Ole Eiesland

Implementation of random finite element method in Plaxis Using API / Python scripting

With examples on slope stability and bearing capacity problems

Trondheim, December 2021

PROJECT THESIS: TBA4510

Main supervisor: Professor Yutao Pan

Department of Civil and Transport Engineering

Norwegian University of Science and Technology (NTNU)

NTNU – Trondheim
Norwegian University of
Science and Technology

Preface

As a pre study for my masters thesis this project work is presented as final deliverable in the subject "TBA4510 - Geotechnical Engineering, Specialization Project" at the Department of Civil and Transport Engineering, NTNU during the spring semester of 2021. The course is 7.5 SP credits.

I would like to thank professor Yutao Pan, NTNU for supervising the project.

Trondheim, 2021-12-20

Ole Eiesland

Summary and Conclusions

Soil is a complex medium. Its inhomogeneous nature means that the physical parameters of soil vary spatially both vertically and laterally. Traditionally soil properties is modeled with a representative value, usually some kind of mean value or similar. In probabilistic methods, this variability, or uncertainty is taken into account by treating the soil as a random variable sampled from a probability distribution. By using random field theory and statistics one can try to describe how the soil parameters are distributed in space and how they vary with distance. In a slope stability or a bearing capacity problem, the spatial distribution of the soil strength governing parameters has a direct impact on the development of the failure surface, the failure mechanism and therefore the over all stability. To simulate stability a finite element program can be used with the soil model parameters input to the finite element mesh based on statistical spatial correlated random fields. To simulate variability and uncertainty, the modeling is repeated many times with different random fields. This is the random finite element method.

Current modern commercial soil modeling software do not support random finite element method, and published research random finite element software code do not have the advanced functionality and soil behaviour models as modern commercial geotechnical software. How ever, since the random finite element method does not change the way the problem is simulated, only the input parameters change, modern software packages can be used if a way to specify the input and the simulation run parameters can be controlled in an automatic and efficient manner. Plaxis, a modern software package developed by Bently, has capabilities like this through its application program interface and python scripting.

This project work presents a method to run the random finite element method in Plaxis geotechnical software package using python API scripting interface, and demonstrate the versatility and verifies the validity of the implementation of the random finite element method on a slope stability and a bearing capacity problem.

Contents

Preface i

Summary and Conclusions ii

1 Introduction **2**

1.1 Background 2

1.2 Objectives 4

1.3 Limitations 4

1.4 Approach 5

1.5 Structure of the Report 5

2 Theory **6**

2.1 Random Fields 6

2.1.1 SRM - Spectral Representation Method 7

2.2 Finite Element Method 8

3 RFEM implementation in Plaxis 2D using python API **10**

3.1 SRM implementation 10

3.2 Local averaging 13

3.3 Monte Carlo 13

3.4 Plaxis 2D 14

4 Results **15**

4.1 Slope Stability problem - Homogeneous isotropic soil - Zero variance 15

4.2 Slope stability problem - Homogeneous anisotropic soil - Scale of Fluctuation 20 and 10 meters, CoV=0.3 - Monte Carlo 250 realizations 18

4.3 Bearing Capacity Problem - Homogeneous isotropic soil - CoV=0.0 - Known analytical solution . . . 20
4.4 Bearing Capacity Problem - Homogeneous anisotropic soil - Scale of Fluctuation 8 and 3 meters, CoV=0.1, Monte Carlo 250 realizations . . . 25

5 Summary **28**
5.1 Summary and Conclusions . . . 28
5.2 Discussion . . . 29
5.3 Recommendations for Further Work . . . 29

A Acronyms **31**

B Source code **32**
B.1 Introduction . . . 32
B.1.1 Easy Python scripting interface - Jupyer notebook . . . 38

Bibliography **40**

Chapter 1

Introduction

The first chapter of a well-structured thesis is always an introduction, setting the scene with background, problem description, objectives, limitations, and then looking ahead to summarize what is in the rest of the report. This is the part that readers look at first—*so make sure it hooks them!*

1.1 Background

The inhomogeneous nature of soil means that the physical parameters vary spatially both vertically and laterally. Traditionally soil properties is modeled with a representative value, usually some kind of mean value or similar. Common cause of deviation of expected performance of geotechnical design is variability in soil properties at the site. In probabilistic methods, this variability, or uncertainty is taken into account by treating the soil as a random variable sampled from a probability distribution. By using random field theory and statistics one can try to describe how the soil parameters are distributed in space and how they vary with distance. In a slope stability or a bearing capacity problem, the spatial distribution of the soil strength governing parameters has a direct impact on the development of the failure surface, the failure mechanism and therefore the over all stability. To simulate stability a finite element program can be used with the soil model parameters input to the finite element mesh based on statistical spatial correlated random fields. To simulate variability and uncertainty, the modeling is repeated many times with different random fields. This is the random finite element method.

Problem Formulation

Current modern soil modeling software do not support random finite element method, and published research random finite element software code do not have the advanced functionality as modern commercial geotechnical software. How ever, since the random finite element method does not change the way the problem is simulated, only the input parameters change, modern software packages can be used if a way to specify the input and the simulation run parameters can be controlled in an automatic and efficient manner. Plaxis, a modern software package developed by Bently, or Optum G2, by has capabilities like this through its application program interface and python scripting. It is of great interest to research on spatially varying soil modeling to utilize the advanced functions and ease of access of existing software. Gaining this ability will expand the complexity of the problems allowed to be simulated by the RFEM method. The current userbase of geotechnical engineers using Plaxis is large and providing a RFEM tool in plaxis could allow for them to incorporate probabilistic methods in their designs.

Literature Survey

The random finite element method (RFEM) has been in use since the mid-1990s [e.g, see Griffiths and Fenton, 1993]. RFEM combines random field theory to represent the spatially varying soil with finite element method (FEM) for deformation analysis. Stochastic analysis in FEM methods can be built into the finite element equations themselves [e.g., see Vanmarcke and Grigoriu, 1983], or a Monte Carlo approach can be used were multiple realizations of different spatial soil models can be analyzed together. The Monte Carlo approach can be computationally demanding, but can be very flexible by utilizing arbitrary FEM code changing just the input. The Monte Carlo RFEM and its application to many geotechnical problems including seepage, bearing capacity, earth pressure and settlement is described in more detail in book by Fenton, Griffiths, et al. [2008]. Example of RFEM analysis for slope stability is presented by [see Fenton, Griffiths, et al., 2008, Chapter 13] based on FEM code developed by Smith, Griffiths, and Margetts [2013]. The RFEM code is publicly available and extensive research has been conducted using it. A list of all but the most recent publications using the code is available here: (`http://random.engmath.dal.ca/rfem/rfem_pubs.html`). The RFEM code is for two-dimensional plane strain analysis of elastic perfectly plastic soils with a Mohr Coulomb failure criterion. For a detailed discussion of the method [e.g, see Griffiths and Lane, 1999]. The limita-

tion of the failure criteria and the soil model can restrict the application of the method to more complex soils who displays different characteristics. Software with a range of failure criteria and soil models exist, such as Plaxis by Bently [Brinkgreve et al., 2010] or Optum G2 [Krabbenhøft et al., 2016]. Plaxis do not have built in random field functionality.

What Remains to be Done?

To be able to describe more complex geotechnical problems, a method to unify material models and numerical methods is needed. Implementation of new soil models into FEM code is not straight forward. Plaxis allows for python scripting through an application programming interface (API). The RFEM Monte Carlo method could be implemented in Plaxis by scripting random field input and automating simulation.

1.2 Objectives

The main objectives of this project are

1. Implement The RFEM Monte Carlo method in Plaxis using python API interface
2. Demonstrate and verify the implementation on a simple slope stability problem
3. Demonstrate versatility by extending the implementation to a simple bearing capacity problem
4. Compare results to analytical results were available

1.3 Limitations

The field of random behavior of soils in geotechnical problems is extensive and this project work scope only scratches in the surface. The methodology of finite element numerical computations and random fields is only described briefly and beyond the scope of this project course to go into in any detail.

The python code presented is not attempted optimized in any way, the focus is on proof of concept. The result presented is a tool, further work, research and application will prove its usability.

1.4 Approach

A literature search was conducted to get an overview of the current implementations of the RFEM code. Source code and accompanying documentation was also studied where available. The main part of the project was to code the implementation of the RFEM method into the Plaxis 2D software using the python scripting API. Study of the Plaxis 2D software manual and online documentation was key to get familiar with Plaxis and the API functions to control and automate the program execution from the python code. Example problems from literature is used to verify the validity of the implementation. Results from other software is reproduced to compare results. Analytical results is compared to the implementation. To demonstrate versatility a slope stability and bearing capacity problem is used.

1.5 Structure of the Report

The rest of the report is structured as follows. Chapter 2 gives an brief introduction to random field theory and finite element analysis. In chapter 3 the implementation of the RFEM method in Plaxis using python API script is discussed. Chapter 4 compares results of simulations using the python API implementation of the RFEM method in Plaxis. Chapter 5 gives a summary and discussion of the results and main findings of the project work.

Chapter 2

Theory

This chapter gives an brief introduction to random field theory and finite element analysis.

2.1 Random Fields

To model the spatial variability of soils random fields are used. To describe the random field of a soil parameter, e. g. the undrained shear strength of the soil, three parameters are commonly used. The Mean, μ, Standard deviation, σ of the soil parameters underlying probability distribution and the Spatial correlation length, θ also known as the scale of fluctuation. The mean is a measure of around which value the soil strength parameter is distributed. The standard deviation tells how much the values in the soil random fields varies in values. The spatial correlation length, or scale of fluctuation, is a measure of how similar in strength points in the spatial random field are depending on how far away the points are from each other. Large spatial correlation length vary smoothly and varying slowly, while a small scale of fluctuation is jagged and rapidly varying. Soil samples taken close together is more likely to be similar than soil samples taken far apart. Du to the process of soil deposition the soil tend to have different properties in differing direction, soil is anisotropic. Typically for layered soils, the soil properties are more similar in the horizontal direction than in the vertical direction. This anisotropy can be represented by having a different scale of fluctuation in the horizontal direction than in the vertical direction.

Remark:There is a difference between homogeneous media and isotropic media, a homoge-

neous media can be anisotropic, but the anisotropy parameter does not change depending on where they are measured.

2.1.1 SRM - Spectral Representation Method

Various methods exist for generating random fields that can be used to represent spatially variable soil. [see e. g. Fenton, Griffiths, et al., 2008, Chapter 6]. The methods differ in their efficiency and accuracy and complexity e. g. ability to describe anisotropy. In this project work the spectral representation method is used. It is showed by Shinozuka and Deodatis [1996] how to simulate multi-dimensional homogeneous stochastic fields using the spectral representation method. Sample functions of the stochastic field can be generated using a cosine series formula. These sample functions accurately reflect the prescribed probabilistic characteristics of the stochastic field when the number of terms in the cosine series,N_1 and N_2 in equation 2.1, is large.

$$f_0(x_1,x_2)=\sqrt{2}\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1}A_{n_1n_2}[cos(\kappa_{1_{n_1}}x_1+\kappa_{2_{n_2}}x_2+\Phi^{(1)}_{n_1n_2})+cos(\kappa_{1_{n_1}}x_1-\kappa_{2_{n_2}}x_2+\Phi^{(2)}_{n_1n_2})] \quad (2.1)$$

were:

$$A_{n_1n_2}=\sqrt{2S_{f_0f_0}(\kappa_{1_{n_1}},\kappa_{2_{n_2}})\Delta\kappa_1\Delta\kappa_2} \quad (2.2)$$

$$\kappa_{1_{n_1}}=n_1\Delta\kappa_1 \quad \text{and} \quad \kappa_{2_{n_2}}=n_2\Delta\kappa_2 \quad (2.3)$$

$$\Delta\kappa_1=\frac{\kappa_{1u}}{N_1} \quad \text{and} \quad \Delta\kappa_2=\frac{\kappa_{2u}}{N_2} \quad (2.4)$$

and:

$$A_{0n_2}=A_{n_10}=0 \quad \text{for} \quad n_1=n_2=0,1,\ldots,N_1-1 \quad (2.5)$$

$\Phi^{(1)}_{n_1n_2}$ and $\Phi^{(2)}_{n_1n_2}$ are random phase angles distributed uniformly over the interval. $[0,2\pi]$. $S_{f_0f_0}(\kappa_{1_{n_1}},\kappa_{2_{n_2}})$ is the power spectral density function and is a real non-negative function of the wave numbers κ_1 and κ_2. κ_{1u} and κ_{2u} are the upper cut-off wave numbers.

To avoid aliasing, the following condition must be true:

$$\Delta x_1 \leq \frac{2\pi}{2\kappa_{1u}} \quad \text{and} \quad \Delta x_2 \leq \frac{2\pi}{2\kappa_{2u}} \tag{2.6}$$

See chapter 3 for details on implementation.

2.2 Finite Element Method

The Finite Element method is a numerical method that can be used to solve a multitude of geotechnical and other engineering problems. Being a numerical method, FEM gives approximate solution, care should be used when constructing the model to be calculated. Things to keep in mind when running FEM analysis is the volume extent and the boundary interface of the model, element type, density of mesh and criteria of convergence. In comparison to limit equilibrium methods (LEM), FEM methods can give deformations in a state before ultimate limit state i.e. before failure. FEM have no prior assumptions of the failure mechanism i.e. no assumption of a circular failure surface in a LEM slope stability analysis. It is a very attractive property of FEM analysis combined with spatial variable soil, to allow slope failure to develop naturally by finding the path of weakest soil.

The main principles of the FEM is to divide the soil into smaller elements, then develop description of what happening in each element, then add the behaviour of each element together to find the behaviour of the whole volume.

Seven steps of a finite element program is described by Nordal [2020]:

1. Element modeling, equations are built for each element, integration in points inside the element is used to form stiffness matrix

2. Global modelling, all element stiffness matrices are assembled into a system of equations to form a global stiffness matrix. An incremental load vector is found.

3. Equation solving of the global equation system for the load increment. This gives a displacement increment

4. Stress evaluation of the calculated displacement increment. The displacements are used to find the strains which is used to find the stress increments

5. Testing for numerical accuracy. If the calculations show too high unbalanced forces, it will be necessary to adjust the load increment and/or add more iterations. If so step 1-5 must be recalculated. When the results converges the program proceeds to step 6.

6. Updating of results by adding the deformations and stress to form total deformations and total stresses.

7. Calculation of new load increment. The response of the new load increment is found by repeating 1 to 6. New load increment is repeatedly added until the specified external load is reached, or failure occur.

Remark:Plaxis measures the fraction of the applied loadstep according to step 7 above in a variable which in this project is accessed to automatically tell if we have failure or not.

Chapter 3

RFEM implementation in Plaxis 2D using python API

This chapter discusses the implementation of the RFEM method in Plaxis using python API scripting.

3.1 SRM implementation

In generating the random soil strength field an implementation of the SRM method written in MATLAB code by Yutao Pan out of Deodatis, Shinozuka, and Papageorgiou [1990] is converted to python code. The implementation in this project uses a Gaussian spectral density function given below in equation 3.2. Other spectral density functions exists and can be implemented.

$$S_{f_0 f_0}(\kappa_{1_{n_1}}, \kappa_{2_{n_2}}) = \frac{1}{4}\frac{\theta_x}{\pi}\frac{\theta_y}{\pi} e^{(-(\frac{\theta_x^2 \kappa_1^2}{4\pi})} e^{(-(\frac{\theta_y^2 \kappa_2^2}{4\pi})} \tag{3.1}$$

```
import numpy as np
def srm():
    inmiu=20
    incov=0.1
    sig=inmiu*incov;
    SOFx=8
    SOFy=3
    b1=SOFx/np.sqrt(np.pi)
    b2=SOFy/np.sqrt(np.pi)
    m1=64 #integration mesh
    m2=64
```

```
    m11=m1-1
    m22=m2-1
    bk1u=5
    bk2u=5
    k1u=bk1u/b1
    k2u=bk2u/b2

    detak1=k1u/m1
    detak2=k2u/m2
    g=0.0
    nrep=1

    nnx=30     #50#x direction number of mesh X
    nny=10     #50#y dir number of mesh

    nnx1=nnx +1
    nny1=nny +1
    tt=np.zeros((nnx1,nny1))
    RR=np.zeros((nnx1,nny1))
    gg=np.zeros((nrep,nnx1,nny1)) #realization at different points
    xx=np.zeros(nnx1) # x dir coor
    yy=np.zeros(nny1) # y dir coor

    lx=30 # length x dir
    ly=10 # length y dir
    spx=lx/nnx
    detax=np.pi/k1u
    spy=ly/nny
    detay=np.pi/k2u
    sq2=np.sqrt(2)
    intsd=0
    intsdf=np.zeros((nnx,nny)) #integrate sdf at different points
    lx
    limitx=period1/4
    ly
    limity=period2/4
    rng = np.random.default_rng(42)

    for nround in range(0,nrep):
        rand1=np.random.rand(m1,m2)
        rand2=np.random.rand(m1,m2)
        fai1=rand1*2*np.pi
        fai2=rand2*2*np.pi
        for nx in range(0,nnx+1):
            for ny in range(0,nny+1):

```

```
58
59              nx1=nx#+1 -py
60              ny1=ny#+1 -py
61              x=nx*spx
62              y=ny*spy
63              xx[nx1]=x
64              yy[ny1]=y
65
66
67
68              for n1 in range(0,m11):
69                  for n2 in range(0,m22):
70                      n1j=n1+1
71                      n2j=n2+1
72                      k1=n1*detak1
73                      k2=n2*detak2
74                      k3=(n1+1)*detak1
75                      k4=(n2+1)*detak2
76                      sdf=b1*b2*(np.e**(-(b1*k1/2)**2-(b2*k2/2)**2)+np.e**(-(b1*k3/2)**2-(
    b2*k4/2)**2)+np.e**(-(b1*k1/2)**2-(b2*k4/2)**2)+np.e**(-(b1*k3/2)**2-(b2*k2/2)**2))
    /16.0/np.pi
77
78                      A1=np.sqrt(2*sdf*detak1*detak2)
79                      A2=A1#sqrt(2*sdf2*detak1*detak2)
80
81                      g=g+A1*np.cos(k1*x+k2*y+fai1[n1j,n2j])+A2*np.cos(k1*x-k2*y+fai2[n1j,
    n2j])
82
83                      intsd=intsd+sdf*detak1*detak2*4
84
85                      if ((n1==0) and (n2==0)):
86                          sdf00=sdf
87              intsd=0;
88              gg[nround,nx1,ny1]=inmiu+g*sq2*sig
89              tt[nx1,ny1]=g*sq2
90              g=0
91  tt=(tt*incov+1)*inmiu;
92  tttrans=np.transpose(tt)
93  return tttrans
```

Listing 3.1: SRM code

3.2 Local averaging

When e.g. a clay sample is sheared in the laboratory to determine strength parameters, failure develops over the whole sample when the bonds of the sample yield. The measured strength is a function of the average bond strength of the sample. The greater the sample size the stronger is the averaging effect. Input parameters in modeling, in the case of RFEM, the mean, standard deviation and spatial correlation length are assumed to be point measures. Therefore when populating a RFEM model spatial averaging needs to be taken into account since the element sizes is in general much grater than the size of the sample from where the parameter was derived. It can be shown [Vanmarcke, 2010] that the reduction in variance due to local averaging is given by:

$$\sigma_A = \sigma\sqrt{\gamma} \tag{3.2}$$

where σ_A is the new spatially averaged variance that is to be used when drawing samples from the distribution to put into the finite element mesh and γ is the variance reduction function, defined for a rectangular element as:

$$\gamma = \frac{4}{T_x^2 T_y^2}\int_0^{T_x}\int_0^{T_y}(T_x-\tau_x)(T_y-\tau_y)\rho(\tau_x,\tau_y)d\tau_x d\tau_y \tag{3.3}$$

where T_x and T_y is the size of the element in the x and y direction respectively, ρ is the correlation function and τ_x and τ_y is the difference between the, respectively, the x and y coordinates of any two points in the random field.

γ has the value of 1.0 when $T = 0$ [see Fenton et al., 2008, chapter 3]. Setting $T = \alpha\theta$ (i.e some scalar α times the scale of fluctuation θ) leads to the conclusion that elements much smaller than the scale of fluctuation is affected to small degree by variance reduction.

In this project work this realization is used.

3.3 Monte Carlo

Monte Carlo is a method that can estimate the means, variances and probabilities of the responses of complex systems to random input [see Fenton et al., 2008, chapter 6.6]. Consider the random response of a system $g(X_1, X_2)$ where X_1 and X_2 are random variables. The system fails if the value of $g(X_1, X_2) > g_{critical}$. Monte Carlo simulates a sequence of realizations of X_1 and

X_2, evaluates $g(X_1, X_2)$ and checks if $g(X_1, X_2) > g_{critical}$.

The method is very versatile and can be applied to most kinds of systems. Drawbacks is that there are no analytical solution, if the system, e.g. the input is changed, the simulation must be rerun, we can not predict the response to a change in input. Also, to simulate rear events, a lot of simulations is needed which can be computational demanding.

3.4 Plaxis 2D

The following procedure to run the RFEM Monte Carlo analysis is implemented in Plaxis using the python API. The user specify the input soil parameters, the problem geometry and the desired number of realizations. One realization is one stability simulation on one random field. The procedure is run fully autonomous without user interaction.

1. The script start by creating a new empty Plaxis project
2. Next, the problem geometry is generated i.e. the slope height, inclination and extent.
3. The random field representing the soil property is generated based on the user specified mean, standard deviation (or CoV) and scale of fluctuation
4. The soil elements are populated with the soil parameter values from the random field.
5. Then Plaxis grids the geometry generating the FEM mesh
6. Next a new phase is added after the initial phase, activating the geometry represented by the soil
7. Finally the deformations are calculated and a result is stored, i.e. failure or no failure in a slope stability problem
8. Now step 3 to 7 is repeated generating a new realization of the random field, a user specified number of times
9. In the end statistics are gathered and written to a file, and plots of the results can be displayed

Chapter 4

Results

To test the validity of the implementation a series of slope stability simulations is run. The results from these simulations is presented in the following sections.

4.1 Slope Stability problem - Homogeneous isotropic soil - Zero variance

To check the base case of spatially invariant soil with known analytical solutions. This is to validate the input to the random field generation, the FEM mesh and calculation. The simulation is a elasto-plastic FEM simulation with Mohr-Couloumb material model using Plaxis undraind(C) behaviour. 15-Node triangular FEM elements are used. The slope is 5 meter high with a 2:1 gradient. The soil parameters is presented in Table 4.1. The resulting random field, or in this particular case a constant field, is shown in Figure 4.1 and the 2:1 slope geometry and Plaxis 2D mesh is shown in Figure 4.2.

Table 4.1: Soil parameters for Homogeneous isotropic soil

Soil model	Mohr Couloumb - Undrained(C)		
Statistical Soil Parameters	Mean μ	Coefficient of Variation $CoV = \frac{\sigma}{\mu}$	Scale of fluctuation θ
Unit weight, $\gamma_{sat} = \gamma unsat$	20 kN/m^3	0	-
Modulus of elasticity, E	10 MPa	0	-
Poissons ratio, ν	0.49	0	-
Undrained Shear Strength,S_u	20 kPa	0	-

Running a Plaxis c-reduction calculation phase, result graph plotted in 4.3, on the uniform soil slope gives a Factor of Safety, $F_s = 1.14$. The corresponding F_s obtained by slope stability charts after Janbu [1968] is $F_s = 1.16$.

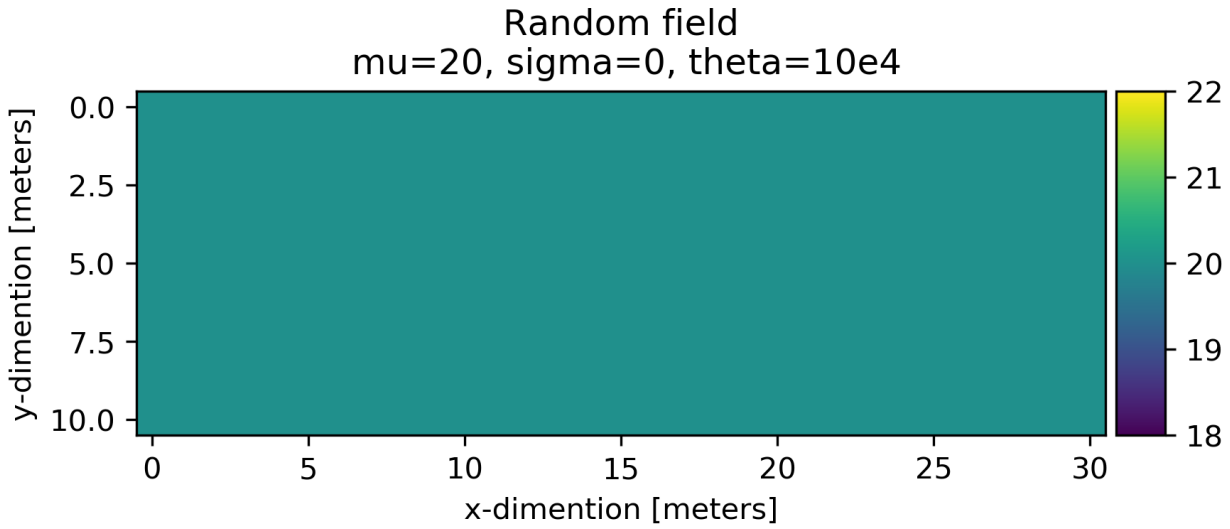


Figure 4.1: S_u Random field, in this particular case the soil strength is uniform

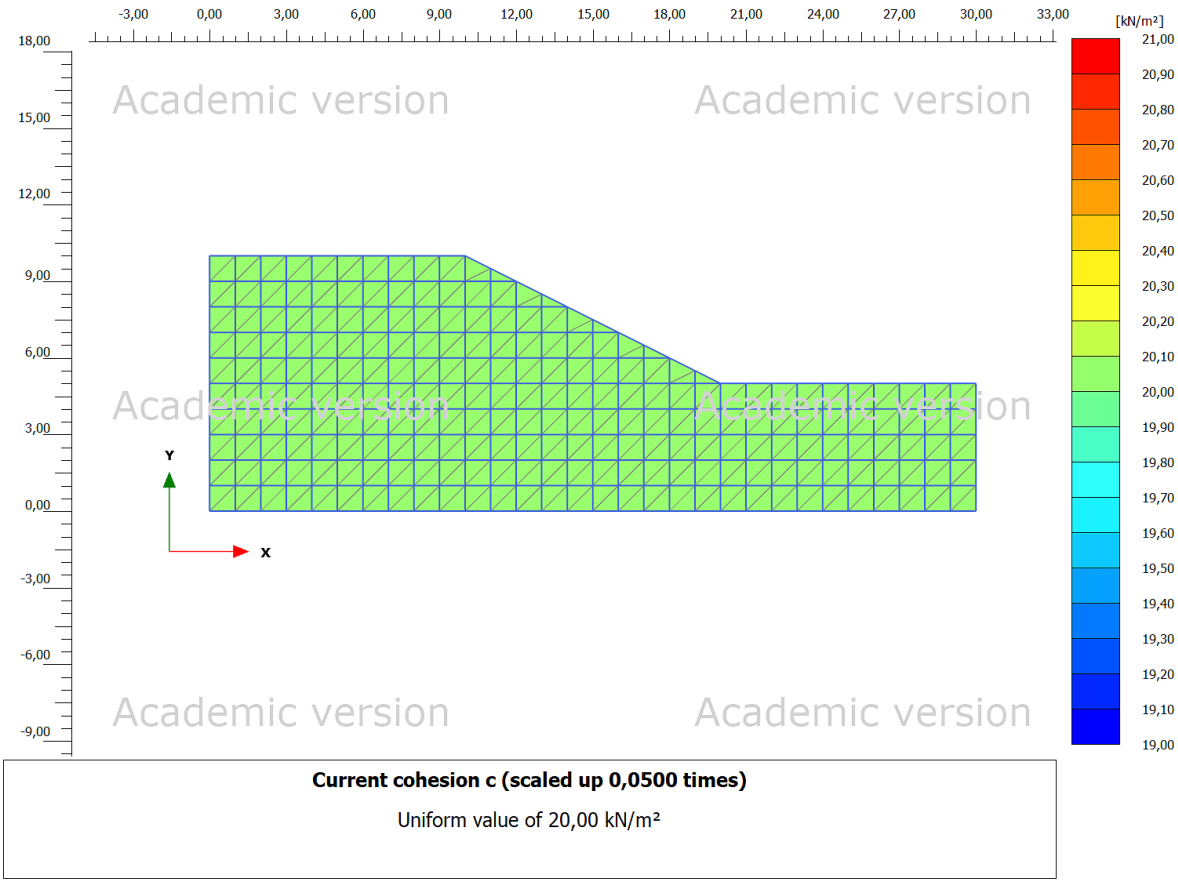


Figure 4.2: Slope geometry, with soil strength property from the random field mapped to the soil Plaxis soil elements and triangular FEM elements displayed

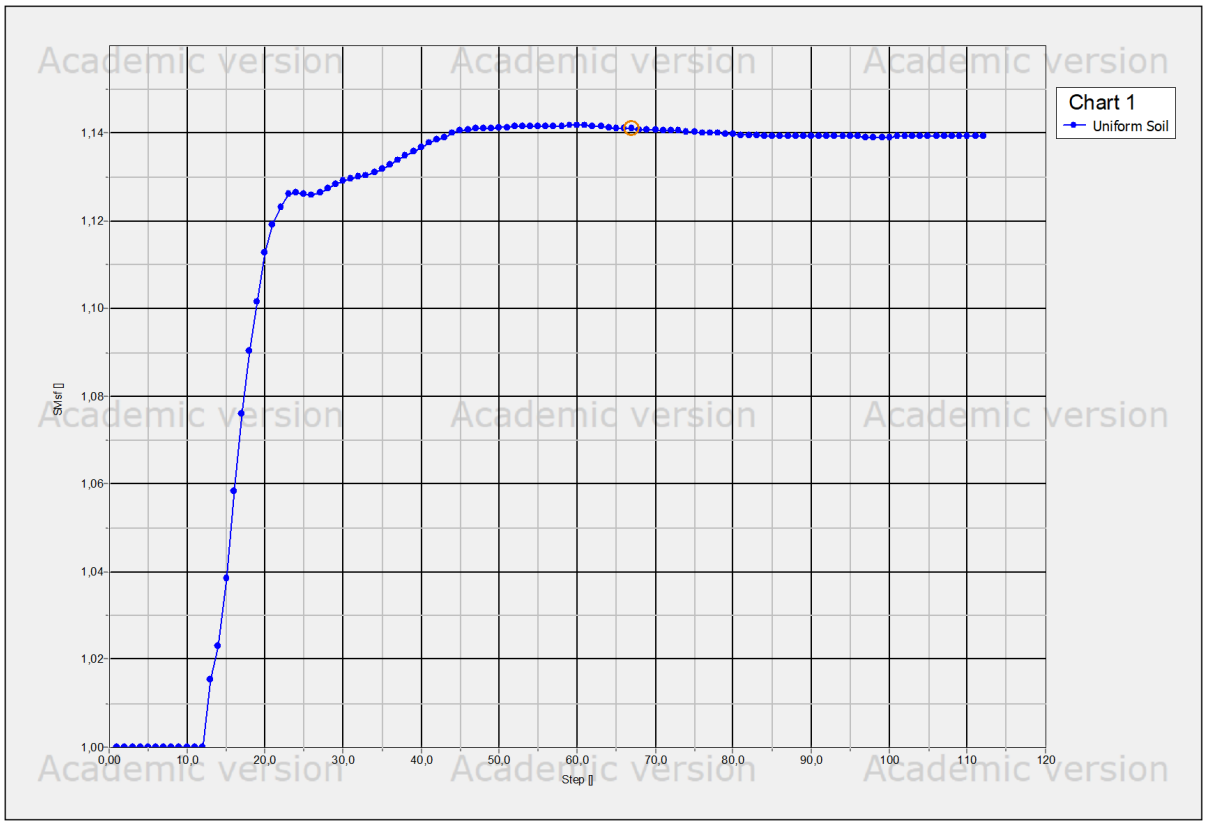


Figure 4.3: Plaxis Safety factor

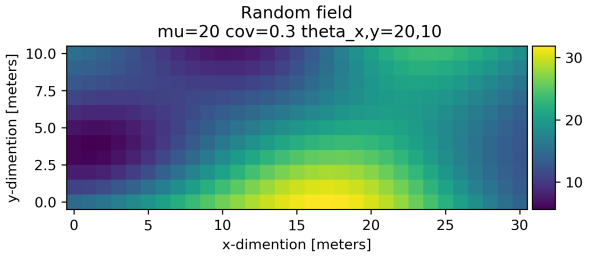
4.2 Slope stability problem - Homogeneous anisotropic soil - Scale of Fluctuation 20 and 10 meters, CoV=0.3 - Monte Carlo 250 realizations

Following is the results from 250 realizations and simulations using random field as undrain strength parameters. The simulations are elasto-plastic FEM simulation with Mohr-Coulomb material model using Plaxis undraind(C) behaviour. 15-Node triangular FEM elements are used. The slope is 5 meter high with a 2:1 gradient. The soil parameters is presented in Table 4.2. Examples of resulting random field is shown in 4.4.

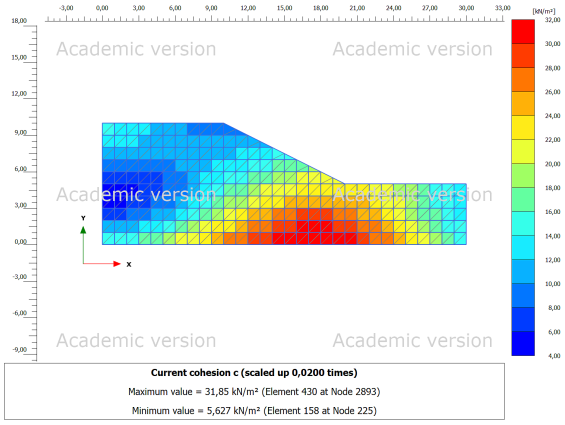
Figure 4.5 shows the probability of failure of the slope vs iteration number.

Table 4.2: Soil parameters for anisotropic soil

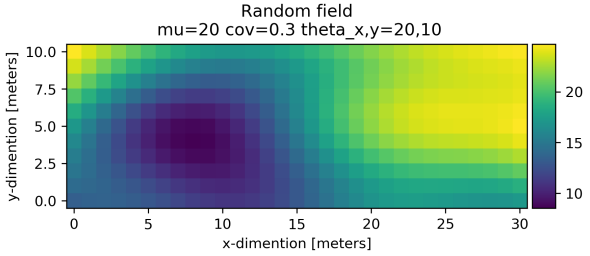
Soil model	Mohr Coulomb - Undrained(C)		
Statistical Soil Parameters	Mean μ	Coefficient of Variation $CoV = \frac{\sigma}{\mu}$	Scale of fluctuation θ_x, θ_y
Unit weight, $\gamma_{sat} = \gamma unsat$	20 kN/m^3	0	-,-
Modulus of elasticity, E	10 MPa	0	-
Poissons ratio, ν	0.49	0	-
Undrained Shear Strength,S_u	20 kPa	0.3	20,10



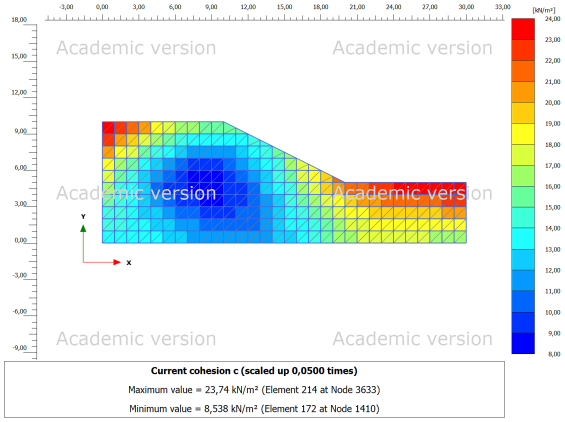
(a) Realization of undrained strength field, iteration 45 of 100



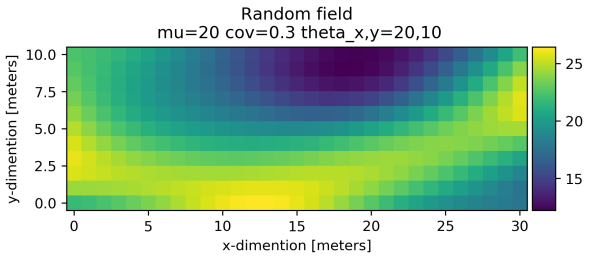
(b) Realization of undrained strength field, iteration 45 of 250



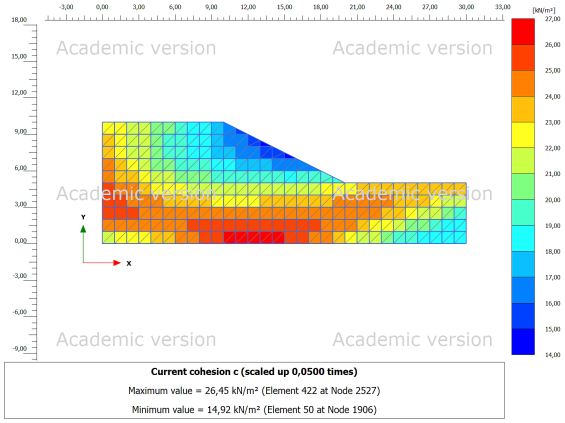
(c) Realization of undrained strength field, iteration 70 of 250



(d) Realization of undrained strength field, iteration 70 of 250



(e) Realization of undrained strength field, iteration 81 of 250



(f) Realization of undrained strength field, iteration 81 of 250

Figure 4.4: Three different realizations of random fields created by SRM and their mapping to the FEM mesh grid.

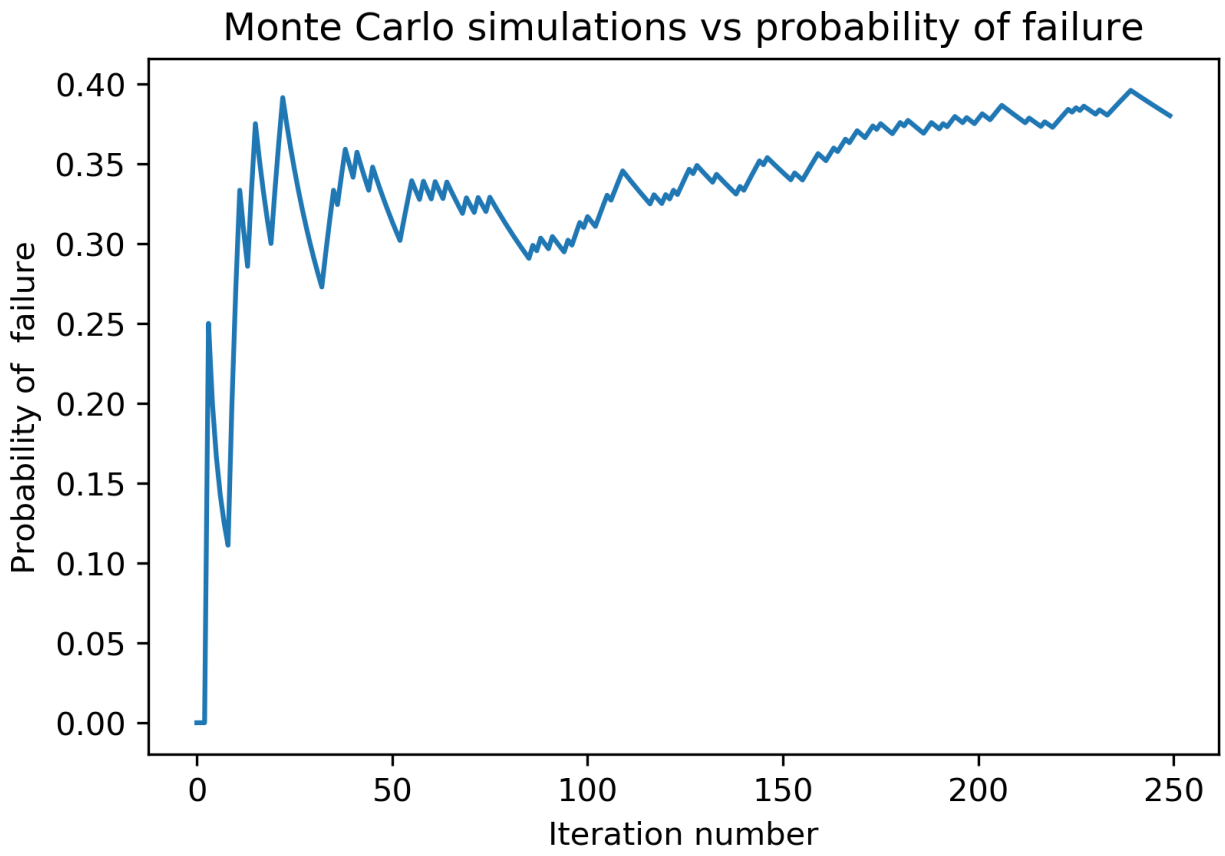


Figure 4.5: Iteration number vs probability of failure for the Monte Carlo RFEM slope stability problem. The failure probability converges towards a value of around 0.38. More iterations might be needed to see if the result is stable.

4.3 Bearing Capacity Problem - Homogeneous isotropic soil - CoV=0.0 - Known analytical solution

To illustrate the versatility of the implementation, the slope stability problem is changed to a bearing capacity problem by specifying a flat topology and adding a line load. This change is done wit the change of 2 lines of code. To verify the validity, again a base case with an analytical solution is set up. Undrained bearing capacity for a smooth foundation with the known bearing capacity:

$$q = 5.14 S_u \tag{4.1}$$

The simulation is a elasto-plastic FEM simulation with Mohr-Coulomb material model using Plaxis undraind(C) behaviour. 15-Node triangular FEM elements are used. A line load of 1000kPa and length of 9 meters is applied to the top of the soil. The soil parameters is presented in Table

4.3. Internal in the Plaxis code, ref step 6 in chapter 3.3, the 1000 kPa load is applied in increments, the fraction of the load applied is given in in the Plaxis variable called $\sum Mstage$. see Figure 4.6 for simulation result. The bearing capacity of the soil is given as:

$$q = 1000 * \sum MStage = 1000 * 0.103 = 103 \tag{4.2}$$

Equation 4.1 gives for $S_u = 20$, $q = 5.14 S_u = 5.14 * 20 = 102.8$. The result of the simulation gives a good approximation.

The soil field, in this particular case a constant field, is shown in Figure 4.7 and bearing capacity geometry, load and Plaxis 2D mesh is shown in Figure 4.8. The failure mechanism is illustrated in 4.9

Table 4.3: Soil parameters for anisotropic soil

Soil model	Mohr Coulomb - Undrained(C)		
Statistical Soil Parameters	Mean μ	Coefficient of Variation $CoV = \frac{\sigma}{\mu}$	Scale of fluctuation θ_x and θ_y
Unit weight, $\gamma_{sat} = \gamma unsat$	20 kN/m^3	0	-,-
Modulus of elasticity, E	10 MPa	0	-
Poissons ratio, ν	0.49	0	-
Undrained Shear Strength,S_u	20 kPa	0	0 and 0

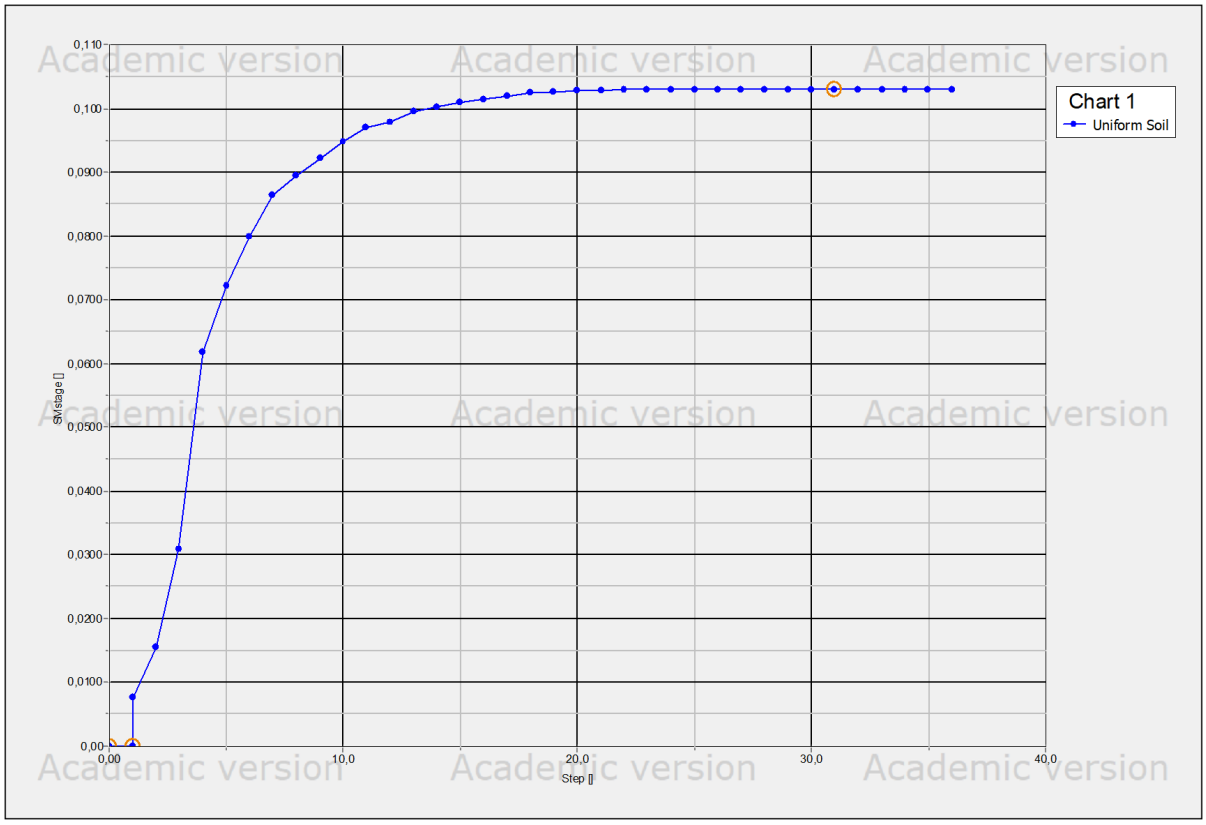


Figure 4.6: Bearing Capacity problem for uniform soil. Plaxis 2D Load step vs fraction of applied load. Failure occur at 0.103 times the applied load of 1000 kPa

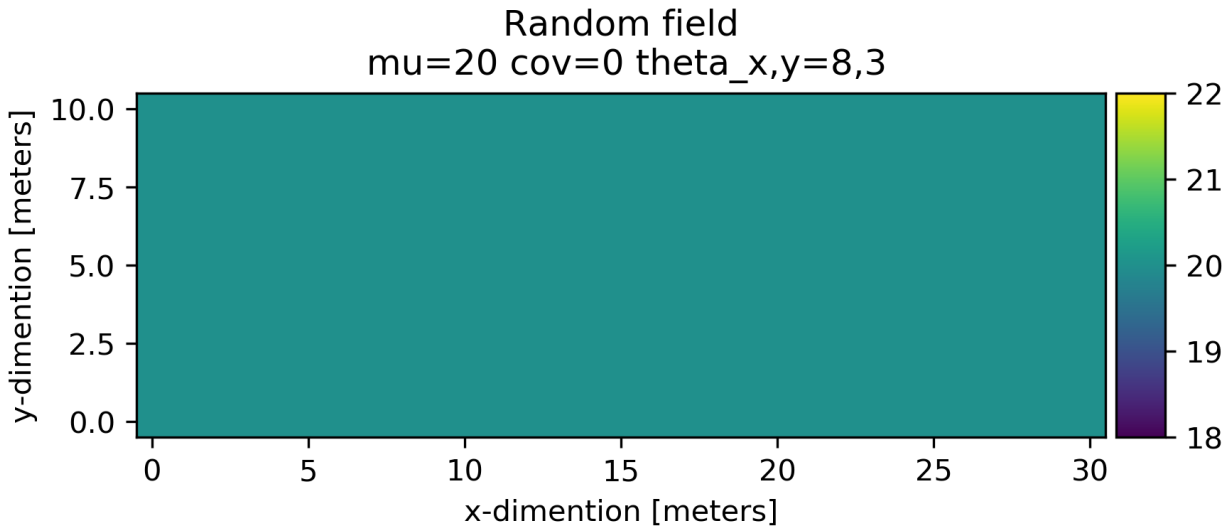


Figure 4.7: Uniform undrained soil shear strength of 20 kPa used in bearing capacity verification simulation

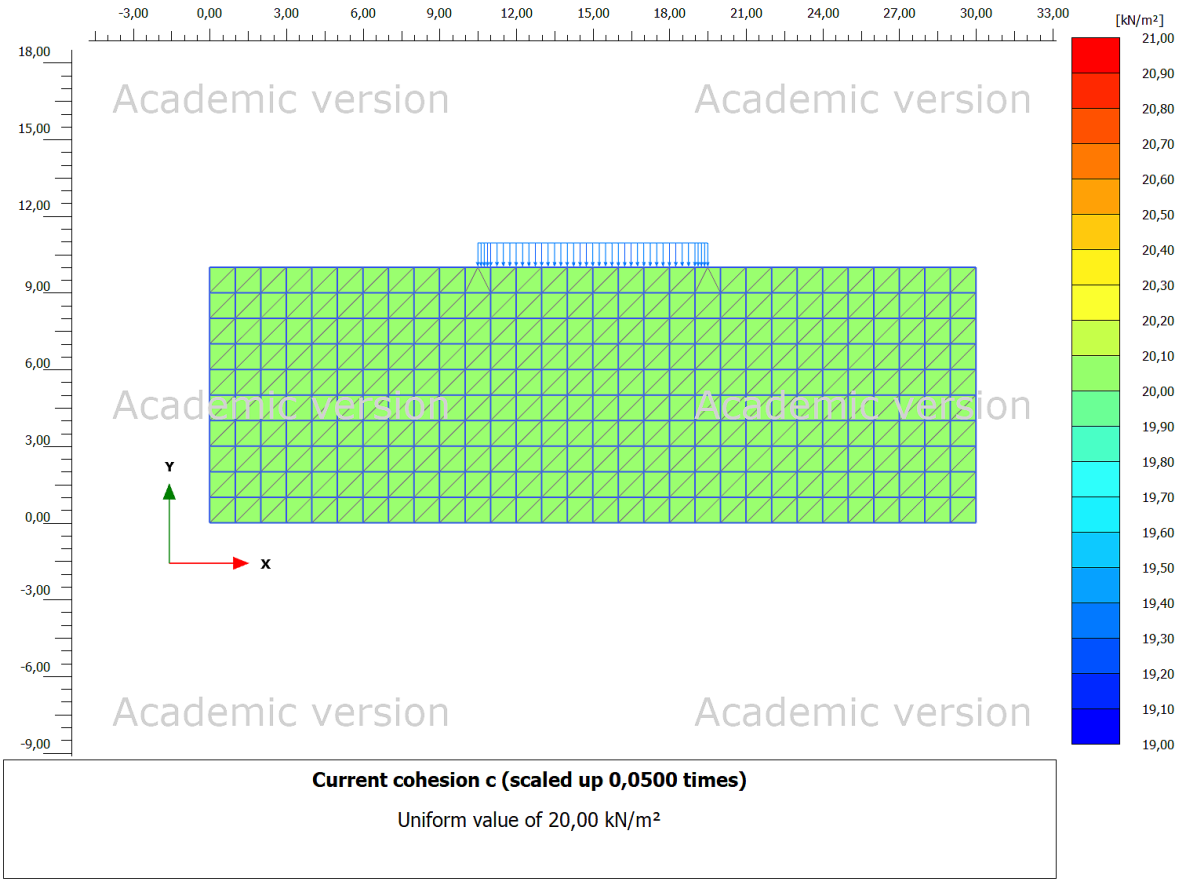


Figure 4.8: Bearing capacity problem geometry, load and FEM mesh.

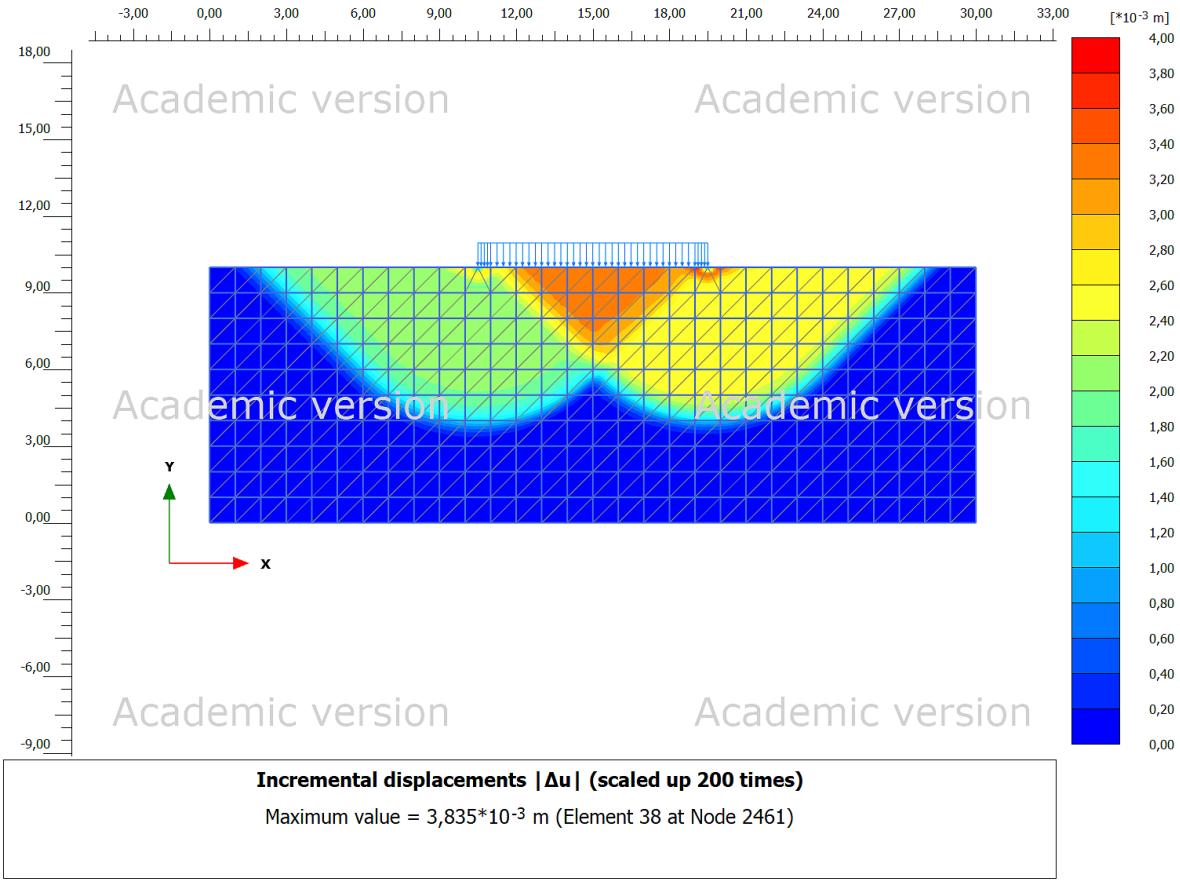


Figure 4.9: Illustration of failure mechanism for the bearing capacity verification simulation

4.4 Bearing Capacity Problem - Homogeneous anisotropic soil - Scale of Fluctuation 8 and 3 meters, CoV=0.1, Monte Carlo 250 realizations

The bearing capacity problem is repeated with a random field input representing the soil undrained strength.The soil parameters are given in table 4.4. 250 iterations are run in a Monte Carlo simulation. The FEM parametrisation is the same as for the other problems, repeated here: Elasto-plastic FEM simulation with Mohr-Coulomb material model using Plaxis undraind(C) behaviour. 15-Node triangular FEM elements are used.

Table 4.4: Soil parameters for anisotropic soil

Soil model	Mohr Coulomb - Undrained(C)		
Statistical Soil Parameters	Mean μ	Coefficient of Variation $CoV = \frac{\sigma}{\mu}$	Scale of fluctuation θ_x, θ_y
Unit weight, $\gamma_{sat} = \gamma unsat$	20 kN/m^3	0	-,-
Modulus of elasticity, E	10 MPa	0	-
Poissons ratio, ν	0.49	0	-
Undrained Shear Strength,S_u	20 kPa	0.3	8,3

The resulting bearing capacities from the 250 realizations of the random finite element run is displayed in a histogram in Figure 4.10. Note that all bearing capacities is lower than that for uniform soil with constant undrained shear strength equal to the mean S_u in the RFEM run.

Figure 4.11 shows three random realizations from the RFEM run and the corresponding variety in failure mechanism.

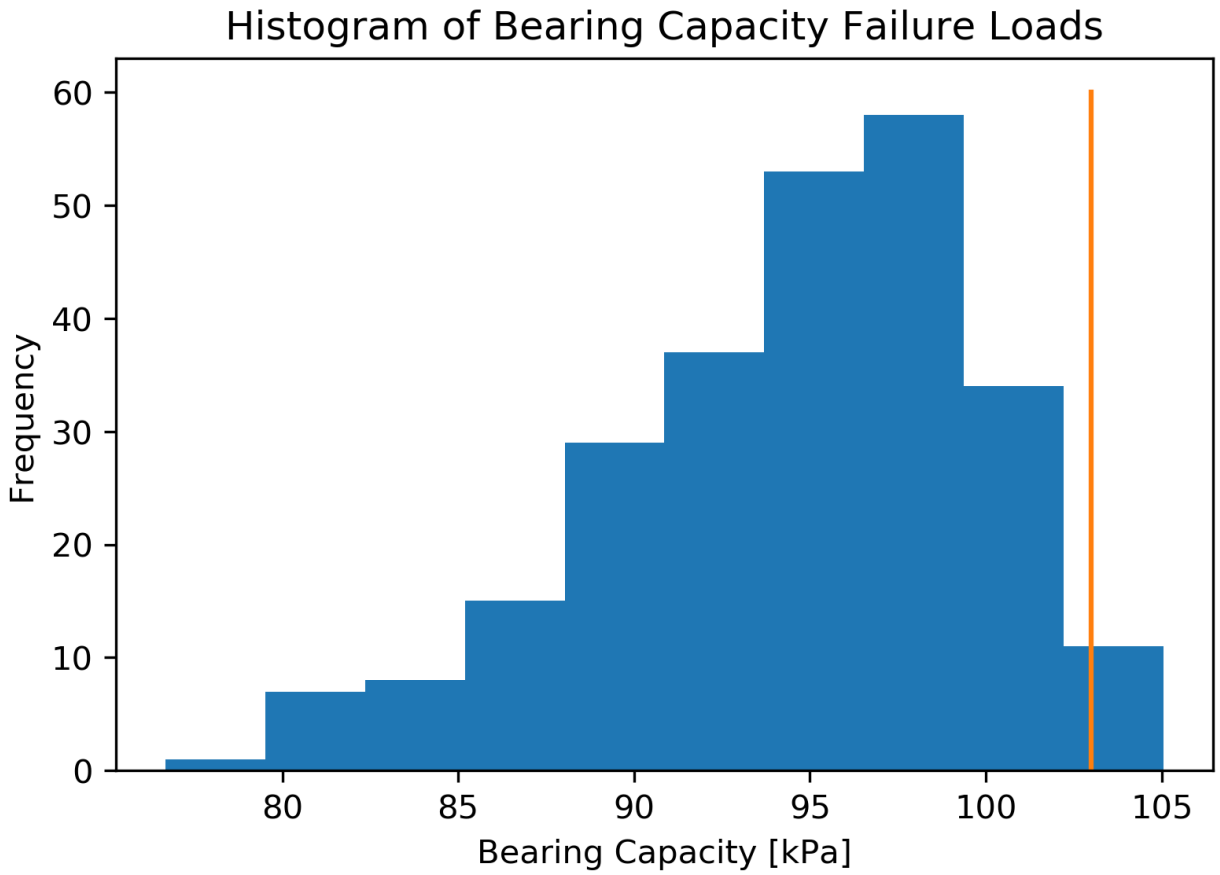
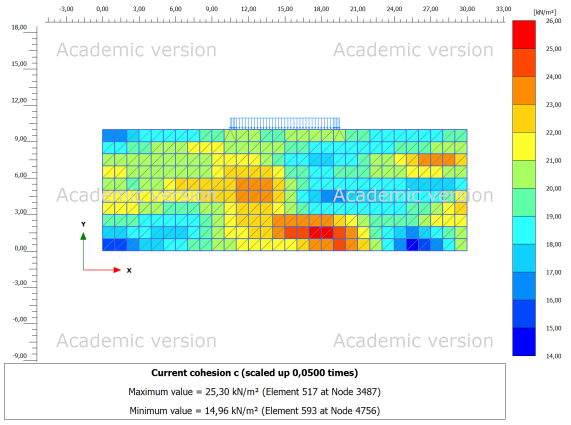
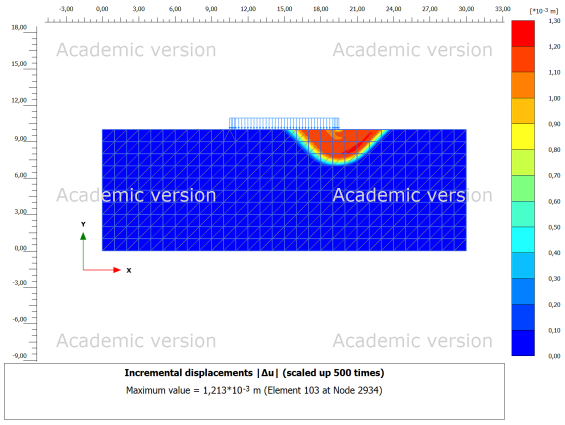


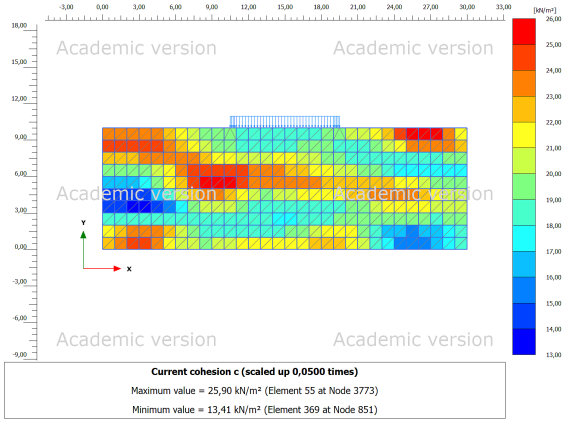
Figure 4.10: Histogram of Bering Capacities from 250 iterations of RFEM



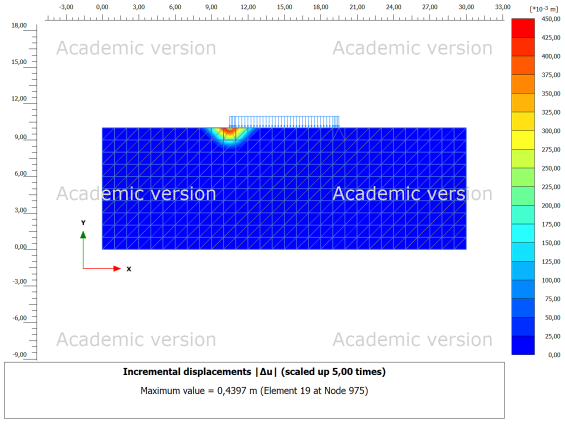
(a) Realization of undrained strength field, iteration 201



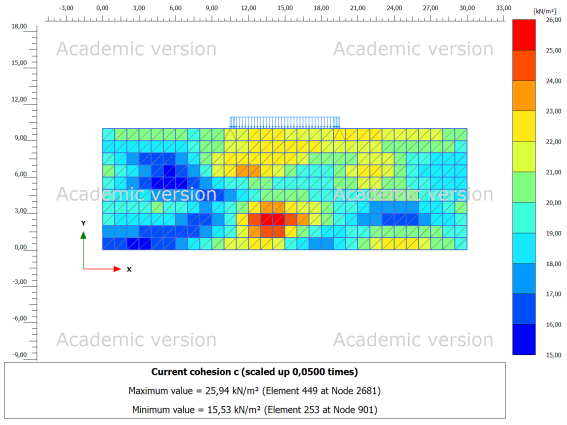
(b) Failure load 97.3 kPa, iteration 201



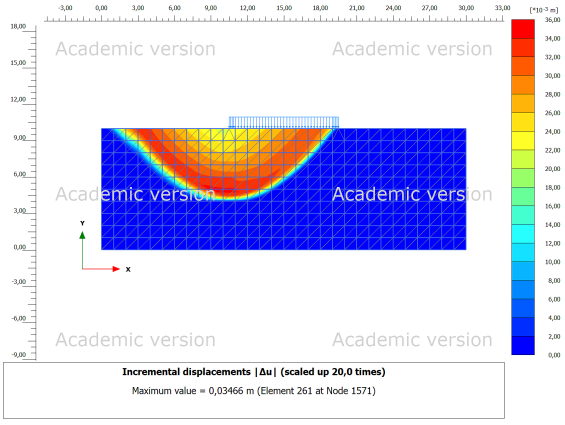
(c) Realization of undrained strength field, iteration 121



(d) Failure load 98.9 kPa, iteration 121



(e) Realization of undrained strength field, iteration 33



(f) Failure load 97.6 kPa, iteration 33

Figure 4.11: Three different realizations of random fields (Left) created by SRM, their FEM mesh and their corresponding failure mechanism (right) after incremental loading.

Chapter 5

Summary and Recommendations for Further Work

In this final chapter presents a summary of the results, a discussion of the findings and recommendations for further work.

5.1 Summary and Conclusions

In this project a method is implemented in Plaxis 2D using python API interface to run Random Finite Element Monte Carlo simulations on geotechnical problems with spatially variable soil models. The method is verified by controlling input parameters, fixing them to constrains giving known analytical solutions. The versatility of the implementation is demonstrated by verification on a slope stability and a bearing capacity problem.

Here, you present a brief summary of your work and list the main results you have got. You should give comments to each of the objectives in Chapter 1 and state whether or not you have met the objective. If you have not met the objective, you should explain why (e.g., data not available, too difficult).

This section is similar to the Summary and Conclusions in the beginning of your report, but more detailed—referring to the various sections in the report.

5.2 Discussion

Here, you may discuss your findings, their strengths and limitations.

The major limitation of the implementation in its current form is its execution time per realization. The python implementation of the Spectral Reference Method for generating random fields is much slower than the Matlab implementation it was adapted from. The differences of the inner workings of Matlab and python is beyond the scope of this project, but it is suggested in further work to improve this. The processing speed improvement is needed for Monte Carlo simulations of rear events which demands many simulations to gather the statistics needed for describing small probabilities with confidence.

5.3 Recommendations for Further Work

Recommendation for further work is given below. The focus of the list below is in improvement of the method implemented in this project work. Though the application of the method may be the most interesting...

The recommendations are classified by time as:

- Short-term further work
  - Stress testing by extending problem size, smaller earth elements and denser mesh. What is the limit of the program? And what size of problems can practically be analyzed?
  - Performance optimization. The Matlab implementation of the SRM runs many orders of magnitude faster than the python implementation. It should be possible to narrow this gap, making the implementation much more efficient, allowing many more iterations of RFEM to be run in the same time
- Medium-term further work
  - More advanced soil models with more random input parameters and correlated random fields. Is it possible to simulate strength softening or sensitivity?
  - More geometries and geotechnical problems like earth pressure problems. The implementation proposed can easily be extended with user input like excavation stages in

a staged construction. Also possible to add and study forces on anchors, plates, sheet piles etc.

- Long-term further work
  - Performance optimization, parallelization. RFEM is a highly parallel process that can benefit by computing realizations in parallel for big performance gains and lower execution time. It is not believed to be straight forward to implement this, but maybe multiple instances of Plaxis can be started and run on different compute nodes.

Appendix A

Acronyms

FEM Finite Element Method

RFEM Random Finite Element Method

SRM Spectral Representation Method

CoV Coefficient of Variation

Appendix B

Source code

This appendix contains the source code to run the RFEM Plaxis imlementation.

B.1 Introduction

```
import random
import numpy as np

def get_RGB_number(R, G, B):
    # get colour number from RGB using BIT LEFT SHIFT
    iB = B<<16 # left shift 16 bits for Blue
    iG = G<<8  # left shift  8 bits for Green
    iR = R     # left shift  0 bits for Red
    return iB + iG + iR

def normaliser(x,xmin,xmax):
    if xmax-xmin==0:
        return x
    return (x-xmin)/(xmax-xmin)

def create_geometry(s_i, g_i):
    """
    Takes the plaxis variables s_i, g_i and performs a series of operations
    to create a soil layer, assign a test material to soil, create a line load
    with dynamic multiplier
    """
    s_i.new()
    g_i.SoilContour.initializerectangular(0, 0, 10, 10)


```

```
26
27
28      #g_i.borehole(3)
29      #g_i.Soillayer_1.Zones[0].Bottom = -3
30      nx1=15
31      nx2=15
32      ny1=5
33      ny2=5
34      n=nx1+nx2
35
36      #topo=[10,10,10,10,10,10,10,10,10,10,9.5,9,8.5,8,7.5,7,6.5,6,5.5,5,5,5,5,5,5,5,5,5,5]
37      #interpoler topografi
38
39      H=5 #Hoeyde slope
40      D=2 #dybde
41      topoy = [D*H,D*H,D*H-H,D*H-H]
42      topox = [0,2*H,4*H,6*H]
43      #topo = np.interp(np.linspace(0,30,30),topox,topoy)
44      topo = np.interp(np.arange(31),topox,topoy)
45
46      print(topo)
47
48
49      m=ny1+ny2
50      matnum=0
51      tttrans=srm() #generate random field
52
53      sumin=np.min(tttrans)
54
55      sumax=np.max(tttrans)
56
57      for i in range(n):
58          for j in range(m):
59              if(j+1>=topo[i]):
60
61                  #g_i.polygon((0+i,0+j),(0+i,topo[i]),(1+i,topo[i+1]),(1+i,0+j))
62                  g_i.polygon((0+i,0+j),(0+i,topo[i]),(1+i,topo[i+1]),(1+i,0+j))
63
64
65
66
67
68
69                  su=tttrans[j,i]
70                  farge=int(normaliser(su,sumin,sumax)*255)
```

```
                material = g_i.soilmat("MaterialName", "Test"+str(i)+str(j), "SoilModel", "Mohr-Coulomb",
                                       "gammaUnsat", 20, "gammaSat", 20, "DrainageType","Undrained (C)",
                                        "Eref",10e3, "nu",0.49,  "cref",su,
                                          "Colour",get_RGB_number(farge, farge, farge)) #nu=poisson-ratio

                g_i.Soils[matnum].Material = material ########virker ikke saa bra med n!=m
                matnum += 1
                break


            g_i.polygon((0+i,0+j),(0+i,1+j),(1+i,1+j),(1+i,0+j))
#             material = g_i.soilmat("MaterialName", "Test"+str(i), "SoilModel", 1,
#                                    "gammaUnsat", 17, "gammaSat", 20, "Gref", 2000)

            su=tttrans[j,i]
            farge=int(normaliser(su,sumin,sumax)*255)
            material = g_i.soilmat("MaterialName", "Test"+str(i)+str(j), "SoilModel", "Mohr-Coulomb",
                                   "gammaUnsat", 20, "gammaSat", 20, "DrainageType","Undrained (C)",
                                    "Eref",10e3, "nu",0.49,  "cref",su,
                                      "Colour",get_RGB_number(farge, farge, farge)) #nu=poisson-ratio

            g_i.Soils[matnum].Material = material ########virker ikke saa bra med n!=m
            matnum += 1

    g_i.gotostructures()
    #g_i.lineload((2.5, 0+m), (7.0, 0+m))
    #lineload_g = g_i.LineLoads[-1]
    #lineload_g.qy_start = -100


def simple_test_case(s_i, g_i):
    """
    Takes the plaxis variables s_i, g_i and performs a series of operations
    to create geometry features and generate the mesh
    """
    create_geometry(s_i, g_i)

    g_i.gotomesh()
    g_i.mesh(0.1)

```

```
    output_port = g_i.viewmesh()
    s, g = new_server('localhost', port=output_port, password=s_i.connection._password)

    return s, g
```

Listing B.1: SRM code

```
from plxscripting.easy import *
import numpy as np
import matplotlib.pyplot as plt
from math import atan2


slopeStatus = np.zeros(1)

for n in range(len(slopeStatus)):

    s_o, g_o = simple_test_case(s_i, g_i)


    # Changes the mode and define phases
    g_i.gotostages()
    phase0_s = g_i.InitialPhase
    phase1_s = g_i.phase(phase0_s)
    phase1_s.Identification = 'phase1_iteration' + str(n)

    # Activate all geometry
    g_i.Geometry.activate(phase1_s)

    g_i.calculate()
    g_i.view(phase1_s)
    print(phase1_s.Reached.SumMstage)
    slopeStatus[n]=float(str(phase1_s.Reached.SumMstage))
    #f=phase1_s.Reached.SumMstage
    #slopeStatus = np.append(slopeStatus,f)
    newest_plot = g_o.Plots[-1]
    newest_plot.ResultType = g_o.ResultTypes.Soil.C
    newest_plot.PlotType = 'shadings'
    newest_plot.Phase = g_o.Phases[-1]
    newest_plot.DrawFrame = False
    #image_wrapper is an object that can save the created
    #image or, if Pillow is installed, you can get the internal
    #Pillow.Image object and use that.
    image_wrapper = newest_plot.export(1600, 1200)
    try:
        from PIL import ImageFilter
```

```
        pil_image = image_wrapper.image
        new_image = pil_image.filter(ImageFilter.DETAIL)
        new_image.save("C:\\Users\\olewei\\Desktop\\testp"+str(time.strftime("%Y%m%d-%H%M%S"
    ))+".png")
    except ImportError:
        #Just save if we don't have Pillow
        image_wrapper.save("C:\\Users\\olewei\\Desktop\\testp"+str(time.strftime("%Y%m%d-%H%
    M%S"))+".png")
print(slopeStatus)
```

Listing B.2: SRM code

```
for it in range(145):
    phase2_s = g_i.phase(phase0_s)
    phase2_s.Identification = 'phase1_iteration' + str(n+len(slopeStatus))



    nx1=15
    nx2=15
    ny1=5
    ny2=5
    n=nx1+nx2

    #topo=[10,10,10,10,10,10,10,10,10,10,9.5,9,8.5,8,7.5,7,6.5,6,5.5,5,5,5,5,5,5,5,5,5,5]
    #interpoler topografi

    H=5 #Hoeyde slope
    D=2 #dybde
    topoy = [D*H,D*H,D*H-H,D*H-H]
    topox = [0,2*H,4*H,6*H]
    #topo = np.interp(np.linspace(0,30,30),topox,topoy)
    topo = np.interp(np.arange(31),topox,topoy)

    print(topo)

    m=ny1+ny2
    matnum=0
    tttrans=srm() #generate random field
    sumin=np.min(tttrans)

    sumax=np.max(tttrans)
    g_i.gotosoil()

    for i in range(n):
            for j in range(m):
                if(j+1>=topo[i]):
```

```
                    #g_i.polygon((0+i,0+j),(0+i,topo[i]),(1+i,topo[i+1]),(1+i,0+j))
                    #g_i.polygon((0+i,0+j),(0+i,topo[i]),(1+i,topo[i+1]),(1+i,0+j))









                    su=tttrans[j,i]
                    #su = 100
                    farge=int(normaliser(su,sumin,sumax)*255)

                    g_i.Soils[matnum].Material.cref = su
                    #g_i.Soils[matnum].Material.Colour = get_RGB_number(farge, farge, farge)


                    matnum += 1
                    break


#              g_i.polygon((0+i,0+j),(0+i,1+j),(1+i,1+j),(1+i,0+j))
#              material = g_i.soilmat("MaterialName", "Test"+str(i), "SoilModel", 1,
#                                     "gammaUnsat", 17, "gammaSat", 20, "Gref", 2000)

                su=tttrans[j,i]
                #su = 100
                farge=int(normaliser(su,sumin,sumax)*255)

                g_i.Soils[matnum].Material.cref = su
                #g_i.Soils[matnum].Material.Colour = get_RGB_number(farge, farge, farge)
                matnum += 1

g_i.gotostages()
# Activate all geometry
g_i.Geometry.activate(phase2_s)



g_i.calculate()
#g_i.view(phase2_s)

slopeStatus = np.append(slopeStatus,float(str(phase2_s.Reached.SumMstage)))
print(slopeStatus)
```

```
79      print(it)
```

Listing B.3: SRM code

```
1  ppfail = np.array([])
2  for n,s in enumerate(slopeStatus):
3      ss=slopeStatus[0:n+1]
4      #print(ss)
5      nonfail = np.count_nonzero(ss > .99)
6      print("number of non failures ",nonfail," of ",len(ss)," iterations")
7      pfail = (len(ss)-nonfail)/len(ss)
8      print("probability of failure",pfail)
9      ppfail = np.append(ppfail,pfail)
10 plt.figure()
11 plt.title('Monte Carlo simulations vs probability of failure')
12 plt.xlabel('Iteration number')
13 plt.ylabel('Probability of  failure')
14 plt.plot(ppfail)
15 plt.savefig("C:\\Users\\olewei\\Desktop\\pfailit"+str(time.strftime("%Y%m%d-%H%M%S"))+".png"
       , dpi=300, bbox_inches='tight')
16 plt.show()
17 np.savetxt("C:\\Users\\olewei\\Desktop\\slopestatus.txt",slopeStatus)
```

Listing B.4: SRM code

```
1  print(slopeStatus)
2  nonfail = np.count_nonzero(slopeStatus > .99)
3  print("number of non failures ",nonfail," of ",len(slopeStatus)," iterations")
4  pfail = (len(slopeStatus)-nonfail)/len(slopeStatus)
5  print("probability of failure",pfail)
```

Listing B.5: SRM code

B.1.1 Starting Python scripting interface - Jupyer notebook

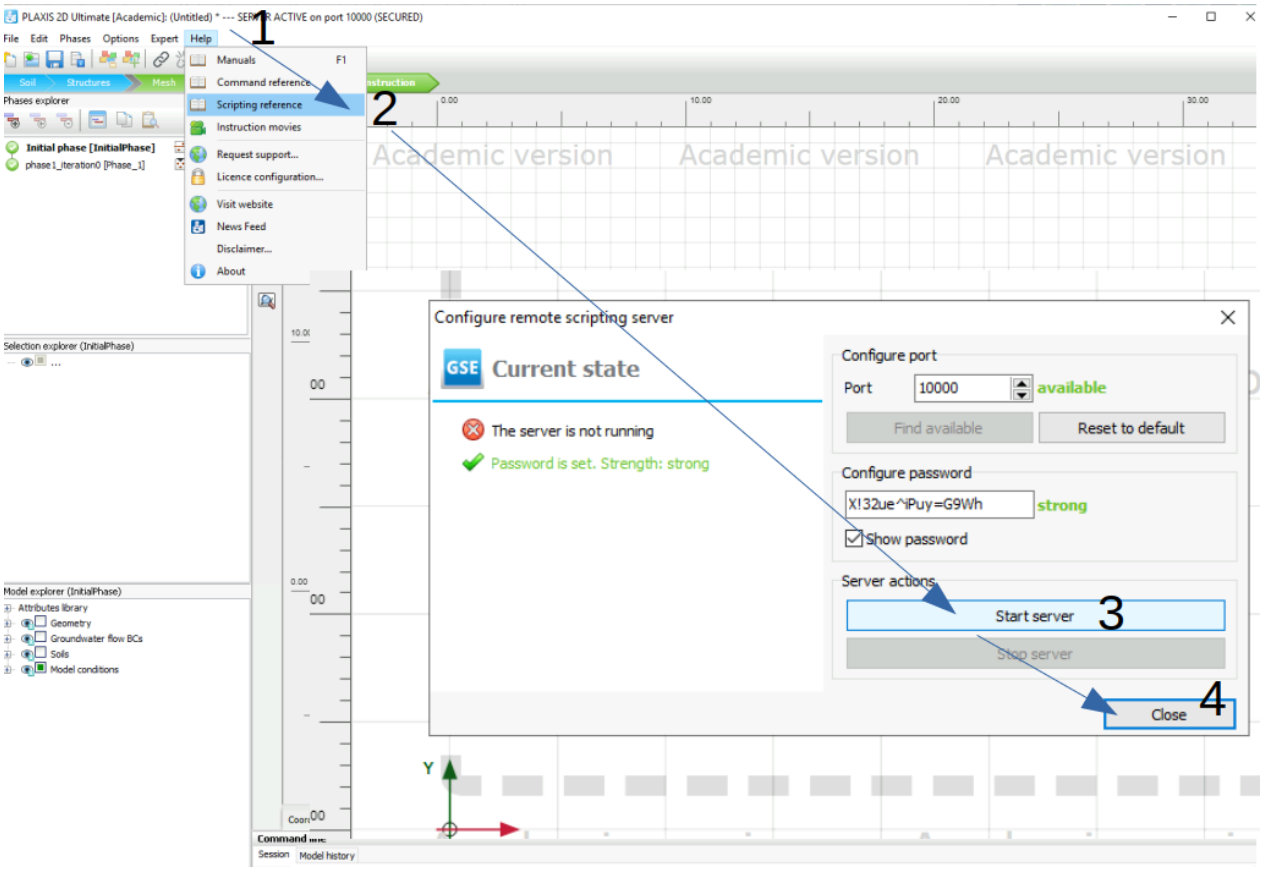

1
2
3
4
Help
Manuals
Command reference
Scripting reference
Instruction movies
Request support...
Licence configuration...
Visit website
News Feed
Disclaimer...
About
Academic version
Configure remote scripting server
Current state
The server is not running
Password is set. Strength: strong
Configure port
Port
10000
available
Find available
Reset to default
Configure password
X!32ue^iPuy=G9Wh
strong
Show password
Server actions
Start server
Stop server
Close

Figure B.1

Bibliography

R. Brinkgreve, W. Swolfs, E. Engin, D. Waterman, A. Chesaru, P. Bonnier, and V. Galavi. Plaxis 2d 2010. *User manual, Plaxis bv*, 2010.

G. Deodatis, M. Shinozuka, and A. Papageorgiou. Stochastic wave representation of seismic ground motion. ii: Simulation. *Journal of engineering mechanics*, 116(11):2381–2399, 1990.

G. A. Fenton, D. V. Griffiths, et al. *Risk assessment in geotechnical engineering*, volume 461. John Wiley & Sons New York, 2008.

D. Griffiths and G. A. Fenton. Seepage beneath water retaining structures founded on spatially random soil. *Geotechnique*, 43(4):577–587, 1993.

D. Griffiths and P. Lane. Slope stability analysis by finite elements. *Geotechnique*, 49(3):387–403, 1999.

N. Janbu. Slope stability computation. *Soil Mechanics and Foundation Engineering Report*, 1968.

K. Krabbenhøft, A. Lymain, and J. Krabbenhøft. Optum g2 2016-user manual, 2016.

S. Nordal. Tba4116 geotechnical engineering advanced course. 2020.

M. Shinozuka and G. Deodatis. Simulation of multi-dimensional gaussian stochastic fields by spectral representation. 1996.

I. M. Smith, D. V. Griffiths, and L. Margetts. *Programming the finite element method*. John Wiley & Sons, 2013.

E. Vanmarcke. *Random fields: analysis and synthesis*. World scientific, 2010.

E. Vanmarcke and M. Grigoriu. Stochastic finite element analysis of simple beams. *Journal of engineering mechanics*, 109(5):1203–1214, 1983.