox_inches="tight")
    plt.show()
    print(f"CDF saved to: {cdf_path}")
```

Cell 8 — Parameter Sensitivity

Shows the relationship between each randomised soil parameter and the resulting Factor of Safety.

The Pearson correlation coefficient r quantifies the strength of the linear relationship. Values close to ±1 indicate a strong influence on the FoS, while values close to 0 indicate little influence. This analysis identifies which parameters most warrant careful characterisation in the site investigation.

In [ ]:

```
if len(df_conv) == 0:
    print("No converged runs to plot.")
else:
    param_cols = [c for c in df_conv.columns
                  if c.startswith(("sand_", "clay_")) and "gammaSat" not in c]

    # Symbol mapping for axis labels
    _SYM = {"phi": r"$\phi$", "cRef": r"$c_{ref}$",
            "gammaUnsat": r"$\gamma_{unsat}$", "gammaSat": r"$\gamma_{sat}$"}

    def _label(col):
        parts = col.split("_")
        mat   = "_".join(parts[:-1]).replace("sand", "Sand").replace("clay", "Clay")
        return f"{mat} {_SYM.get(parts[-1], parts[-1])}"

    ncols = 3
    nrows = (len(param_cols) + ncols - 1) // ncols
    fig, axes = plt.subplots(nrows, ncols, figsize=(5 * ncols, 4 * nrows))
    axes = np.array(axes).flatten()

    for i, col in enumerate(param_cols):
        ax   = axes[i]
        lbl  = _label(col)
        corr = df_conv[col].corr(df_conv["FoS"])
        ax.scatter(df_conv[col], df_conv["FoS"],
                   alpha=0.35, s=20, color="steelblue", edgecolors="none")
        m, b   = np.polyfit(df_conv[col], df_conv["FoS"], 1)
        x_line = np.linspace(df_conv[col].min(), df_conv[col].max(), 100)
        ax.plot(x_line, m * x_line + b, color="red", lw=1.5, ls="--")
        ax.set_title(f"{lbl}\nPearson r = {corr:.2f}", fontsize=9)
        ax.set_xlabel(lbl, fontsize=8)
        ax.set_ylabel("Factor of Safety", fontsize=8)
        ax.axhline(1.0, color="red", lw=0.8, ls=":")
        ax.grid(True, alpha=0.2)

    for j in range(i + 1, len(axes)):
        axes[j].set_visible(False)

    fig.suptitle(
        f"Parameter Sensitivity — {SAMPLING_METHOD.replace('_', ' ').title()} ({len(df_conv)} runs)\n"
        "Pearson r closer to ±1 indicates stronger influence on FoS",
        fontsize=11, fontweight="bold"
    )
    plt.tight_layout()
```

```
    sens_path = RESULTS_DIR / f"sensitivity_{_tag}_{RUN_TIMESTAMP}.png"
    plt.savefig(sens_path, dpi=150, bbox_inches="tight")
    plt.show()
    print(f"Sensitivity plot saved to: {sens_path}")
```

Cell 9 — CDF Excel Export

Exports the CDF data to an Excel file with an embedded scatter chart. The chart format matches the report style with Factor of Safety on the x-axis (0.9–2.0) and Probability [%] on the y-axis (0–100%).

In [ ]:
```
if len(df_conv) == 0:
    print("No converged runs to export.")
else:
    from openpyxl import load_workbook
    from openpyxl.chart import ScatterChart, Reference, Series
    from openpyxl.chart.axis import ChartLines

    fos_sorted = np.sort(df_conv["FoS"].values)
    cdf_values = np.arange(1, len(fos_sorted) + 1) / len(fos_sorted)

    df_cdf = pd.DataFrame({
        "Factor of Safety": fos_sorted,
        "Probability [%]":  cdf_values * 100,
    })

    cdf_xlsx = RESULTS_DIR / f"cdf_{_tag}_{RUN_TIMESTAMP}.xlsx"
    df_cdf.to_excel(cdf_xlsx, index=False, engine="openpyxl")

    wb = load_workbook(cdf_xlsx)
    ws = wb.active

    chart = ScatterChart()
    chart.title  = f"Cumulative Distribution Function — {SAMPLING_METHOD.replace('_', ' ').title()}"
    chart.style  = 2
    chart.legend = None
    chart.width  = 20
    chart.height = 15

    chart.x_axis.title          = "Factor of Safety"
    chart.y_axis.title          = "Probability [%]"
    chart.x_axis.scaling.min    = 0.9
    chart.x_axis.scaling.max    = 2.0
    chart.y_axis.scaling.min    = 0.0
    chart.y_axis.scaling.max    = 100.0
    chart.x_axis.majorGridlines = ChartLines()
    chart.y_axis.majorGridlines = ChartLines()

    x_data = Reference(ws, min_col=1, min_row=2, max_row=len(df_cdf) + 1)
    y_data = Reference(ws, min_col=2, min_row=2, max_row=len(df_cdf) + 1)
    series = Series(y_data, x_data)
    series.marker.symbol = "none"
    series.graphicalProperties.line.solidFill = "1F77B4"
    series.graphicalProperties.line.width     = 20000

    chart.series.append(series)
    ws.add_chart(chart, "D2")
    wb.save(cdf_xlsx)

    print(f"CDF Excel saved to: {cdf_xlsx}")
    print(f"  Rows      : {len(df_cdf)}")
    print(f"  FoS range : {fos_sorted[0]:.3f} – {fos_sorted[-1]:.3f}")
    print(f"  P(FoS<1.0): {(fos_sorted < 1.0).mean()*100:.2f} %")
```

F

Python script — full version

Probabilistic Slope Stability Analysis — PLAXIS 2D

Full version (production)

This notebook runs a full probabilistic slope stability analysis with PLAXIS 2D, intended for **large runs (e.g. 1000 simulations)**. It includes the robustness features needed for long, unattended runs:

- a **keep-alive** that nudges the mouse so Windows does not lock the screen,
- an **Output sync + automatic retry** so a single slow Factor-of-Safety read does not stop the whole simulation,
- an explicit **MaxStepsStored** setting so the Safety phase always stores enough steps to read SumMsf reliably,
- an optional **tab-closer** that closes each PLAXIS Output tab after it is read, so windows do not pile up over hundreds of runs.

For a small teaching run, use the **simple version** instead.

Before you begin

1. **Open PLAXIS 2D** and load the slope model you want to analyse.
2. **Turn on the scripting server** in PLAXIS 2D under `Expert → Configure remote scripting server`. Note the input port, output port, and password.
3. **Run this notebook on the same computer** as PLAXIS 2D (Windows).
4. **Edit Cell 1 and Cell 2** with your settings.
5. **Replace every placeholder** in capitals or angle brackets (`YOUR_PASSWORD` , `C:\Path\To\...` , `<version>`).
6. If you want the tab-closer on, read **Cell 4b** carefully — it depends on the screen position of the Output tab's close button and must be calibrated on the computer you run on.

How to run

Run the cells in order from top to bottom (`Shift + Enter`). Results are kept in memory during the run and written to Excel once at the end. The simulation can be stopped with the stop button (■) at any time.

Cell 1 — Settings

Fill in the values below and run the cell. Lines you usually change are marked `# <-- CHANGE THIS` .

In [ ]:
```
from pathlib import Path
from datetime import datetime as _dt

# ============================================================================
#  PLAXIS CONNECTION
#  From PLAXIS 2D: Expert -> Configure remote scripting server
# ============================================================================
PORT_IN   = 10000
PORT_OUT  = 10001
PASSWORD  = "YOUR_PASSWORD"        # <-- CHANGE THIS (copy from PLAXIS)

# ============================================================================
#  PLAXIS OUTPUT EXECUTABLE
#  Full path to Plaxis2DOutput.exe on this computer.
#  Used to restart Output automatically if it becomes unresponsive.
#  Run Cell 1b to locate the correct path.
# ============================================================================
PLAXIS_OUTPUT_EXE = r"C:\Program Files\Seequent\PLAXIS 2D <version>\Plaxis2DOutput.exe"  # <-- CHANGE

# ============================================================================
#  PHASES
# ============================================================================
SAFETY_PHASE     = "Phase_3"                              # <-- CHANGE if needed
FULL_PHASE_CHAIN = ["InitialPhase", "Phase_1", "Phase_3"] # <-- CHANGE if needed

# ============================================================================
#  MESH
#  0.1 = very coarse | 0.05 = medium | 0.01 = fine
# ============================================================================
MESH_COARSENESS = 0.05

# ============================================================================
#  SIMULATION
```

```
#  SAMPLING_METHOD : "monte_carlo" or "latin_hypercube"
#  N_SIMULATIONS   : number of runs (200+ recommended for production)
#  N_SAND_LAYERS   : number of sand layers (used for naming the output folder)
#  RANDOM_SEED     : None = different each time | integer = reproducible
# ============================================================================
N_SIMULATIONS   = 1000                # <-- CHANGE THIS
SAMPLING_METHOD = "latin_hypercube"   # <-- "monte_carlo" or "latin_hypercube"
N_SAND_LAYERS   = 3                   # <-- CHANGE if needed
RANDOM_SEED     = None                # <-- set a number for reproducible results

# ============================================================================
#  MAX STEPS STORED
#  Number of calculation steps stored for the Safety phase. The PLAXIS default
#  of 1 can make the Output server fail when reading SumMsf; 10 is safe.
# ============================================================================
MAX_STEPS_STORED = 10

# ============================================================================
#  TAB-CLOSER (optional robustness feature)
#  If True, the active PLAXIS Output tab is closed after each FoS reading so
#  open windows do not accumulate over many runs. It clicks the X button at the
#  screen coordinates below using the mouse, so those coordinates must match
#  YOUR screen. Calibrate them in Cell 4b before turning this on.
#  Leave False if you are unsure -- the simulation also works without it.
# ============================================================================
ENABLE_TAB_CLOSE   = False    # <-- set True only after calibrating in Cell 4b
OUTPUT_TAB_CLOSE_X = 1908     # <-- set in Cell 4b
OUTPUT_TAB_CLOSE_Y = 36       # <-- set in Cell 4b

# ============================================================================
#  TIMING
#  SLEEP_AFTER_VIEW : seconds after view() before reading Output (increase if slow)
#  SETTLE_SLEEP     : pause between runs to let PLAXIS release memory
#  FOS_RETRY_WAIT   : seconds to wait before retrying a failed FoS read
# ============================================================================
SLEEP_AFTER_VIEW = 5
SETTLE_SLEEP     = 2
FOS_RETRY_WAIT   = 120

# ============================================================================
#  WHERE TO SAVE RESULTS
#  One Excel file and the figures are written here at the end of the run.
# ============================================================================
_base_dir     = Path(r"C:\Path\To\Output")   # <-- CHANGE THIS
RUN_TIMESTAMP = _dt.now().strftime("%Y-%m-%d_%H-%M")

print("Settings loaded.")
print(f"  Simulations     : {N_SIMULATIONS}")
print(f"  Sampling method : {SAMPLING_METHOD}")
print(f"  Tab-closer      : {'ON' if ENABLE_TAB_CLOSE else 'OFF'}")
print(f"  Output folder   : {_base_dir}")
```

Cell 1b — Locate PLAXIS Output Executable (optional)

Run this cell if the path to `Plaxis2DOutput.exe` is unknown. Copy the result into `PLAXIS_OUTPUT_EXE` in Cell 1.

In [ ]:
```
import os
from pathlib import Path

print(f"Current path exists: {os.path.exists(PLAXIS_OUTPUT_EXE)}")
print(f"  {PLAXIS_OUTPUT_EXE}")
print()
print("Searching for Plaxis2DOutput.exe ...")
for root in [r"C:\Program Files\Bentley", r"C:\Program Files\Seequent",
             r"C:\Program Files (x86)\Bentley", r"C:\ProgramData\Seequent"]:
    try:
        for p in Path(root).rglob("Plaxis2DOutput.exe"):
            print(f"  Found: {p}")
    except Exception:
        pass
print("Search complete.")
```

Cell 2 — Soil Parameter Distributions

Define the statistical distribution for each soil parameter. Set a parameter to `None` to keep it fixed at the model value.

The `GAMMA_SAT_OFFSET` defines the difference between saturated and unsaturated unit weight. If `gammaUnsat` is randomised, `gammaSat` is updated automatically.

If `gammaUnsat` is set to `None` for all materials, the initial stress phases are calculated only once before the loop. This is valid because the stress distribution does not change when unit weight is fixed, and significantly reduces the total runtime.

Available distribution types

```
# Normal
{"type": "normal", "mean": 33.8, "std": 2.07}

# Normal with truncation (values outside range are rejected and redrawn)
{"type": "normal", "mean": 33.8, "std": 2.07, "min": 25.0, "max": 42.0}

# Normal with 90% confidence interval (cutoff at mean ± 1.645 * std)
{"type": "normal", "mean": 33.8, "std": 2.07, "confidence": 0.90}

# Lognormal (enter real-space mean and std)
{"type": "lognormal", "mean": 2.3, "std": 0.5}

# Triangular
{"type": "triangular", "low": 28.0, "mode": 33.8, "high": 40.0}

# Uniform
{"type": "uniform", "low": 28.0, "high": 40.0}

# Custom (fitted KDE from measured data points)
{"type": "custom", "data": [32.1, 33.4, 34.2, 35.0, 33.8]}

# Fixed at model value
None
```

In [ ]:

```
# ============================================================================
#  UNIT WEIGHT OFFSET
#  gammaSat = gammaUnsat + GAMMA_SAT_OFFSET  (kN/m3)
#  Applied automatically when gammaUnsat is randomised.
# ============================================================================
GAMMA_SAT_OFFSET = 1.0

# ============================================================================
#  SAND_1
#  Material name must match exactly the name in the PLAXIS model (case-sensitive)
# ============================================================================
SAND_1 = {
    "phi": {
        "type": "normal",
        "mean": 33.3,
        "std":   1.58,
        "min":  23.5,
        "max":  44.1,
    },
    "cRef":       None,
    "gammaUnsat": None,
}

# ============================================================================
#  SAND_2
# ============================================================================
SAND_2 = {
    "phi": {
        "type": "normal",
        "mean": 33.4,
        "std":   1.50,
        "min":  23.5,
        "max":  44.1,
    },
    "cRef":       None,
    "gammaUnsat": None,
}

# ============================================================================
#  SAND_3
# ============================================================================
SAND_3 = {
    "phi": {
        "type": "normal",
        "mean": 34.2,
        "std":   1.92,
        "min":  23.5,
        "max":  44.1,
    },
    "cRef":       None,
    "gammaUnsat": None,
}

# ============================================================================
```

```python
#  CLAY
# ============================================================================
CLAY = {
    "phi": {
        "type": "normal",
        "mean": 32.9,
        "std":   1.28,
        "min":  26.5,
        "max":  39.3,
    },
    "cRef": {
        "type": "normal",
        "mean": 2.3,
        "std":  0.1,
        "min":  1.8,
        "max":  2.8,
    },
    "gammaUnsat": None,
}

# Bundle all materials -- do not edit
MATERIALS = {
    "Sand_1": SAND_1,
    "Sand_2": SAND_2,
    "Sand_3": SAND_3,
    "Clay":   CLAY,
}

# ============================================================================
#  RESOLVE CONFIDENCE INTERVAL CUTOFFS
#  Converts {"confidence": 0.90} to explicit min/max at mean ± 1.645 * std
# ============================================================================
import math
_CONFIDENCE_Z = {0.90: 1.645, 0.95: 1.960, 0.99: 2.576}
for _mp in MATERIALS.values():
    for _cfg in _mp.values():
        if isinstance(_cfg, dict) and "confidence" in _cfg:
            _z = _CONFIDENCE_Z.get(_cfg["confidence"], 1.645)
            _cfg["min"] = _cfg["mean"] - _z * _cfg["std"]
            _cfg["max"] = _cfg["mean"] + _z * _cfg["std"]
            del _cfg["confidence"]

# ============================================================================
#  BUILD OUTPUT FOLDER NAME
#  The folder name encodes the sampling method, number of runs, number of
#  sand layers, and the distribution type used for each randomised parameter.
#  Example: lhs_200runs_3sand_N_phi_N_cRef
# ============================================================================
_DIST_SHORT = {"normal": "N", "normal_ci": "NCI", "lognormal": "LN",
               "triangular": "T", "uniform": "U", "custom": "KDE"}
_METH_SHORT = {"latin_hypercube": "lhs", "monte_carlo": "mc"}

_meth_s      = _METH_SHORT.get(SAMPLING_METHOD, SAMPLING_METHOD)
_params_seen = {}
for _mp in MATERIALS.values():
    for _pname, _cfg in _mp.items():
        if _cfg is not None and _pname not in _params_seen:
            _params_seen[_pname] = _DIST_SHORT.get(_cfg["type"], _cfg["type"])
_param_str   = "_".join(f"{v}_{k}" for k, v in _params_seen.items())
_tag         = f"{_meth_s}_{N_SIMULATIONS}runs_{N_SAND_LAYERS}sand_{_param_str}"
RESULTS_DIR  = _base_dir / _tag
RESULTS_XLSX = RESULTS_DIR / f"mc_results_{RUN_TIMESTAMP}.xlsx"
LOG_FILE     = RESULTS_DIR / f"mc_log_{RUN_TIMESTAMP}.log"

# ============================================================================
#  SUMMARY PRINTOUT
# ============================================================================
print(f"Output folder    : {RESULTS_DIR}")
print(f"gammaSat offset  : gammaUnsat + {GAMMA_SAT_OFFSET} kN/m3")
print()
print("Soil parameter distributions:")
for mat_name, params in MATERIALS.items():
    print(f"  {mat_name}:")
    for param, cfg in params.items():
        if cfg is None:
            print(f"    {param:<14}: Fixed at model value")
        elif cfg["type"] == "normal":
            lo = f"{cfg['min']:.2f}" if "min" in cfg else "-inf"
            hi = f"{cfg['max']:.2f}" if "max" in cfg else "+inf"
            print(f"    {param:<14}: Normal       mean={cfg['mean']:>6}  std={cfg['std']:>5}  [{lo}, {hi}]")
        elif cfg["type"] == "lognormal":
            print(f"    {param:<14}: Lognormal    mean={cfg['mean']:>6}  std={cfg['std']:>5}")
        elif cfg["type"] == "triangular":
```

```
            print(f"    {param:<14}: Triangular   low={cfg['low']}  mode={cfg['mode']}  high={cfg['high']}")
        elif cfg["type"] == "uniform":
            print(f"    {param:<14}: Uniform      low={cfg['low']}  high={cfg['high']}")
        elif cfg["type"] == "custom":
            print(f"    {param:<14}: Custom KDE   {len(cfg['data'])} data points")
    print()
```

Cell 2b — Output file names (no editing needed)

Builds the output folder name automatically from your settings and creates the folder. Just run it.

In [ ]:
```
# Build an automatic, descriptive folder name from the settings above.
_tag = (f"{'lhs' if SAMPLING_METHOD == 'latin_hypercube' else 'mc'}"
        f"_{N_SIMULATIONS}runs_{N_SAND_LAYERS}sand")

RESULTS_DIR  = _base_dir / _tag
RESULTS_XLSX = RESULTS_DIR / f"results_{_tag}_{RUN_TIMESTAMP}.xlsx"
LOG_FILE     = RESULTS_DIR / f"log_{_tag}_{RUN_TIMESTAMP}.txt"

RESULTS_DIR.mkdir(parents=True, exist_ok=True)
print(f"Results will be saved in:\n  {RESULTS_DIR}")
```

Cell 3 — Load libraries and helper functions (no editing needed)

Loads the libraries and defines all helper functions: sampling, the keep-alive, the Output sync + retry reader, the (optional) tab-closer, and the results/Excel functions. Just run it.

In [ ]:
```
import sys, time, logging, subprocess, threading
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
from scipy import stats
from scipy.stats import norm, lognorm, qmc

# Install any missing packages automatically
for pkg in ["pandas", "matplotlib", "scipy", "openpyxl", "psutil", "pyautogui"]:
    try:
        __import__(pkg)
    except ImportError:
        print(f"Installing {pkg} ...")
        subprocess.check_call([sys.executable, "-m", "pip", "install", pkg])
import psutil
import pyautogui

# Add the PLAXIS scripting library to the Python path
PLAXIS_PATH = r"C:\ProgramData\Seequent\PLAXIS Python Distribution V<n>\python\Lib\site-packages"  # <-- CHANGE version number
if PLAXIS_PATH not in sys.path:
    sys.path.insert(0, PLAXIS_PATH)
from plxscripting.easy import new_server

# Write log messages to both the screen and a log file
for h in logging.root.handlers[:]:
    logging.root.removeHandler(h)
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s  %(levelname)-8s  %(message)s",
    handlers=[logging.FileHandler(LOG_FILE, encoding="utf-8"),
              logging.StreamHandler(sys.stdout)],
)

# ============================================================================
#  KEEP-ALIVE
#  Moves the mouse one pixel every 30 seconds so Windows does not lock the
#  screen during a long run. Started when the loop begins, stopped when it ends.
# ============================================================================
_keep_alive_running = False

def _keep_alive_worker():
    while _keep_alive_running:
        try:
            pyautogui.moveRel(1, 0); time.sleep(30)
            pyautogui.moveRel(-1, 0); time.sleep(30)
        except Exception:
            pass
```

```python
def start_keep_alive():
    global _keep_alive_running, _keep_alive_thread
    _keep_alive_running = True
    _keep_alive_thread  = threading.Thread(target=_keep_alive_worker, daemon=True)
    _keep_alive_thread.start()
    logging.info("Keep-alive started.")

def stop_keep_alive():
    global _keep_alive_running
    _keep_alive_running = False
    logging.info("Keep-alive stopped.")

# ============================================================================
#  SAMPLING FUNCTIONS
# ============================================================================

def _lognormal_params(mean, std):
    """Convert real-space mean and std to lognormal mu and sigma."""
    v = std**2
    return np.log(mean**2 / np.sqrt(v + mean**2)), np.sqrt(np.log(1 + v / mean**2))

def _get_active_params(materials):
    return [(mat, p, cfg)
            for mat, params in materials.items()
            for p, cfg in params.items()
            if cfg is not None]

def _ppf(cfg, u):
    """Transform a uniform sample u in [0, 1] to a parameter value via the
    inverse CDF of the chosen distribution."""
    t    = cfg["type"]
    vmin = cfg.get("min", None)
    vmax = cfg.get("max", None)

    if t == "normal":
        m, s = cfg["mean"], cfg["std"]
        u_lo = norm.cdf(vmin, m, s) if vmin is not None else 0.0
        u_hi = norm.cdf(vmax, m, s) if vmax is not None else 1.0
        return float(norm.ppf(u_lo + u * (u_hi - u_lo), m, s))
    elif t == "lognormal":
        mu, sigma = _lognormal_params(cfg["mean"], cfg["std"])
        u_lo = lognorm.cdf(vmin, sigma, scale=np.exp(mu)) if vmin is not None else 0.0
        u_hi = lognorm.cdf(vmax, sigma, scale=np.exp(mu)) if vmax is not None else 1.0
        return float(lognorm.ppf(u_lo + u * (u_hi - u_lo), sigma, scale=np.exp(mu)))
    elif t == "triangular":
        from scipy.stats import triang
        lo, mode, hi = cfg["low"], cfg["mode"], cfg["high"]
        c = (mode - lo) / (hi - lo)
        return float(triang.ppf(u, c, loc=lo, scale=hi - lo))
    elif t == "uniform":
        return float(cfg["low"] + u * (cfg["high"] - cfg["low"]))
    elif t == "custom":
        kde = stats.gaussian_kde(np.array(cfg["data"], dtype=float))
        for _ in range(1000):
            v = float(kde.resample(1)[0][0])
            if v > 0 and (vmin is None or v >= vmin) and (vmax is None or v <= vmax):
                return v
        return float(np.mean(cfg["data"]))
    else:
        raise ValueError(f"Unknown distribution type: '{t}'.")

def generate_samples(materials, n, rng, method):
    active   = _get_active_params(materials)
    n_params = len(active)
    if method == "latin_hypercube":
        lhs     = qmc.LatinHypercube(d=n_params, seed=RANDOM_SEED)
        u_array = lhs.random(n=n)
    else:
        u_array = rng.uniform(0, 1, size=(n, n_params))
    samples = []
    for i in range(n):
        sample = {mat: {} for mat in materials}
        for j, (mat_name, param, cfg) in enumerate(active):
            sample[mat_name][param] = _ppf(cfg, u_array[i, j])
        samples.append(sample)
    return samples

# ============================================================================
#  RESULTS FUNCTIONS
# ============================================================================

def _build_summary_df(records):
```

```
    from scipy.stats import norm as _norm
    df_all  = pd.DataFrame(records)
    df_conv = df_all[df_all["converged"]].copy()
    rows    = []
    if len(df_conv) > 0:
        fos     = df_conv["FoS"].values
        mu      = fos.mean(); sigma = fos.std(); V = sigma / mu
        beta_n  = (mu - 1.0) / sigma
        beta_ln = np.log(mu / np.sqrt(1 + V**2)) / np.sqrt(np.log(1 + V**2))
        pf_n    = _norm.cdf(-beta_n)  * 100
        pf_ln   = _norm.cdf(-beta_ln) * 100
        rows += [
            {"Category": "Results",     "Parameter": "Sampling method",   "Value": SAMPLING_METHOD},
            {"Category": "Results",     "Parameter": "Simulations",       "Value": N_SIMULATIONS},
            {"Category": "Results",     "Parameter": "Converged runs",    "Value": len(df_conv)},
            {"Category": "Results",     "Parameter": "Mean FoS",          "Value": round(mu, 3)},
            {"Category": "Results",     "Parameter": "Std FoS",           "Value": round(sigma, 3)},
            {"Category": "Results",     "Parameter": "Min FoS",           "Value": round(fos.min(), 3)},
            {"Category": "Results",     "Parameter": "Max FoS",           "Value": round(fos.max(), 3)},
            {"Category": "Results",     "Parameter": "",                  "Value": ""},
            {"Category": "Reliability", "Parameter": "beta (Normal)",     "Value": round(beta_n, 3)},
            {"Category": "Reliability", "Parameter": "P(f) Normal [%]",   "Value": f"{pf_n:.4e}"},
            {"Category": "Reliability", "Parameter": "beta (Lognormal)",  "Value": round(beta_ln, 3)},
            {"Category": "Reliability", "Parameter": "P(f) Lognormal [%]", "Value": f"{pf_ln:.4e}"},
            {"Category": "Reliability", "Parameter": "",                  "Value": ""},
        ]
    for mat_name, params in MATERIALS.items():
        for param, cfg in params.items():
            if cfg is None:
                rows.append({"Category": mat_name, "Parameter": param, "Value": "Fixed (model value)"})
            elif cfg["type"] == "normal":
                rows.append({"Category": mat_name, "Parameter": param,
                             "Value": f"Normal  mean={cfg['mean']}  std={cfg['std']}  [{cfg.get('min','-
inf')}, {cfg.get('max','+inf')}]"})
            elif cfg["type"] == "lognormal":
                rows.append({"Category": mat_name, "Parameter": param,
                             "Value": f"Lognormal  mean={cfg['mean']}  std={cfg['std']}"})
            elif cfg["type"] == "triangular":
                rows.append({"Category": mat_name, "Parameter": param,
                             "Value": f"Triangular  low={cfg['low']}  mode={cfg['mode']}  high=
{cfg['high']}"})
            elif cfg["type"] == "uniform":
                rows.append({"Category": mat_name, "Parameter": param,
                             "Value": f"Uniform  low={cfg['low']}  high={cfg['high']}"})
            elif cfg["type"] == "custom":
                rows.append({"Category": mat_name, "Parameter": param,
                             "Value": f"Custom KDE  ({len(cfg['data'])} data points)"})
    return pd.DataFrame(rows)

def save_to_excel(records, filepath):
    df_all     = pd.DataFrame(records)
    df_conv    = df_all[df_all["converged"]].copy()
    df_failed  = df_conv[df_conv["FoS"] < 1.0].copy() if len(df_conv) > 0 else pd.DataFrame()
    df_summary = _build_summary_df(records)
    with pd.ExcelWriter(filepath, engine="openpyxl") as writer:
        df_all.to_excel(writer, sheet_name="All results", index=False)
        if len(df_failed) > 0:
            df_failed.to_excel(writer, sheet_name="FoS below 1.0", index=False)
        else:
            pd.DataFrame({"Note": ["No runs with FoS < 1.0"]}).to_excel(
                writer, sheet_name="FoS below 1.0", index=False)
        df_summary.to_excel(writer, sheet_name="Summary", index=False)

# ============================================================================
#  PLAXIS OUTPUT MANAGEMENT
#  Tab-closer: closes the active Output tab by clicking its X button. Only used
#  when ENABLE_TAB_CLOSE is True (set in Cell 1, calibrated in Cell 4b).
#  Reader: waits until Output shows the latest step, then reads SumMsf (= FoS).
#  One automatic retry handles a temporarily unresponsive Output.
# ============================================================================

def close_output_tab():
    """Close the active Output tab by clicking the X button (only if enabled)."""
    if not ENABLE_TAB_CLOSE:
        return
    time.sleep(0.5)
    pyautogui.click(OUTPUT_TAB_CLOSE_X, OUTPUT_TAB_CLOSE_Y)

def _try_read_fos(g_i, safety_phase_obj):
    """One attempt to read FoS. Returns (fos, success)."""
    try:
        g_i.view(safety_phase_obj)
        time.sleep(SLEEP_AFTER_VIEW)
```

```
        target_step = f"Step_{safety_phase_obj.LastStep.value}"
        synced      = False
        for attempt in range(60):
            _, g_o    = new_server("localhost", PORT_OUT, password=PASSWORD)
            out_phase = getattr(g_o, SAFETY_PHASE)
            out_last  = out_phase.Steps[-1].Name.value
            if out_last == target_step:
                logging.info(f"    Output synced after {attempt + 1}s ({target_step})")
                synced = True
                break
            time.sleep(1)
        if not synced:
            logging.warning(f"    Output not synced (got {out_last}, expected {target_step}) -- reading
anyway")
        fos = out_phase.Reached.SumMsf.value
        close_output_tab()
        return fos, True
    except Exception:
        return None, False

def read_fos_with_timeout(g_i, safety_phase_obj):
    """Read FoS, with one automatic retry after FOS_RETRY_WAIT seconds."""
    fos, ok = _try_read_fos(g_i, safety_phase_obj)
    if ok and fos is not None and fos > 0:
        return fos
    logging.warning(f"    FoS read failed -- waiting {FOS_RETRY_WAIT}s before retry ...")
    time.sleep(FOS_RETRY_WAIT)
    fos, ok = _try_read_fos(g_i, safety_phase_obj)
    if ok and fos is not None and fos > 0:
        logging.info("    Retry successful.")
        return fos
    raise RuntimeError("FoS could not be read after retry.")

print("Libraries loaded and helper functions defined.")
```

Cell 4 — Connect to PLAXIS and build the mesh

Connects to your running PLAXIS model, checks that all the materials and phases named in Cell 1 and Cell 2 really exist, and builds the finite element mesh once.

If this cell fails: make sure PLAXIS 2D is open with the model loaded and the scripting server turned on, and that the names in Cell 1/Cell 2 match the model.

In [ ]:
```
logging.info(f"Connecting to PLAXIS 2D on localhost:{PORT_IN} ...")
s_i, g_i = new_server("localhost", PORT_IN, password=PASSWORD)
logging.info("Connection established.")

# Verify that all defined materials exist in the model
mat_lookup = {m.Name.value: m for m in g_i.Materials}
for mat_name in MATERIALS:
    if mat_name not in mat_lookup:
        raise RuntimeError(
            f"Material '{mat_name}' not found in the PLAXIS model.\n"
            f"Available materials: {list(mat_lookup.keys())}\n"
            f"Check that the name in Cell 2 matches exactly (case-sensitive)."
        )
    logging.info(f"Found material: '{mat_name}'")

# Verify that all phases exist
g_i.gotostages()
phase_lookup     = {ph.Name.value: ph for ph in g_i.Phases}
safety_phase_obj = phase_lookup.get(SAFETY_PHASE)
if safety_phase_obj is None:
    raise RuntimeError(
        f"Safety phase '{SAFETY_PHASE}' not found.\n"
        f"Available phases: {list(phase_lookup.keys())}"
    )
for name in FULL_PHASE_CHAIN:
    if name not in phase_lookup:
        raise RuntimeError(f"Phase '{name}' not found. Check FULL_PHASE_CHAIN in Cell 1.")
logging.info(f"Found all phases: {FULL_PHASE_CHAIN}")

# Generate the finite element mesh
logging.info(f"Generating mesh (coarseness = {MESH_COARSENESS}) ...")
g_i.gotomesh()
g_i.mesh(MESH_COARSENESS)
logging.info("Mesh generated. Ready to start simulation.")
```

Cell 4b — Activate and calibrate the tab-closer (optional)

The tab-closer closes each PLAXIS **Output** tab after its result has been read, so that open windows do not pile up over hundreds of runs. It works by physically **clicking the X button** on the Output tab, so it needs the screen coordinates of that button on **your** monitor.

To calibrate:

1. Open PLAXIS Output so a results tab with a close (X) button is visible.
2. Run the cell below. It counts down, then reads the current mouse position.
3. While it counts down, hover the mouse exactly over the **X** on the Output tab.
4. Copy the printed X and Y values into `OUTPUT_TAB_CLOSE_X` / `OUTPUT_TAB_CLOSE_Y` in **Cell 1**, and set `ENABLE_TAB_CLOSE = True`.
5. Re-run Cell 1 and Cell 3 so the new values take effect.

If you would rather not use this feature, skip this cell and leave `ENABLE_TAB_CLOSE = False`. The simulation still runs correctly without it.

In [ ]:
```
# Calibration helper: hover over the X button of the Output tab during countdown.
import time, pyautogui

print("Hover the mouse over the X (close) button of the PLAXIS Output tab.")
for s in range(5, 0, -1):
    print(f"  Reading mouse position in {s} s ...")
    time.sleep(1)

pos = pyautogui.position()
print()
print(f"Mouse position: X = {pos.x}, Y = {pos.y}")
print("Copy these into OUTPUT_TAB_CLOSE_X / OUTPUT_TAB_CLOSE_Y in Cell 1,")
print("then set ENABLE_TAB_CLOSE = True and re-run Cell 1 and Cell 3.")
```

Cell 5 — Run the simulation

The main loop. It generates all parameter combinations first, then for each run sends the parameters to PLAXIS, calculates, and reads the Factor of Safety. The keep-alive runs in the background so the screen does not lock. Results are kept in memory and written to Excel once at the end (fastest; no repeated disk writes).

Stop any time with the stop button (■).

In [ ]:
```
# -- Choose which phases to calculate each run ------------------------------
gamma_is_randomised = any(
    params.get("gammaUnsat") is not None for params in MATERIALS.values()
)

if gamma_is_randomised:
    PHASE_CHAIN = FULL_PHASE_CHAIN
    logging.info("gammaUnsat is randomised -- all phases recalculated each run.")
else:
    PHASE_CHAIN = [SAFETY_PHASE]
    logging.info("gammaUnsat is fixed -- pre-calculating initial phases once ...")
    g_i.gotostages()
    for ph_name in ["InitialPhase", "Phase_1"]:
        phase_lookup[ph_name].ShouldCalculate = True
        g_i.calculate(phase_lookup[ph_name])
        logging.info(f"  {ph_name} calculated.")
    logging.info("Initial phases complete. Only the Safety phase is redone each run.")

# -- Generate all parameter combinations before the loop --------------------
rng     = np.random.default_rng(RANDOM_SEED)
samples = generate_samples(MATERIALS, N_SIMULATIONS, rng, SAMPLING_METHOD)
logging.info(f"Generated {N_SIMULATIONS} parameter combinations ({SAMPLING_METHOD}).")

# -- Start keep-alive -------------------------------------------------------
start_keep_alive()

records    = []
start_time = time.time()

logging.info("=" * 60)
logging.info("Simulation started")
logging.info(f"  Runs            : {N_SIMULATIONS}")
logging.info(f"  Sampling method : {SAMPLING_METHOD}")
logging.info(f"  Phase chain     : {' -> '.join(PHASE_CHAIN)}")
logging.info(f"  MaxStepsStored  : {MAX_STEPS_STORED}")
```

```
logging.info(f"  Tab-closer      : {'ON' if ENABLE_TAB_CLOSE else 'OFF'}")
logging.info("=" * 60)

for run_id in range(1, N_SIMULATIONS + 1):

    sample    = samples[run_id - 1]
    run_start = time.time()

    record = {
        "run_id":    run_id,
        "converged": False,
        "FoS":       None,
        "timestamp": datetime.now().isoformat(timespec="seconds"),
    }
    for mat_name, params in sample.items():
        for param, val in params.items():
            record[f"{mat_name.lower()}_{param}"] = round(val, 2)
            if param == "gammaUnsat" and GAMMA_SAT_OFFSET is not None:
                record[f"{mat_name.lower()}_gammaSat"] = round(val + GAMMA_SAT_OFFSET, 2)

    try:
        # Step 1: send the sampled parameters to the PLAXIS model
        g_i.gotosoil()
        for mat_name, params in sample.items():
            mat   = mat_lookup[mat_name]
            props = []
            for param, val in params.items():
                props += [param, val]
                if param == "gammaUnsat" and GAMMA_SAT_OFFSET is not None:
                    props += ["gammaSat", val + GAMMA_SAT_OFFSET]
            mat.setproperties(*props)

        # Step 2: run the calculation phases
        g_i.gotostages()
        for ph_name in PHASE_CHAIN:
            phase_lookup[ph_name].ShouldCalculate = True
        safety_phase_obj.MaxStepsStored = MAX_STEPS_STORED
        for ph_name in PHASE_CHAIN:
            g_i.calculate(phase_lookup[ph_name])

        # Step 3: read the Factor of Safety from Output
        fos = read_fos_with_timeout(g_i, safety_phase_obj)
        if fos is None or fos <= 0:
            raise RuntimeError(f"Unexpected FoS value received: {fos}")

        record["FoS"]       = round(float(fos), 3)
        record["converged"] = True

        elapsed = (time.time() - start_time) / 60
        run_sec = time.time() - run_start
        logging.info(
            f"  Run {run_id:>4d}/{N_SIMULATIONS}  FoS={fos:.3f}"
            + "".join(f"  {mn}_phi={sample[mn].get('phi','-'):.1f}"
                      for mn in ["Sand_1", "Sand_2", "Sand_3"] if mn in sample)
            + (f"  Clay_phi={sample['Clay'].get('phi','-'):.1f}"
               f"  Clay_c={sample['Clay'].get('cRef','-'):.2f}" if "Clay" in sample else "")
            + f"  [{run_sec:.0f}s | {elapsed:.1f}min]"
        )

    except Exception as exc:
        logging.warning(f"  Run {run_id:>4d}/{N_SIMULATIONS}  FAILED -- {exc}")

    records.append(record)

    try:
        g_i.gotostages()
    except Exception:
        pass
    time.sleep(SETTLE_SLEEP)

# -- Save everything once, at the end ----------------------------------------
df_results = pd.DataFrame(records)
save_to_excel(records, RESULTS_XLSX)
total_min  = (time.time() - start_time) / 60

stop_keep_alive()

logging.info("=" * 60)
logging.info("Simulation complete.")
logging.info(f"  Total time : {total_min:.1f} minutes")
logging.info(f"  Converged  : {df_results['converged'].sum()} / {N_SIMULATIONS}")
logging.info(f"  Saved to   : {RESULTS_XLSX}")
logging.info("=" * 60)
```

Cell 6 — Results Summary

Prints a statistical summary of the simulation results including the reliability index β and probability of failure P(f) under both normal and lognormal FoS assumptions.

In [ ]:
```
from scipy.stats import norm as _norm

df_conv   = df_results[df_results["converged"]].copy()
df_failed = df_results[~df_results["converged"]].copy()

print(f"{'='*60}")
print(f"  Probabilistic Slope Stability — Results Summary")
print(f"{'='*60}")
print(f"  Sampling method   : {SAMPLING_METHOD}")
print(f"  Total runs        : {N_SIMULATIONS}")
print(f"  Converged runs    : {len(df_conv)}")
print(f"  Failed runs       : {len(df_failed)}")

if len(df_conv) > 0:
    fos     = df_conv["FoS"]
    mu      = fos.mean()
    sigma   = fos.std()
    V       = sigma / mu
    beta_n  = (mu - 1.0) / sigma
    beta_ln = np.log(mu / np.sqrt(1 + V**2)) / np.sqrt(np.log(1 + V**2))
    pf_n    = _norm.cdf(-beta_n)  * 100
    pf_ln   = _norm.cdf(-beta_ln) * 100

    print()
    print(f"  Factor of Safety")
    print(f"  {'-'*40}")
    print(f"  {'Mean':<22}: {mu:.3f}")
    print(f"  {'Standard deviation':<22}: {sigma:.3f}")
    print(f"  {'Minimum':<22}: {fos.min():.3f}")
    print(f"  {'Maximum':<22}: {fos.max():.3f}")
    print(f"  {'P(FoS < 1.0)':<22}: {(fos < 1.0).mean()*100:.2f} %")
    print()
    print(f"  Reliability Index")
    print(f"  {'-'*40}")
    print(f"  {'beta (Normal)':<22}: {beta_n:.3f}")
    print(f"  {'P(f) Normal [%]':<22}: {pf_n:.4e}")
    print(f"  {'beta (Lognormal)':<22}: {beta_ln:.3f}")
    print(f"  {'P(f) Lognormal [%]':<22}: {pf_ln:.4e}")

print(f"{'='*60}")
print(f"  Results saved to: {RESULTS_XLSX.parent.name}")
print(f"{'='*60}")

param_cols   = [c for c in df_conv.columns
                if c.startswith(("sand_", "clay_")) and "gammaSat" not in c]
df_conv[["run_id", "FoS"] + param_cols].head(10)
```

Cell 6b — Sampled Parameter Distributions

Histograms of all sampled parameters with the theoretical distribution overlaid. Confirms that the sampling covered the full range of the defined distributions.

In [ ]:
```
# -- Parameter Distribution Check ---------------------------------------------
# Plots histograms of all sampled parameters with the theoretical
# distribution overlaid. Confirms that sampling covered the full range.

if len(df_conv) == 0:
    print("No converged runs to plot.")
else:
    from scipy.stats import norm as _norm_dist

    param_cols = [c for c in df_conv.columns
                  if any(c.startswith(m.lower() + "_") for m in MATERIALS)
                  and "gammaSat" not in c]

    _SYM = {"phi": r"$\phi$", "cRef": r"$c_{ref}$",
            "gammaUnsat": r"$\gamma_{unsat}$"}

    def _plabel(col):
        parts = col.split("_")
```

```
        mat   = "_".join(parts[:-1]).replace("sand", "Sand").replace("clay", "Clay")
        return f"{mat} {_SYM.get(parts[-1], parts[-1])}"

    ncols = 3
    nrows = (len(param_cols) + ncols - 1) // ncols
    fig, axes = plt.subplots(nrows, ncols, figsize=(5 * ncols, 4 * nrows))
    axes = np.array(axes).flatten()

    for i, col in enumerate(param_cols):
        ax   = axes[i]
        lbl  = _plabel(col)
        data = df_conv[col].values

        ax.hist(data, bins=20, density=True, color="steelblue",
                alpha=0.5, edgecolor="white", label="Sampled")

        # Overlay theoretical distribution if defined in MATERIALS
        parts    = col.split("_")
        param    = parts[-1]
        mat_key  = next((k for k in MATERIALS
                         if k.lower() == "_".join(parts[:-1]).lower()), None)
        if mat_key and isinstance(MATERIALS.get(mat_key, {}).get(param), dict):
            cfg = MATERIALS[mat_key][param]
            if cfg["type"] == "normal":
                x_th = np.linspace(data.min(), data.max(), 200)
                ax.plot(x_th, _norm_dist.pdf(x_th, cfg["mean"], cfg["std"]),
                        color="navy", lw=2, label="Theoretical")

        ax.set_title(lbl, fontsize=9)
        ax.set_xlabel(lbl, fontsize=8)
        ax.set_ylabel("Density", fontsize=8)
        ax.legend(fontsize=7)
        ax.grid(True, alpha=0.2)

    for j in range(i + 1, len(axes)):
        axes[j].set_visible(False)

    fig.suptitle(
        f"Sampled Parameter Distributions — {SAMPLING_METHOD.replace('_', ' ').title()} ({len(df_conv)}
runs)",
        fontsize=11, fontweight="bold"
    )
    plt.tight_layout()
    dist_path = RESULTS_DIR / f"parameter_distributions_{_tag}_{RUN_TIMESTAMP}.png"
    fig.savefig(dist_path, dpi=150, bbox_inches="tight")
    plt.show()
    print(f"Saved to: {dist_path}")
```

Cell 7 — FoS Distribution Plots

Generates two plots saved as separate image files:

Probability Density Function (PDF): Histogram of FoS values with a smooth Kernel Density Estimate (KDE) overlay. The KDE places a Gaussian kernel over each simulated FoS value and sums the resulting curves to produce a continuous estimate of the distribution.

Cumulative Distribution Function (CDF): For any FoS value on the x-axis, the y-axis shows the probability that the actual FoS is below that value.

In [ ]:
```
if len(df_conv) == 0:
    print("No converged runs to plot.")
else:
    fos   = df_conv["FoS"].values
    _meth = SAMPLING_METHOD.replace("_", " ").title()

    # -- Probability Density Function -----------------------------------------
    fig1, ax1 = plt.subplots(figsize=(8, 5))
    ax1.hist(fos, bins=30, density=True, color="steelblue", alpha=0.5,
             edgecolor="white", label="Histogram")
    kde_x = np.linspace(fos.min() - 0.2, fos.max() + 0.2, 300)
    ax1.plot(kde_x, stats.gaussian_kde(fos)(kde_x), color="navy", lw=2, label="KDE")
    ax1.axvline(1.0, color="red",   lw=2, ls="--", label="FoS = 1.0 (failure)")
    ax1.axvline(fos.mean(), color="green", lw=2, label=f"Mean = {fos.mean():.3f}")
    ax1.set_xlabel("Factor of Safety", fontsize=11)
    ax1.set_ylabel("Probability Density", fontsize=11)
    ax1.set_title(
        f"Probability Density Function — {_meth} ({len(df_conv)} runs)\n"
        f"Mean = {fos.mean():.3f}  |  Std = {fos.std():.3f}",
```

```
        fontsize=11, fontweight="bold")
    ax1.legend(fontsize=9)
    ax1.grid(True, alpha=0.3)
    fig1.tight_layout()
    pdf_path = RESULTS_DIR / f"pdf_{_tag}_{RUN_TIMESTAMP}.png"
    fig1.savefig(pdf_path, dpi=150, bbox_inches="tight")
    plt.show()
    print(f"PDF saved to: {pdf_path}")

    # -- Cumulative Distribution Function ------------------------------------
    fig2, ax2 = plt.subplots(figsize=(8, 5))
    fos_s = np.sort(fos)
    cdf   = np.arange(1, len(fos_s) + 1) / len(fos_s)
    ax2.plot(fos_s, cdf * 100, color="steelblue", lw=2)
    ax2.axvline(1.0, color="red", lw=2, ls="--", label="FoS = 1.0 (failure)")
    pf = (fos < 1.0).mean()
    if pf > 0:
        ax2.fill_between(fos_s[fos_s <= 1.0], cdf[fos_s <= 1.0] * 100,
                         alpha=0.3, color="red",
                         label=f"P(FoS < 1.0) = {pf*100:.2f}%")
    ax2.set_xlabel("Factor of Safety", fontsize=11)
    ax2.set_ylabel("Probability [%]", fontsize=11)
    ax2.set_title(
        f"Cumulative Distribution Function — {_meth} ({len(df_conv)} runs)\n"
        f"Mean = {fos.mean():.3f}  |  Std = {fos.std():.3f}",
        fontsize=11, fontweight="bold")
    ax2.legend(fontsize=9)
    ax2.grid(True, alpha=0.3)
    fig2.tight_layout()
    cdf_path = RESULTS_DIR / f"cdf_{_tag}_{RUN_TIMESTAMP}.png"
    fig2.savefig(cdf_path, dpi=150, bbox_inches="tight")
    plt.show()
    print(f"CDF saved to: {cdf_path}")
```

Cell 8 — Parameter Sensitivity

Shows the relationship between each randomised soil parameter and the resulting Factor of Safety.

The Pearson correlation coefficient r quantifies the strength of the linear relationship. Values close to ±1 indicate a strong influence on the FoS, while values close to 0 indicate little influence. This analysis identifies which parameters most warrant careful characterisation in the site investigation.

In [ ]:
```
if len(df_conv) == 0:
    print("No converged runs to plot.")
else:
    param_cols = [c for c in df_conv.columns
                  if c.startswith(("sand_", "clay_")) and "gammaSat" not in c]

    # Symbol mapping for axis labels
    _SYM = {"phi": r"$\phi$", "cRef": r"$c_{ref}$",
            "gammaUnsat": r"$\gamma_{unsat}$", "gammaSat": r"$\gamma_{sat}$"}

    def _label(col):
        parts = col.split("_")
        mat   = "_".join(parts[:-1]).replace("sand", "Sand").replace("clay", "Clay")
        return f"{mat} {_SYM.get(parts[-1], parts[-1])}"

    ncols = 3
    nrows = (len(param_cols) + ncols - 1) // ncols
    fig, axes = plt.subplots(nrows, ncols, figsize=(5 * ncols, 4 * nrows))
    axes = np.array(axes).flatten()

    for i, col in enumerate(param_cols):
        ax   = axes[i]
        lbl  = _label(col)
        corr = df_conv[col].corr(df_conv["FoS"])
        ax.scatter(df_conv[col], df_conv["FoS"],
                   alpha=0.35, s=20, color="steelblue", edgecolors="none")
        m, b   = np.polyfit(df_conv[col], df_conv["FoS"], 1)
        x_line = np.linspace(df_conv[col].min(), df_conv[col].max(), 100)
        ax.plot(x_line, m * x_line + b, color="red", lw=1.5, ls="--")
        ax.set_title(f"{lbl}\nPearson r = {corr:.2f}", fontsize=9)
        ax.set_xlabel(lbl, fontsize=8)
        ax.set_ylabel("Factor of Safety", fontsize=8)
        ax.axhline(1.0, color="red", lw=0.8, ls=":")
        ax.grid(True, alpha=0.2)

    for j in range(i + 1, len(axes)):
        axes[j].set_visible(False)
```

```
    fig.suptitle(
        f"Parameter Sensitivity — {SAMPLING_METHOD.replace('_', ' ').title()} ({len(df_conv)} runs)\n"
        "Pearson r closer to ±1 indicates stronger influence on FoS",
        fontsize=11, fontweight="bold"
    )
    plt.tight_layout()
    sens_path = RESULTS_DIR / f"sensitivity_{_tag}_{RUN_TIMESTAMP}.png"
    plt.savefig(sens_path, dpi=150, bbox_inches="tight")
    plt.show()
    print(f"Sensitivity plot saved to: {sens_path}")
```

Cell 9 — CDF Excel Export

Exports the CDF data to an Excel file with an embedded scatter chart. The chart format matches the report style with Factor of Safety on the x-axis (0.9–2.0) and Probability [%] on the y-axis (0–100%).

In [ ]:
```
if len(df_conv) == 0:
    print("No converged runs to export.")
else:
    from openpyxl import load_workbook
    from openpyxl.chart import ScatterChart, Reference, Series
    from openpyxl.chart.axis import ChartLines

    fos_sorted = np.sort(df_conv["FoS"].values)
    cdf_values = np.arange(1, len(fos_sorted) + 1) / len(fos_sorted)

    df_cdf = pd.DataFrame({
        "Factor of Safety": fos_sorted,
        "Probability [%]":  cdf_values * 100,
    })

    cdf_xlsx = RESULTS_DIR / f"cdf_{_tag}_{RUN_TIMESTAMP}.xlsx"
    df_cdf.to_excel(cdf_xlsx, index=False, engine="openpyxl")

    wb = load_workbook(cdf_xlsx)
    ws = wb.active

    chart = ScatterChart()
    chart.title  = f"Cumulative Distribution Function — {SAMPLING_METHOD.replace('_', ' ').title()}"
    chart.style  = 2
    chart.legend = None
    chart.width  = 20
    chart.height = 15

    chart.x_axis.title          = "Factor of Safety"
    chart.y_axis.title          = "Probability [%]"
    chart.x_axis.scaling.min    = 0.9
    chart.x_axis.scaling.max    = 2.0
    chart.y_axis.scaling.min    = 0.0
    chart.y_axis.scaling.max    = 100.0
    chart.x_axis.majorGridlines = ChartLines()
    chart.y_axis.majorGridlines = ChartLines()

    x_data = Reference(ws, min_col=1, min_row=2, max_row=len(df_cdf) + 1)
    y_data = Reference(ws, min_col=2, min_row=2, max_row=len(df_cdf) + 1)
    series = Series(y_data, x_data)
    series.marker.symbol = "none"
    series.graphicalProperties.line.solidFill = "1F77B4"
    series.graphicalProperties.line.width     = 20000

    chart.series.append(series)
    ws.add_chart(chart, "D2")
    wb.save(cdf_xlsx)

    print(f"CDF Excel saved to: {cdf_xlsx}")
    print(f"  Rows      : {len(df_cdf)}")
    print(f"  FoS range : {fos_sorted[0]:.3f} – {fos_sorted[-1]:.3f}")
    print(f"  P(FoS<1.0): {(fos_sorted < 1.0).mean()*100:.2f} %")
```

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY