Probabilistic classification of soils based on Local Average Subdivision method and CPT data

Additional thesis (10 ECTs)

Sijun Zeng

MSc in Geo-Engineering

5561213

Supervisors:

Dr. Divya Varkey

Dr. ir. Bram van den Eijnden

Prof. Dr. Michael Hicks

October 7, 2022

Contents

Abstract

A random field generator based on Local Average Subdivision (LAS) method is proposed in this study in order to achieve probabilistic soil classification and quantify the uncertainty of the generated most probable geological cross section. CPT data and Robertson's soil classification chart (1990) are adopted to classify the soil. The sole application of LAS makes the random field unconditional, which has been improved to conditional random field generator by using Kriging interpolation. Both unconditional and conditional generator are tested in an illustrative example and the results indicate that the improvement from unconditional to conditional random field reduces the uncertainty of the most probable result of classifications and the classifications in the unconditional random field will converge if there are enough realizations. Additionally, the conditional random field generator is further applied in a case with three conducted CPTs, which build up a domain with very large scale of fluctuations. It's found that the uncertainty of the generated most probable result of classifications is pretty low so it's speculated that the proposed generator can be best applied in a large scale of fluctuation scenario. Another finding in the case study is that the proposed random field generator can be used to verify the reliability of conducted CPTs.

1. Introduction

1.1 General background

The subsurface soil stratigraphy, which constitutes a part of site characterization, plays an important role in the construction works and geotechnical structures (Wang, Wang & Liang, 2017; Clayton, 2001). As for how to develop a underground geological cross-section, traditionally people simply draw straight lines to connect the boundaries of the same soil types between two adjacent site investigation (e.g. CPT, borehole, etc.). This traditional method has been used for a long time but it can only be applied when site has relatively simple geology or when extensive ground investigations are available (Shi & Wang, 2021) because it actually ignores the possible inconsistency of the soil boundaries and reduces the uncertainty of soil classifications arbitrarily. The application of such a method in real engineering cases may lead to many problems. So how to obtain a geological cross section approaching to the real in-situ condition under the circumstance that the site investigations are sparse and the underground geological situation is complicated becomes the research question for this project.

The solution, also the research objectivity of this project, to the aforementioned problem is to carry on probabilistic soil classifications and find the geological cross section in a stochastic way. This method offers a great opportunity to capture the soil inherent variability by random field and infer the most probable result of classifications, especially the classifications at unsampled locations. Moreover the uncertainty can be quantified, which means that the uncertainty of the most probable result can also be obtained. In all, in this way the geological cross section can be inferred in a stochastic way instead of a deterministic way.

Many probabilistic approaches have been developed such as random field and machine learning methods (Shi & Wang, 2021a; Shi & Wang, 2021b; Hu & Wang, 2020; Chessa, 2006). Specifically in this study, how to achieve the probabilistic analysis has been divided into two parts: the soil classifications based on CPT data and the random field generator based on Local Average Subdivision method and Kriging interpolation.

1.2 Scope of this study

In order to deal with stochastic soil stratigraphy, there are two key parts: soil classification model and random field generator. In this study, for the former one, Robertson's CPT classification method (Robertson, 1990) is adopted and it will elaborated in Section 3.1. For the latter one, conditional random filed generator based on Local Average Subdivision and Kriging Interpolation is adopted, which will be elaborated in Section 3.2.1 (LAS) and Section 3.2.2 (Kriging interpolation). The most probable results for classifications and their uncertainties, as well as some other post-analysis will be elaborated in Chapter 4 and Chapter 5, respectively for an illustrative example and a real case.

There are many other methods to achieve soil classification besides CPT classification and there are many other methods to generate random field no matter it is stationary or non-stationary. This study can function as a contrast example for researchers who want to investigate in and find out the most accurate and convenient way to achieve probabilistic soil classifications.

2. Literature Review

2.1 Cone Penetration Test

The site investigation method used on this study is Cone Penetration Test (CPT). The test method consists of pushing an instrumented cone, with the tip facing down, into the ground at a controlled rate (controlled between 1.5 -2.5 cm/s accepted) ("Cone penetration test - Wikipedia", 2022). It is a widely used in-situ test to identify soil types. There are a couple of advantages of such a test: providing accurate and continuous profiles (Ramsey, 2010), being fast, repeatable and economical (Robertson, 2010); the data obtained from CPT being able to interpret soil properties based on some empirical relations (e.g. shear wave velocity, undrained shear strength, particle size and hydraulic conductivity; McGann, Bradley, Taylor, Wotherspoon & Cubrinovski, 2015; Anagnostopoulos, Koukis, Sabatakakis, & Tsiambaos, 2003; Tillmann et al., 2008)**.** The disadvantage exists in terms of only being applicable for shallow subsurface, not considering cyclic loading conditions, etc. (Ramsey, 2010). What is obtained from CPT and how to use specifically in this study will be elaborated in Section 3.1.

2.2 Soil Classification Method

Soil classification has been important in geotechnical engineering field for a long time and there are many methods well developed. In the following there are some classical soil classification methods chosen to display: American Association of State Highway and Transportation Officials (AASHTO) soil classification system which applies sieve analysis, plastic limit and liquid limit to classify; Unified Soil Classification System (USCS) which applies texture and grain size to classify; The United States Department of Agriculture (USDA) method textural soil classification method which also applies soil textures to classify; soil behavior type (Robertson, 1990) and soil behavior type index Ic (Robertson & Wride, 1998) which applies normalized cone resistance and friction ratio to classify.

In order to obtain necessary parameters for those classification methods, there are various in-situ tests and experimental tests developed. For AASHTO soil classification, sieve analysis and Atterberg Limit Test are needed. For soil behavior type and soil behavior type index Ic, CPT or flat plate dilatometer test (Robertson, 2015) are needed. In addition, some methods only require people to identify visually, like UCSC and USDA method, needing people to classify the soils by observing soils' texture.

2.3 Random field

Random field originates from mathematics and physics and it has a wide range of applications. Generally it can be defined as the representation of the joint probability distribution for a set of random variables (Hernández-Lemus, 2021). In this study, the random field can be simply understood as a domain where there are random variables generated in each element for each realization. One realization means the generation of a random property field and subsequent analysis of the problem (Hicks, 2009).

As for how to generate random field, there are various ways. In general it can be divided into two types: stationary (Gaussian) random field, which is commonly used and non-stationary (non-Gaussian) random field. For stationary method, there are Fast Fourier Method, Turning Bands Method, Spectral method, Matrix decomposition method, Karhunen-Loeve expansion, Moving average, Sequential simulation and Local Average Subdivision method (Liu, Li, Sun & Yu, 2019). Compared to other methods, the Local Average Subdivision, which is adopted in this study, has the advantage that it is straightforward and fast (Fenton & Griffiths, 2007). Detailed information about LAS will be shown in Section 3.2.1. In non-stationary random field, the statistics of data points (e.g. mean, standard deviation) change in temporal dimension or in space dimension. This may be an advantage, especially when analyzing CPT data because CPT data is strictly non-stationary (Jamshidi & Kamyab, 2015). It has an inherent depth trend. In non-stationary random field all the data doesn't have to be detrended which conforms with reality and avoid potential inaccuracies. Nowadays, there are a large number of investigations in non-stationary random field (Griffiths, Huang & Fenton, 2015; Montoya-Noguera, Zhao, Hu, Wang, & Phoon, 2019; Hu & Wang, 2020), which shows the potential to become a trending topic.

There is an overview of a non-stationary generator, Bayesian compressive sampling Karhunen-Loève (BCS-KL) generator, proposed by Hu & Wang (2020), which may be helpful for researchers who are interested in it. This generator is based on a novel sampling concept called compressive sampling/sensing (CS): A certain signal or image (like the qt and Fr which are investigated in this study) can be reconstructed from sparse measurements on that signal or image. The reconstruction is based on the fact that many natural signals are compressible. In other words, a signal/image can be estimated by a limited number of weight coefficients multiplied by respective basis functions like wavelet functions and cosine functions. Specifically it is achieved by the following equations:

$$F = \sum_{t=1}^{N_{X1} \times N_{X2}} B_t^{2D} w_t^{2D} \tag{1}$$

$$Y = \psi_{x1} F \psi_{x2} \tag{2}$$

where F is the 2D signal/image to be reconstructed, B_t^{2D} is the t-th 2D basis function; w_t^{2D} is the weight coefficients corresponding to B_t^{2D}; Y is a sub-matrix of F constituted by spare measurements. In this case, Y can be simply understood as a 2D matrix which consists of 1D conducted CPTs. Detailed mathematical derivation can be viewed in Hu and Wang's paper. One highlight can be seen in those equations that this generator bypasses the difficulty to connect the conducted CPT to the input parameters used in the random field. The measurements are directly used as input for reconstruction. How to solve such a difficulty in this study is shown in Section 3.3 but no matter how subtle the solution is, it is not as direct as the BCS-KL generator and more procedures mean more inaccuracies. So that's the most valuable point worth researching. In addition, the 2D matrix F can automatically remain the anisotropic/isotropic trait.

3. Methodology

3.1 Soil Classification method based on CPT

In this study, Robertson's CPT classification chart (1990) is adopted. There are some empirical relations between the soil behavior types and normalized friction ratio (Fr) and normalized cone resistance (Qt), which can be seen in Fig. 1. The number 1 to 9 in Fig. 1 refers to the soil behavior type, and Fig. 2 illustrates detailed descriptions. It's worthy noticing that Fr and Qt can not be directly measured from CPT but can be indirectly calculated by some parameters measured by CPT. The relationship is shown below.

$$Q_t = \frac{q_t - \sigma_{v0}}{\sigma'_{v0}} \tag{3}$$

$$F_r = \frac{f_s}{q_t - \sigma_{v0}} \times 100 \tag{4}$$

Where,

$q_t = q_c + u_2 \times (1-a)$ is the corrected tip resistance (MPa) (Campanella, Gillespie & Robertson, 1982), q_c is the tip resistance measured from CPT (MPa), u_2 is the measured pore pressure behind the cone tip, $a = \frac{A_N}{A_T}$ is the net area ratio, A_N and A_T are respectively load transfer area behind the cone tip and cross-sectional area at the base of cone tip, σ_{v0} is the vertical total stress (MPa), $\sigma'_{v0} = \sigma_{v0} - u_0$ is the vertical effective stress(MPa), u_0 is the in-situ pore water pressure (MPa) and f_s is the sleeve friction ratio (MPa).

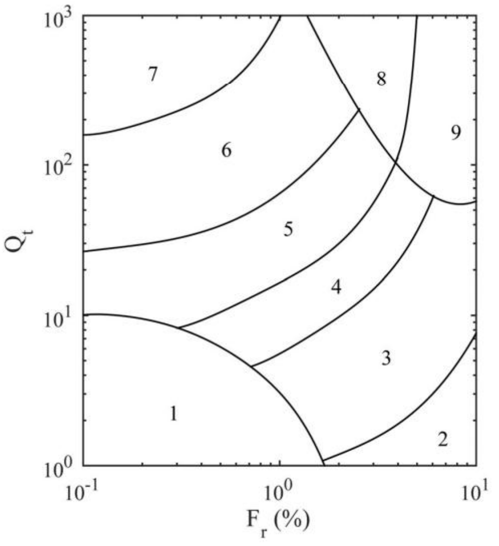
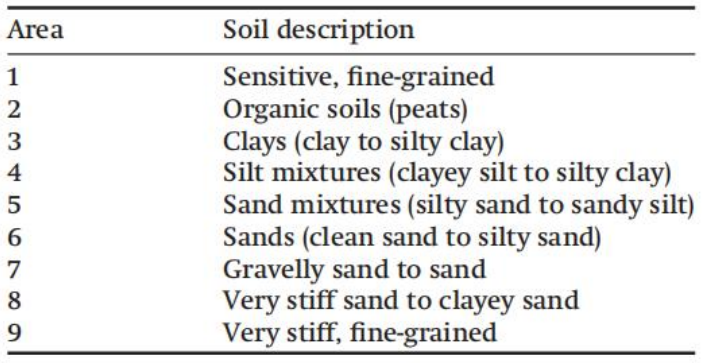


Fig. 1. Robertson's soil classification chart based on CPT with normalized friction ratio and cone resistance (1990)



Area	Soil description
1	Sensitive, fine-grained
2	Organic soils (peats)
3	Clays (clay to silty clay)
4	Silt mixtures (clayey silt to silty clay)
5	Sand mixtures (silty sand to sandy silt)
6	Sands (clean sand to silty sand)
7	Gravelly sand to sand
8	Very stiff sand to clayey sand
9	Very stiff, fine-grained

Fig. 2. Soil behaviour type description based on Robertson's chart (1990)

In Netherlands, there is a website (https://www.dinoloket.nl/en) which offers open-source CPT profiles with a .GEF format and in Chapter 5, for the real case study, three CPTs are acquired from the website. Those .GEF files can be read by a CPT processing script built up by Guido de Zeeuw and it will classify the CPT data automatically based on the aforementioned chart. This script can be divided into two part: reading CPT and classifying based on Robertson's chart. The classifying part is extracted and used individually in Chapter 4 because there are no real CPTs to be read. The exemplary processing result by using this script is shown in Fig. 3.

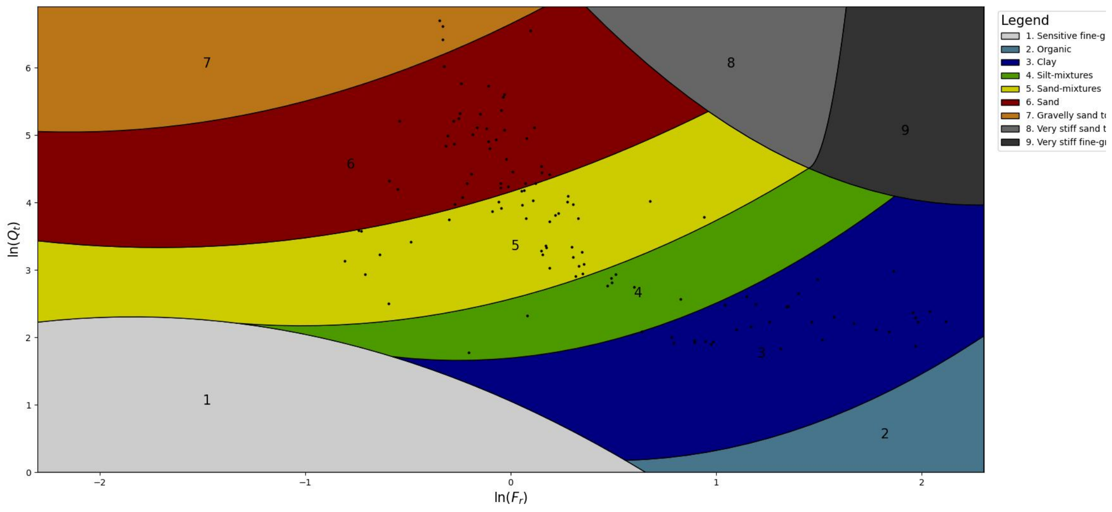


Fig. 3. An example to the result of CPT processing script

As it can be seen, the result of this script is in logarithmic scale. So actually Fig. 1 is used in the way as shown in Fig. 4.

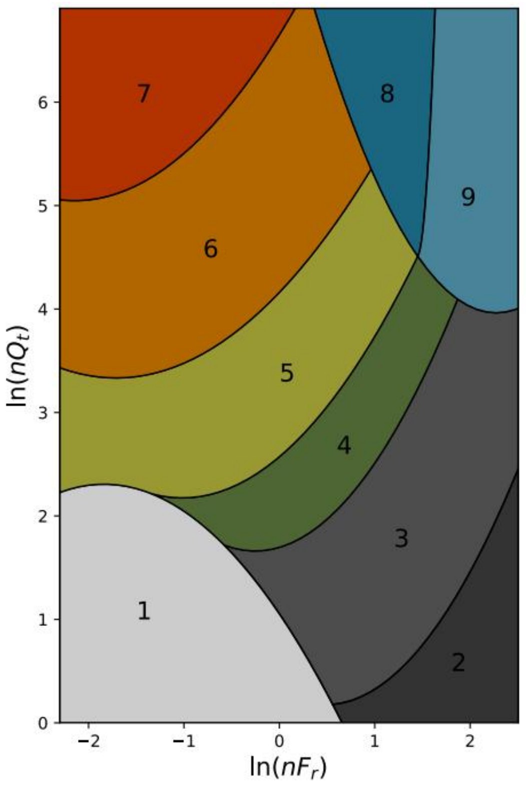


Fig. 4. Robertson's soil classification chart based on CPT with logarithmic normalized friction ratio and cone resistance (1990)

3.2 Conditional random field generator based on LAS and Kriging

3.2.1 Random field generator based on Local Average Subdivision

The random field generator used in this study is based on Local Average Subdivision (LAS) method, which was first proposed by Fenton (1990). The procedures of this method can be conceptually viewed in Fig. 5. From the stage 0 to stage 3, this is the process to subdivide the random field, which will continue until the needed maximal subdivisions. As for how to calculate the values in each cell, for example, in stage 3, the values of B1, B2 and B3 are determined by a random number generator and parent cell values. For the former one, it is built up based on the mean (μ), standard deviation (σ), scale of fluctuation (θ) as well as anisotropy of heterogeneity (ξ, the ratio between the horizontal and vertical scale of fluctuation) of the investigated material property. For the latter one, not only the cell P, but also the cells surrounding P are parent cells. Then for the value of A, upward averaging is taken, which means (A + B1 + B2+ B3)/4 = P. Pay attention the initial global mean (Z10) here is NOT equivalent to the mean of the system. Instead it will be determined by the scale of fluctuation and the problem domain, which won't be elaborated in this study. Additionally, if the field is anisotropic (the scale of fluctuation is different in horizontal and vertical direction), it will be squashed as it can be seen in the right part of Fig. 5. Taking ξ = 2 for example, the vertical direction size is reduced by a factor of 2, later the cell size will go back to the original size by taking average of the two small cells. This is the basic method used in this research.

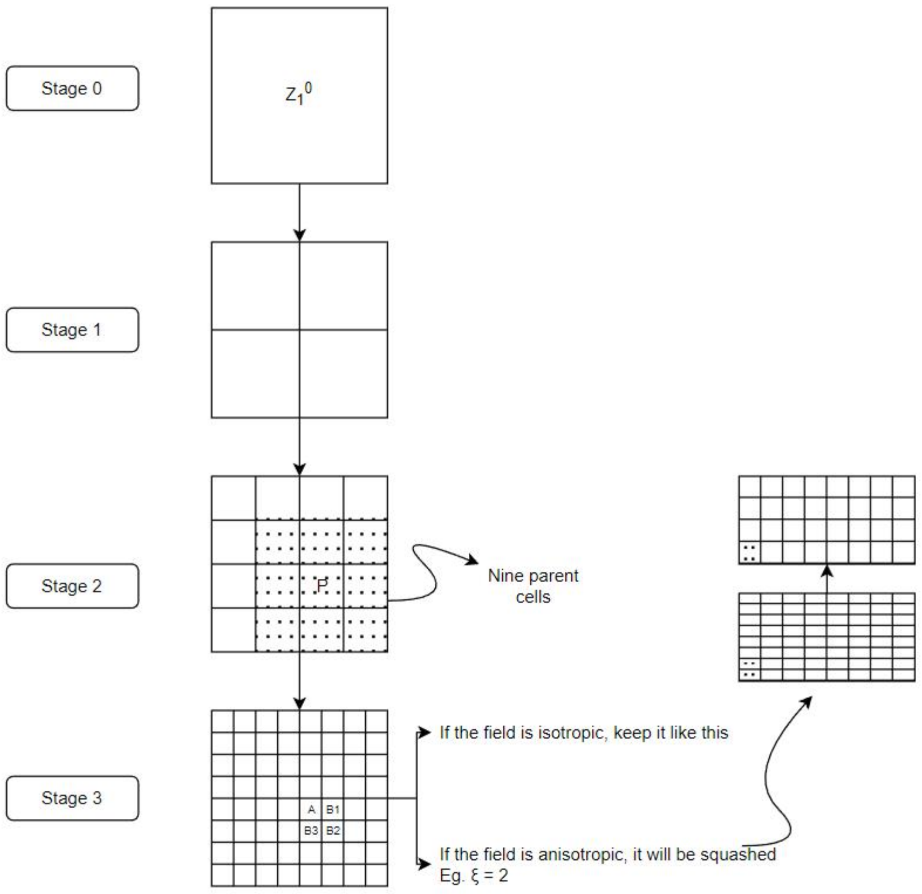
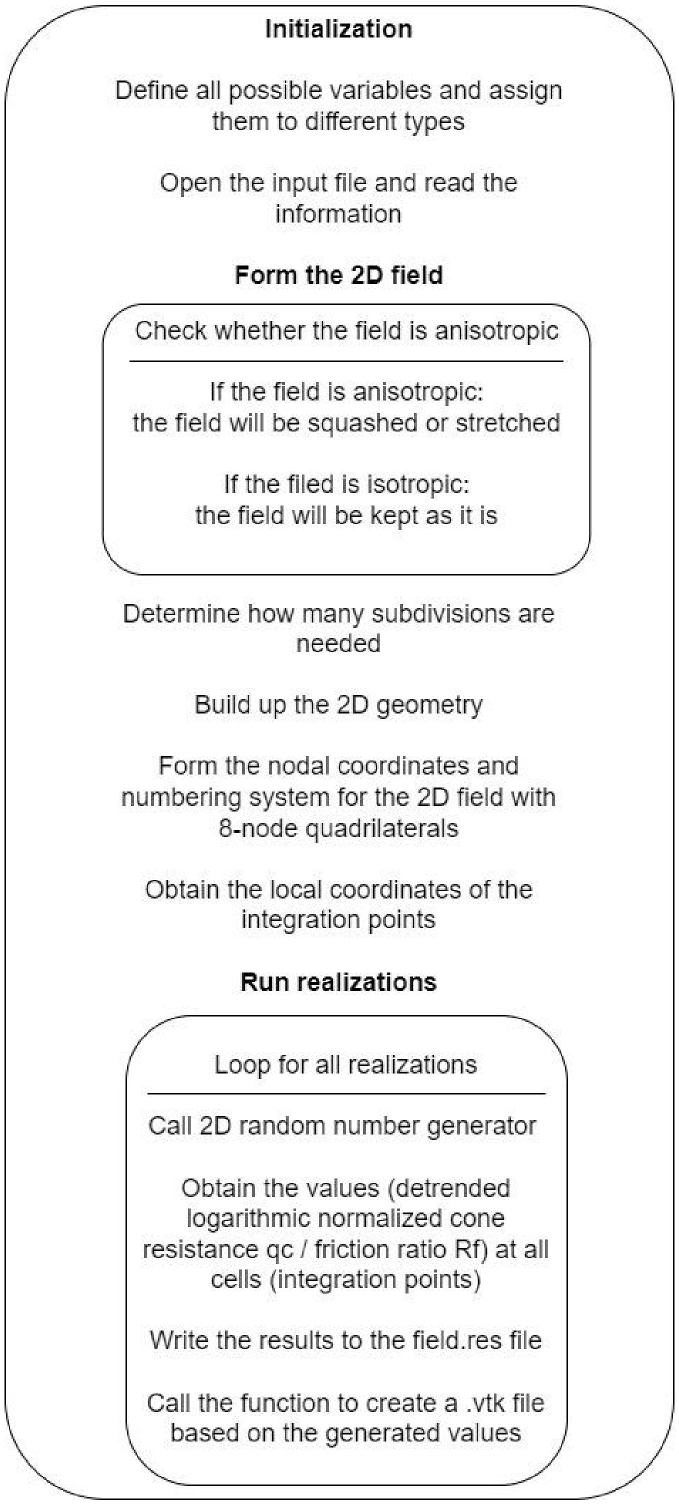

Stage 0
Z_1^0
Stage 1
Stage 2
P
Nine parent cells
Stage 3
A
B1
B3
B2
If the field is isotropic, keep it like this
If the field is anisotropic, it will be squashed
Eg. ξ = 2

Fig. 5. Local Average Subdivision process

The LAS generator is mainly built up in Fortran by Divya Varkey and there is a pseudo code to show how to achieve step by step in Fig. 6.



```
Initialization
    Define all possible variables and assign them to different types
    Open the input file and read the information
Form the 2D field
    Check whether the field is anisotropic
        If the field is anisotropic:
            the field will be squashed or stretched
        If the filed is isotropic:
            the field will be kept as it is
    Determine how many subdivisions are needed
    Build up the 2D geometry
    Form the nodal coordinates and numbering system for the 2D field with 8-node quadrilaterals
    Obtain the local coordinates of the integration points
Run realizations
    Loop for all realizations
        Call 2D random number generator
        Obtain the values (detrended logarithmic normalized cone resistance qc / friction ratio Rf) at all cells (integration points)
        Write the results to the field.res file
        Call the function to create a .vtk file based on the generated values
```

Fig. 6. Pseudo code for building random filed generator based on LAS in Fortran

3.2.2 Condition random field generator based on LAS and Kriging interpolation

Furthermore, the model can be improved to be more realistic and precise by adopting conditional random field (also implemented by Divya Varkey). This can be achieved by using Kriging interpolation method. Basically this method can be explained in the following equations (Lloret-Cabot, Hicks & van den Eijnden, 2012).

$$Z^*(x_0) = \sum_{\alpha=1}^{n} \lambda_\alpha Z(x_\alpha) \tag{5}$$

$$\sum_{\alpha=1}^{n} \lambda_\alpha = 1 \tag{6}$$

$Z^*(x_0)$ is a material property at a location x0, which can be obtained from a linear combination of the known values of Z at various measured points (i.e. $Z(x_\alpha)$). λ_α are n unknown weights, to be determined so as to find the best estimate for Z at x0.

It is worthy noticing that the Kriging interpolation cannot be directly applied to conditional random field analysis because there actually are no random variations over realizations. Instead, a superposition algorithm is adopted (Fenton & Griffiths, 2008):

$$X_c(z) = X_u(z) + X_m(z) - X_s(z) \tag{7}$$

where z = (z1, z2) is the location along which z1 and z2 are sampled and unsampled; Xc is the conditional simulation; Xu is the unconditional simulation (In this study Xu is the results generated by the unconditional random filed generator mentioned in Section 3.2.1); Xm is the Kriging prediction based on measured values; Xs is the Kriging prediction based on unconditional simulation.

Similarly, there is a pseudo code in Fortran in Fig. 7. One thing deserves noticing is that when there is no real CPT data as input, the program will take realization 1 as the CPT input. It will be elaborated later in Section 4.2.2.

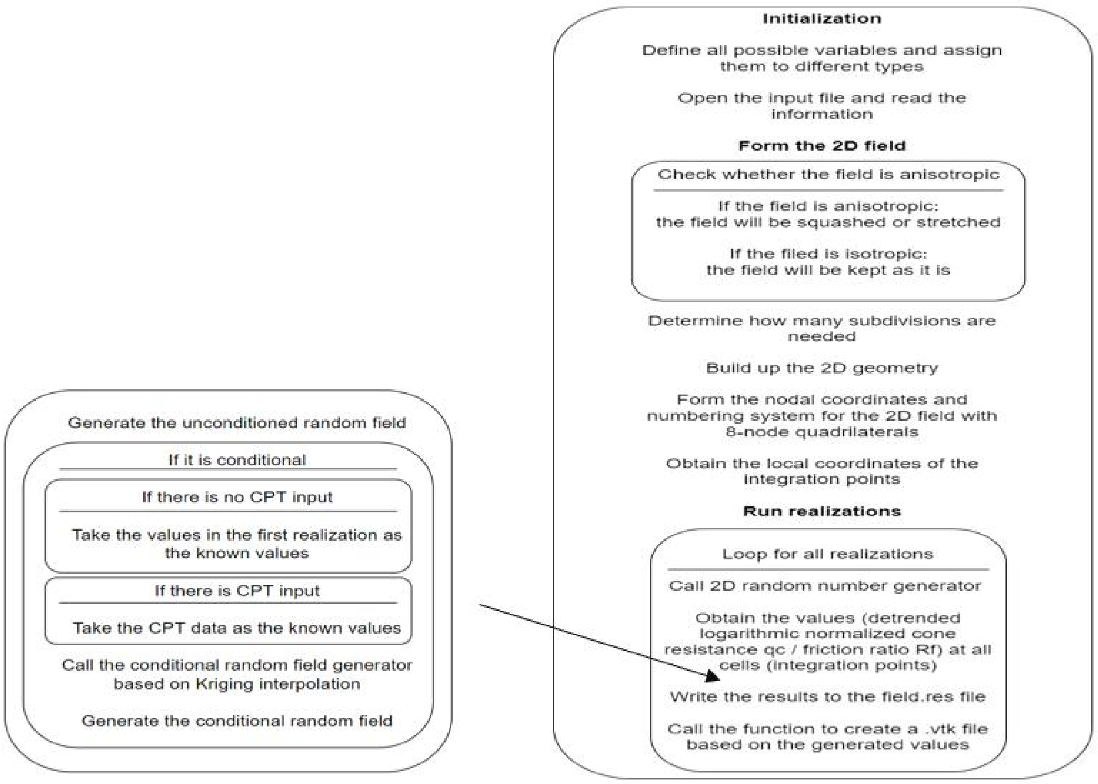

Initialization
Define all possible variables and assign them to different types
Open the input file and read the information
Form the 2D field
Check whether the field is anisotropic
If the field is anisotropic: the field will be squashed or stretched
If the filed is isotropic: the field will be kept as it is
Determine how many subdivisions are needed
Build up the 2D geometry
Form the nodal coordinates and numbering system for the 2D field with 8-node quadrilaterals
Obtain the local coordinates of the integration points
Run realizations
Loop for all realizations
Call 2D random number generator
Obtain the values (detrended logarithmic normalized cone resistance qc / friction ratio Rf) at all cells (integration points)
Write the results to the field.res file
Call the function to create a .vtk file based on the generated values
Generate the unconditioned random field
If it is conditional
If there is no CPT input
Take the values in the first realization as the known values
If there is CPT input
Take the CPT data as the known values
Call the conditional random field generator based on Kriging interpolation
Generate the conditional random field

Fig. 7. Pseudo code for building conditional random field generator based on LAS and Kriging interpolation in Fortran

3.3 Input parameter identification

For the random field generator, there must be input for the mean, standard deviation and scale of fluctuation of the value to be generated. In this study, there are two proposals how to determine those input. Proposal one is to use mean and standard deviation which can cover all classifications in Robertson's chart. This proposal departures from the idea that because of the inherent variability of the soils, any type of soil possibly exists in between two adjacent CPTs. So it is reasonable to use the mean and standard deviation, which can cover all possibilities (classifications). Specifically, the mean and standard deviation for logarithmic normalized cone resistance and friction ratio for this proposal is shown in Table 1.

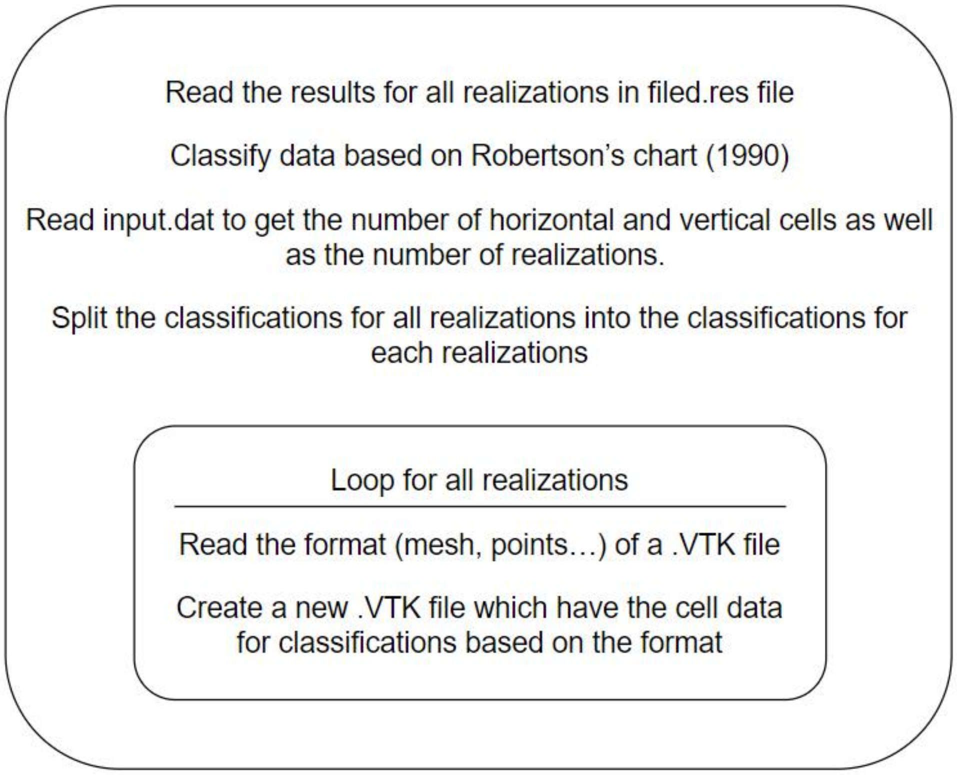
Table 1. Mean and standard deviation used in the first proposal

	mean	standard deviation
ln(nQt) [MPa]	3.5	1
ln(nFr) [%]	0	0.7

Another proposal is to use the average value of measured data (tip resistance and friction ratio) over all CPT profiles as the input mean and use the average of standard deviation for each CPT profile as the input standard deviation (Lloret-Cabot, Hicks & van den Eijnden, 2012). For scale of fluctuations, the vertical scale of fluctuation is determined by correlation function fit method, which compares the experimental correlation function plot and theoretical Markov correlation function to see which value of θv fits best. The horizontal scale of fluctuation is determined by the principle that the natural deposit has a ξ more than 25.

3.4 Automatic data processing

Countless data will be generated by the conditional random field generator, so a series of scripts must be made to process the data automatically and produce the meaningful results. Those scripts are mainly created in Python. Fig. 8 is the pseudo code which illustrates the main process. After running enough realizations, the results for all realizations are store in a res file. Here the results mean the generated logarithmic normalized tip resistance and friction ratio. Subsequently the aforementioned CPT classification script will be applied. So now the classifications for all realizations have been obtained, which later will be split into classifications for each realization and written in separate .vtk file. The .vtk file can be shown in Paraview to achieve the data visualization.

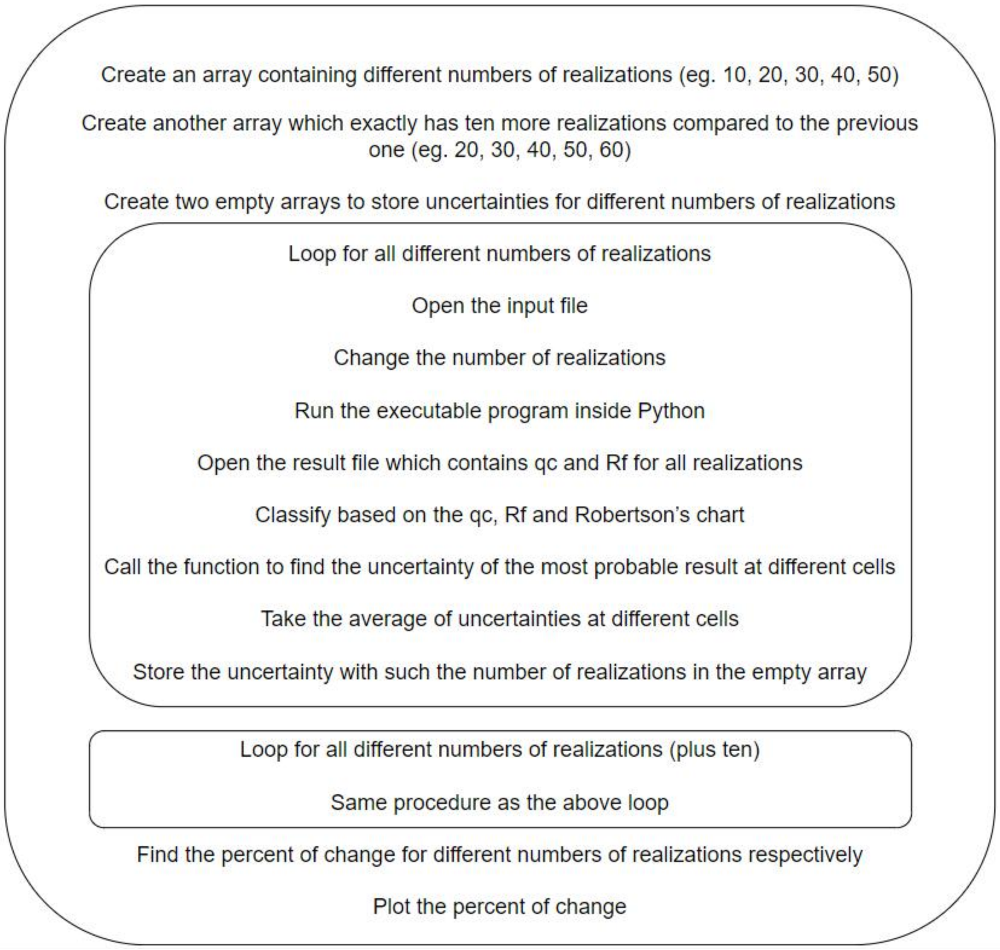


```
Read the results for all realizations in filed.res file
Classify data based on Robertson's chart (1990)
Read input.dat to get the number of horizontal and vertical cells as well
as the number of realizations.
Split the classifications for all realizations into the classifications for
each realizations
    Loop for all realizations
        Read the format (mesh, points…) of a .VTK file
        Create a new .VTK file which have the cell data
        for classifications based on the format
```

Fig. 8. Pseudo code for the main process to process generated random values

There is one remaining demand to successfully run the above script: determining how many realizations are enough. The way to figure out how many realizations are enough is to add ten more realizations to the current number of realizations and compare the change of a certain result.

If the change of such a result is less than the tolerance, the number of realizations is enough. Specifically, the result used to compare is the uncertainty of the most probable classifications and the tolerance is 0.05% (Shi & Wang, 2021b). The pseudo code is shown in Fig. 9.



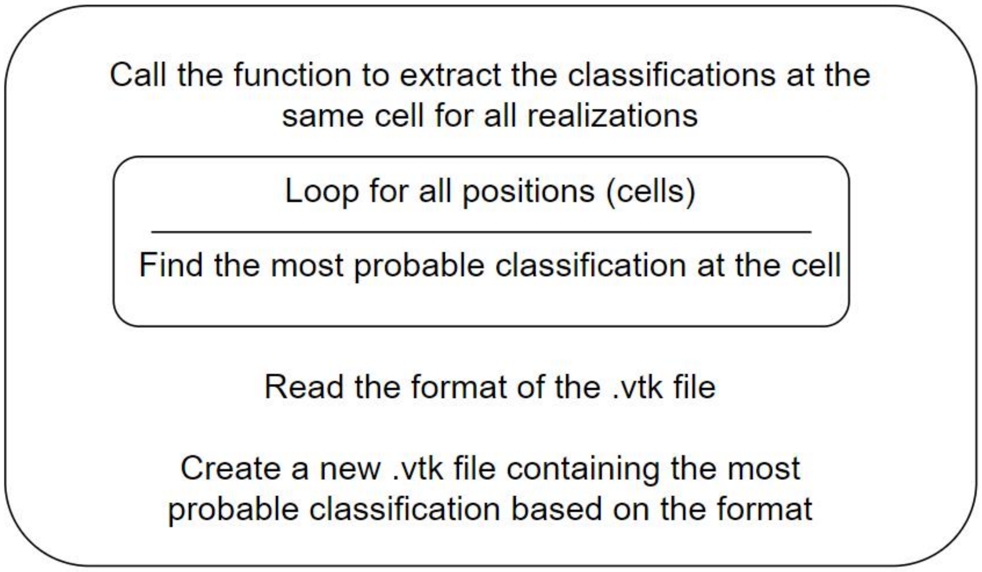
```
Create an array containing different numbers of realizations (eg. 10, 20, 30, 40, 50)
Create another array which exactly has ten more realizations compared to the previous one (eg. 20, 30, 40, 50, 60)
Create two empty arrays to store uncertainties for different numbers of realizations
    Loop for all different numbers of realizations
        Open the input file
        Change the number of realizations
        Run the executable program inside Python
        Open the result file which contains qc and Rf for all realizations
        Classify based on the qc, Rf and Robertson's chart
        Call the function to find the uncertainty of the most probable result at different cells
        Take the average of uncertainties at different cells
        Store the uncertainty with such the number of realizations in the empty array
    Loop for all different numbers of realizations (plus ten)
        Same procedure as the above loop
Find the percent of change for different numbers of realizations respectively
Plot the percent of change
```

Fig. 9. Pseudo code for determination of the number of realizations

There is a subsequent demand to successfully run script shown in Fig. 9: how to find the uncertainty of the most probable result. Actually the prior demand is to determine what the most probable result is. It is not a certain realization but a comprehensive result which reflects the classification that appears most frequently over realizations at each cell. For example, for a certain cell, after 5 realizations, the classifications are respectively 3, 4, 5, 5, 5. Then type 5 is the most probable classification for this cell. The most probable classifications for all cells can be determined in the same way. Thus the most probable result can be obtained. The uncertainty of the most probable result is the average uncertainty of each cell. The uncertainty of each cell is to calculate ratio between the classifications which are different from the most probable classification and the number of realizations. Using the same example early in this paragraph, the uncertainty of the cell is 40%. This uncertainty is also known as dispersion (Shi & Wang, 2021b). The dispersion can be calculated by the following formula:

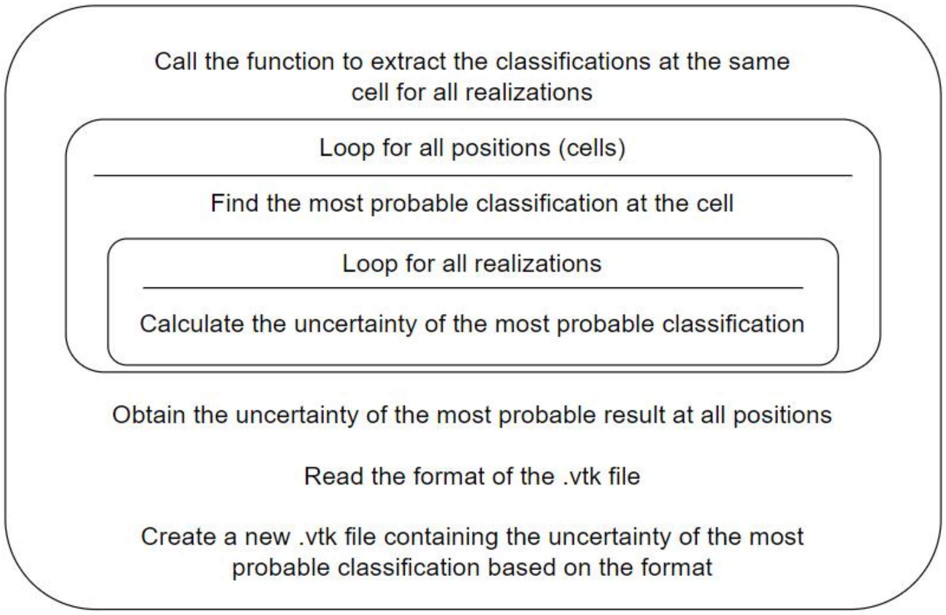
$$Dp(x) = \frac{\sum_{r=1}^{N_r} I[Z_r(x) \neq Z_{mp}(x)]}{N_r} \tag{8}$$

where $Dp(x)$ is dispersion at location x; Zr(x) is the categorical variable of rth realization at location x; Zmp(x) represents the most probable classification value at location x; Nr denotes the total number of realizations. The pseudo codes for those two demands are shown in Fig. 10 and Fig. 11.



```
Call the function to extract the classifications at the same cell for all realizations
    Loop for all positions (cells)
        Find the most probable classification at the cell
Read the format of the .vtk file
Create a new .vtk file containing the most probable classification based on the format
```

Fig. 10. Pseudo code for determination of the most probable result



```
Call the function to extract the classifications at the same cell for all realizations
    Loop for all positions (cells)
        Find the most probable classification at the cell
        Loop for all realizations
            Calculate the uncertainty of the most probable classification
Obtain the uncertainty of the most probable result at all positions
Read the format of the .vtk file
Create a new .vtk file containing the uncertainty of the most probable classification based on the format
```

Fig. 11. Pseudo code for determination of the uncertainty of the most probable result

There is a lastly remaining demand: extracting the classification results for each cell over realizations. The pseudo is shown in Fig. 12. The code implementation for pseudo codes from Fig. 8 to Fig. 12 is shown in Appendix A to Appendix C.

```
Find the period
Create an empty list to store classifications at different positions (cells)
Loop for all positions
    Create an empty array to store the classifications at the same positions (cells)
    Loop for all realizations
        Extract the classification from the file containing classifications for all realizations with the index of position
        Store the classification at the same position in the array
        Index = index + period
    Store the the array in the list
```

Fig. 12. Pseudo code for extraction of the classifications at the same cell over realizations

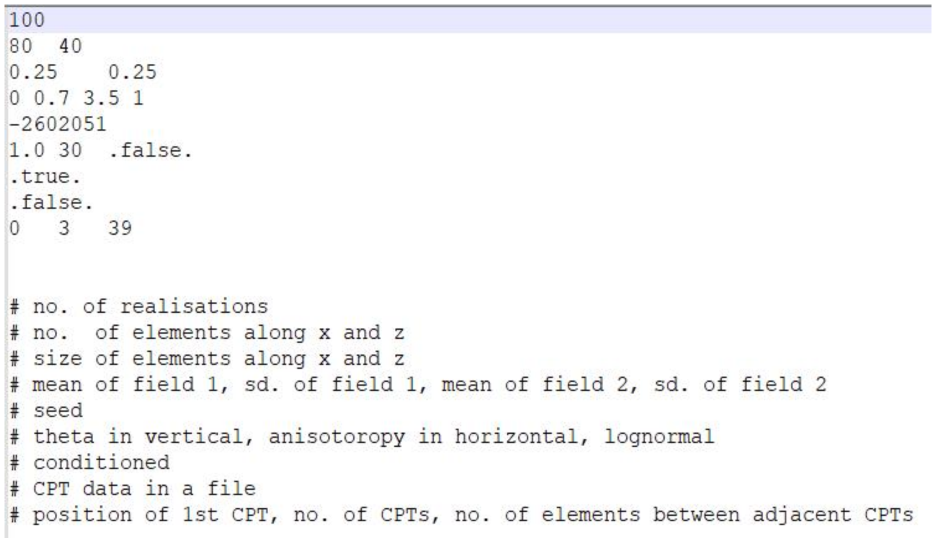
4. Illustrative example

4.1 Input determination

In this part, there is an example to show what can be obtained from the generator and accessary scripts. How to determine the input for the generator is firstly described. Fig. 13 shows the input file. The number of realizations is left to be determined after determining all the rest of input. The domain size is 20m by 10m. The mean, std are values which can cover all the classifications in the Robertson's chart, which correspond to Table 1. The vertical scale of fluctuation in this example cannot be calculated by the correlation function fit method because there are no available real CPTs. So θv is assumed to be a reasonable value (e.g. 1m) and the horizontal one is assumed to be 30m (for natural soil $\xi > 25$). The element size then can be calculated as θv/4 = 0.25m (Hicks, 2009).

Notice in this generator there is only one integration point in an element so the cell size is equal to the element size. Since there are no available CPTs, after switching to conditional random field generator, the program will automatically choosing realization 1 as the input CPT which will be used for Kriging interpolation and won't be changed in the following realizations. Notice not all columns are be taken as the input CPT but the columns where the user set to have conducted CPTs. In this example, as can be seen in raw 9 in Fig. 13, there are three columns to have CPTs, respectively corresponding to column 0, column 40 and column 79 (the numbering starts from 0). The unconditional results are also generated from the program, which can be used for the comparison between conditional and unconditional scenarios.

Now moving backing towards the number of realizations, it is determined by the script shown in Appendix B. Other inputs remain the same for different numbers of realizations. It can be seen in Fig. 14, when the number of realizations is equal to 100, the percent of change is around 0.01%, which is smaller than the tolerance (0.05%). So it is logical to run for 100 realizations.



```
100
80  40
0.25    0.25
0 0.7 3.5 1
-2602051
1.0 30  .false.
.true.
.false.
0   3   39


# no. of realisations
# no.  of elements along x and z
# size of elements along x and z
# mean of field 1, sd. of field 1, mean of field 2, sd. of field 2
# seed
# theta in vertical, anisotoropy in horizontal, lognormal
# conditioned
# CPT data in a file
# position of 1st CPT, no. of CPTs, no. of elements between adjacent CPTs
```

Fig. 13. Input for the illustrative example

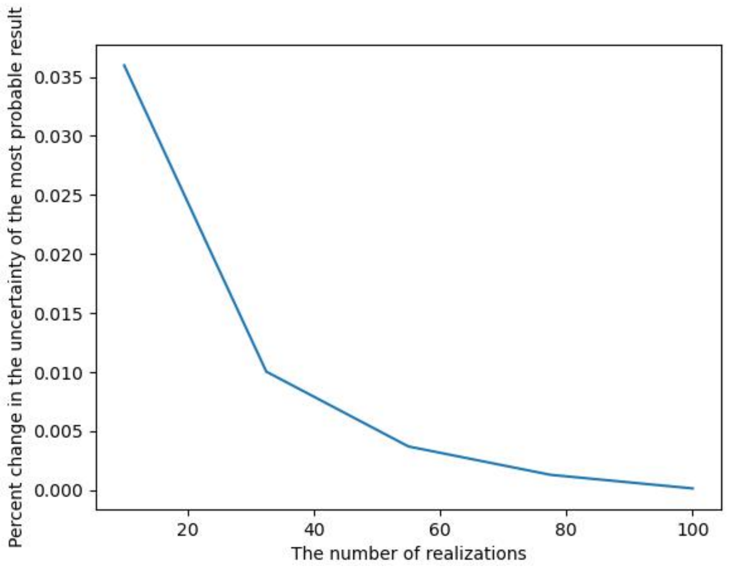


Fig. 14. Percent change in the uncertainty of the most probable result over realizations

4.2 Results and analysis

Subsequently, the generated data is processed by the aforementioned Python scripts from Appendix A to Appendix C and Fig. 15 to Fig. 20 show the final results.

4.2.1 Most probable result for classifications and its uncertainty

The most probable result is shown in Fig. 15 and it can be seen that the main soil type is 5. This is because the middle part of Robertson's chart is chosen as the mean, so type5, which locates in the middle of the chart, appears most frequently. The uncertainty is shown in Fig. 16 and the average uncertainty is 17.2%. It can be seen that the uncertainty decreases as the cell approaches to any side of the adjacent CPTs. In other words, the uncertainty is the highest in the middle part of two adjacent CPTs while the uncertainty is the lowest (actually 0%) in the conducted CPT positions. This phenomenon is in alignment with the reality.

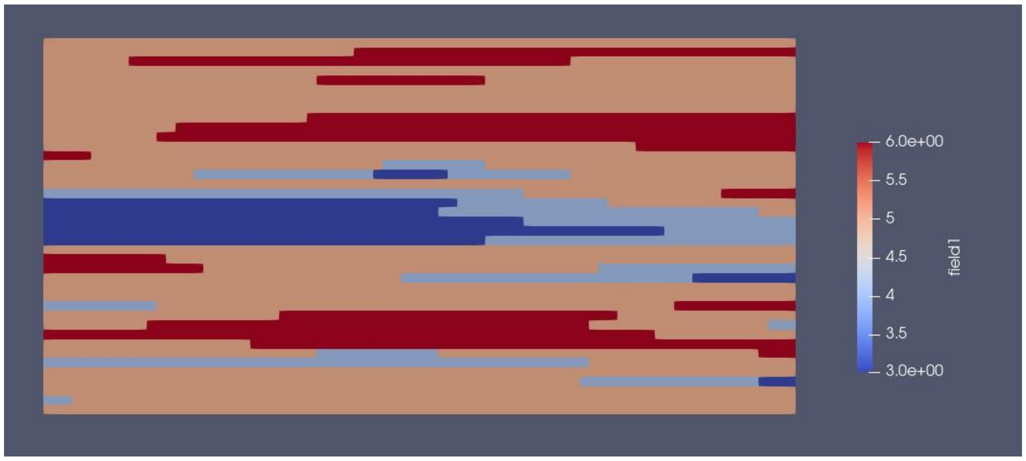


Fig. 15. The most probable result of classifications in the illustrative example

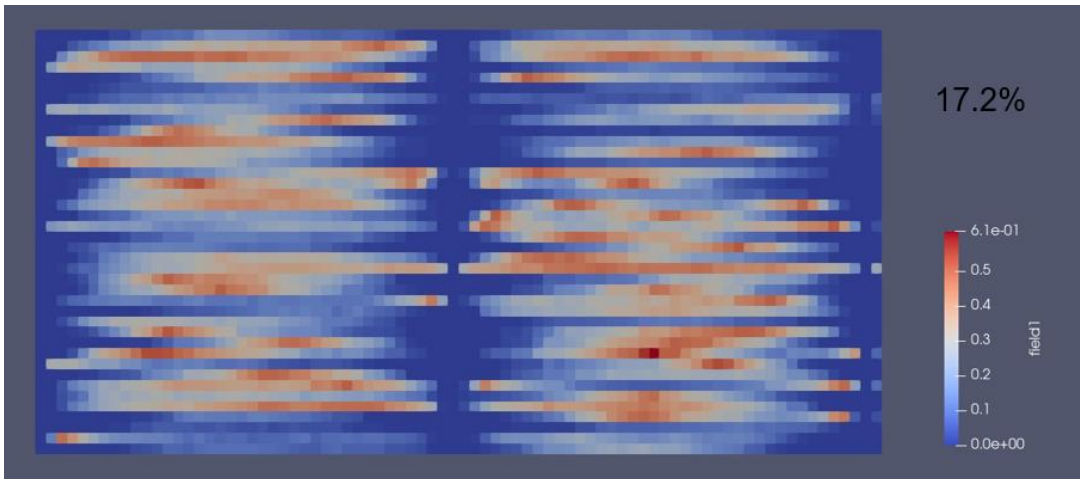


Fig. 16. The uncertainty of the most probable result of classifications in the illustrative example with an average uncertainty of 17.2%

4.2.2 Comparison between conditional and unconditional random field generator

The first comparison is to show what the difference between conditional and unconditional random field generator is. For conditional scenario, as mentioned earlier, in all realizations the classifications at conducted CPT positions don't change, while for unconditional scenario, there is no such a constrain. This phenomenon can be seen in Fig. 17 and Fig. 18. For simplicity, only column 0 is compared. In addition, no matter it is conditional or unconditional scenario, the 1st realization is always the same as shown in Fig. 19.

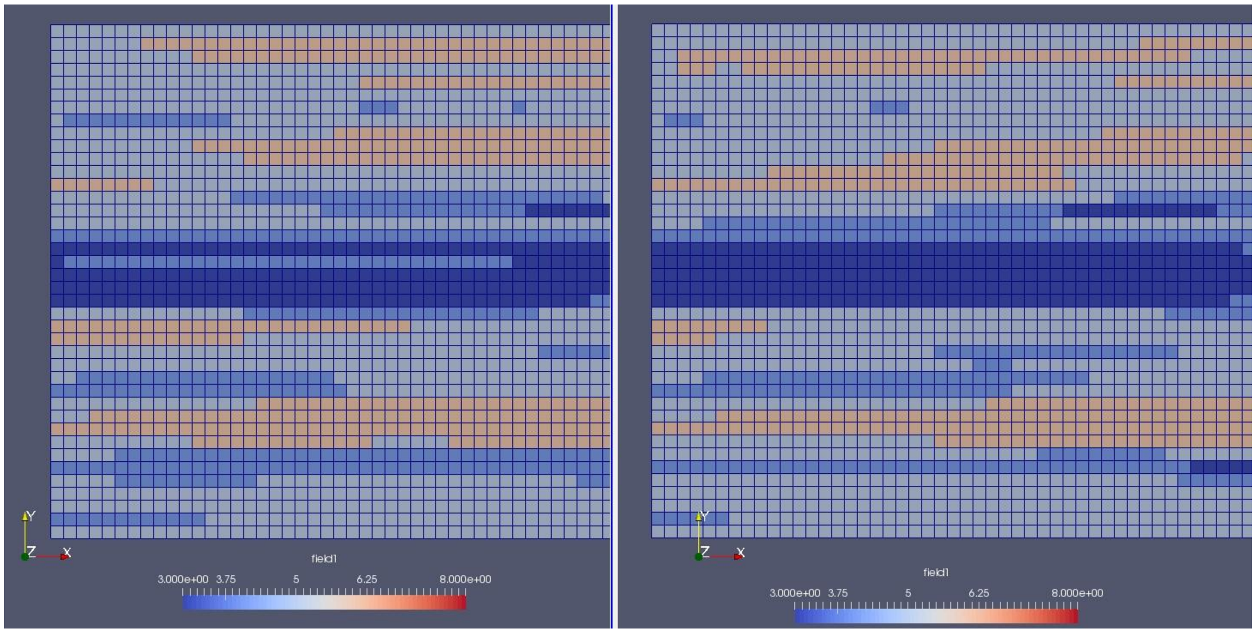


Fig. 17. The classifications in the first and second realization in conditional random field, comparing the first column on the left side

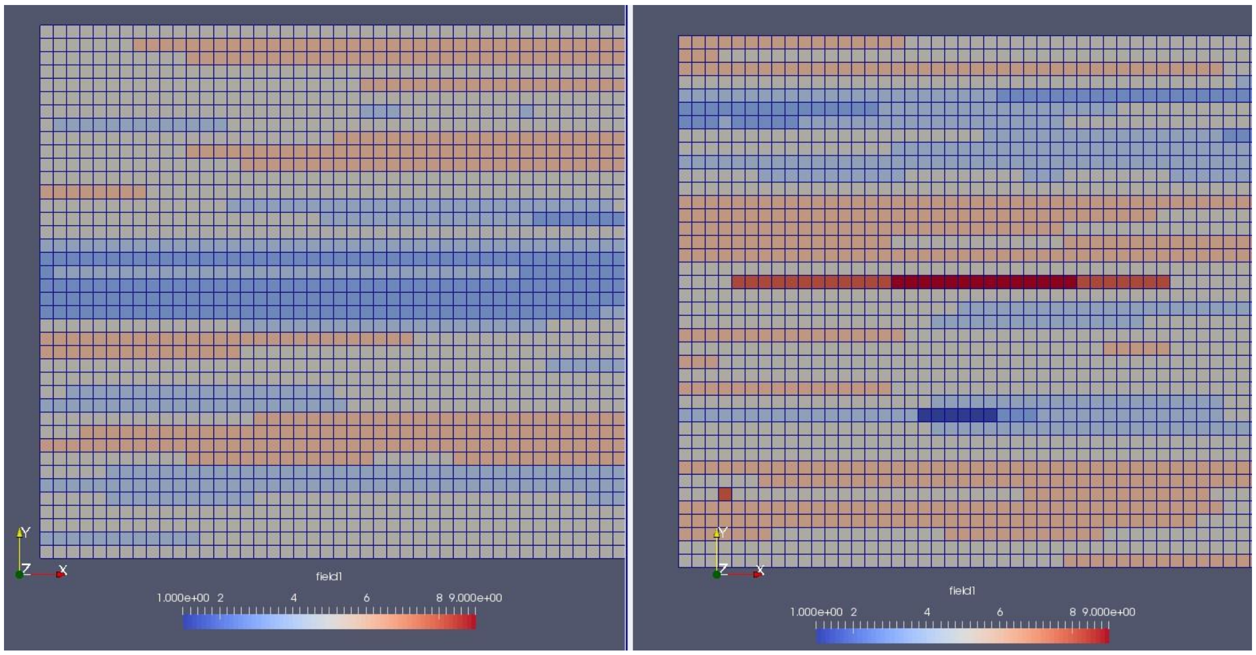


Fig. 18. The classifications in the first and second realization in unconditional random field, comparing the first column on the left side

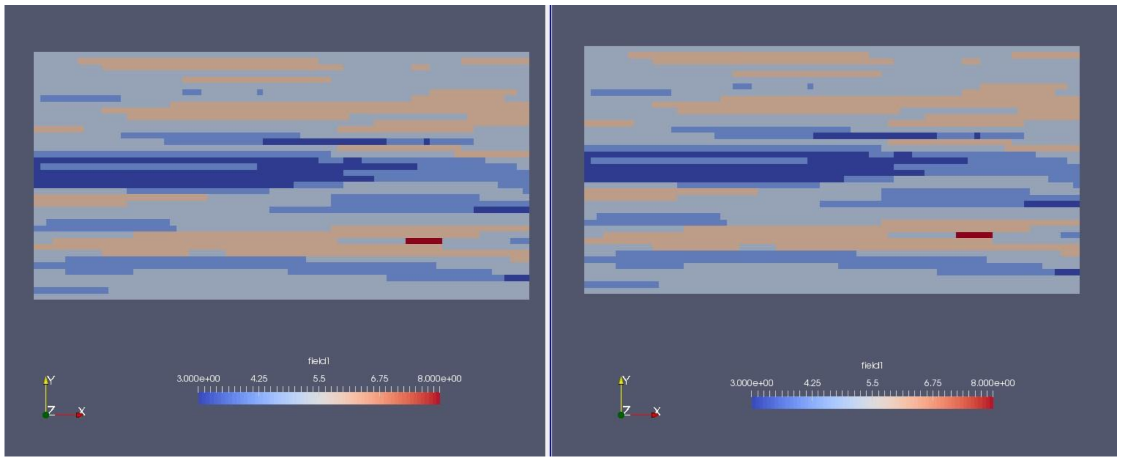


Fig. 19. The classifications of the first realization in conditional random field (left) and unconditional random field (right)

Moreover, the average uncertainty of the conditional and unconditional random field generator can be compared. It is shown in Fig. 20. It is evident that the average uncertainty is higher in unconditional scenario, which is logical because for the conditional generator, based on Kriging interpolation, there is more information extracted from the real CPTs or there are more constrains from the real CPTs for the random field. So definitely the uncertainty is reduced in conditional random field.

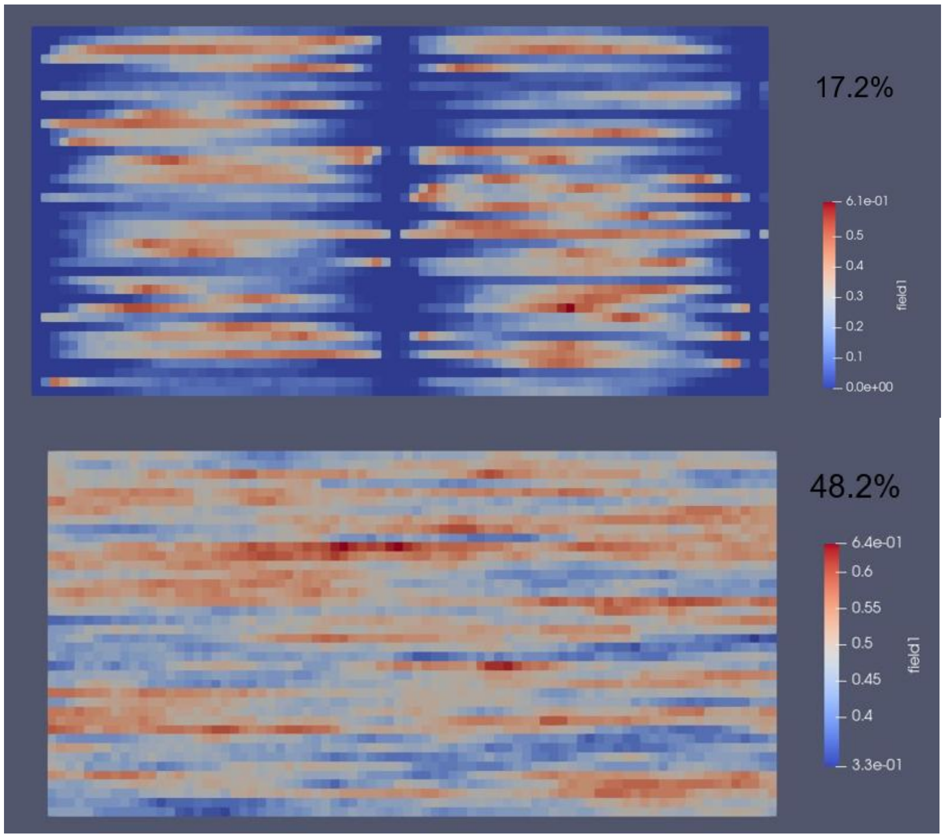


Fig. 20. The average uncertainty comparison between the conditional random field (17.2%, upper) and the unconditional random field (48.2%, lower)

4.2.3 Convergence in the unconditional random field

Since in the unconditional random field, the generation of random values doesn't take Kriging interpolation into consideration, there is one speculation that with the increase of realizations, the most probable classifications for each cell will converge to a certain type. Additionally, because the generation conforms with Gaussian distribution shown in Fig. 21, the convergence type is speculated to be type 5.

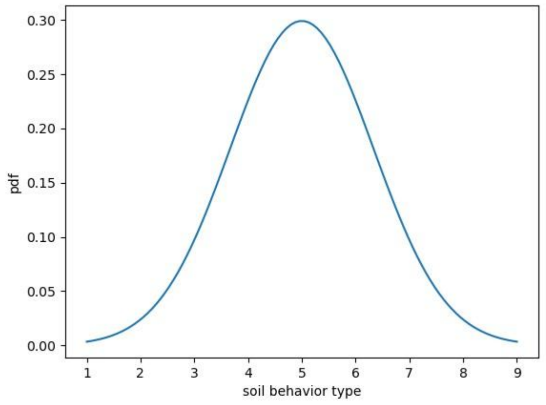


Fig. 21. The normal distribution of soil behaviour type

The method is similar to the method to determine how many realizations are enough. Specifically, the current most probable result and the most probable result with ten more realizations are compared but this time the focus is on the classifications instead of the average uncertainty. The result is plotted in Fig. 22. The difference refers to the difference between two most probable results of classifications. For each cell, if the classification is different when the number of realizations is n and when the number of realizations is n+10, the counter will be added by 1. After looping for all cells, counter will be divided by the total number of cells and that is the difference. It can be found that as the increase of realizations, the difference tend to be 0. So it proves the speculation that in unconditional random field, the classification for each cell tends to converge when there are enough realizations.

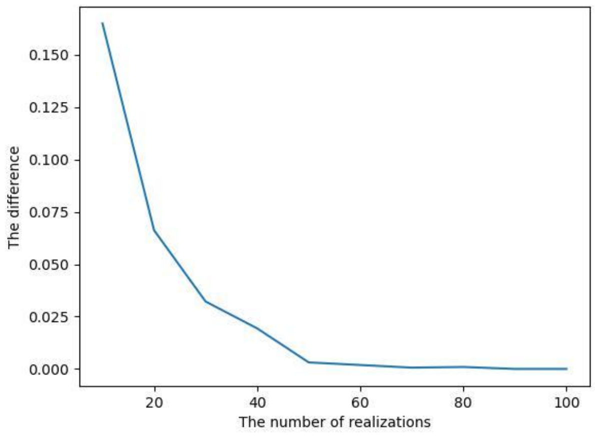


Fig. 22. The difference between the most probable result of classifications of n realizations and n+10 realizations in unconditional random field

Additionally, the most probable result for 100 and 110 realizations can be viewed in Paraview to have a more direct comparison. They are shown respectively in Fig. 23 and Fig. 24. It can be also seen that the convergence type is type 5, which conforms with the second speculation.

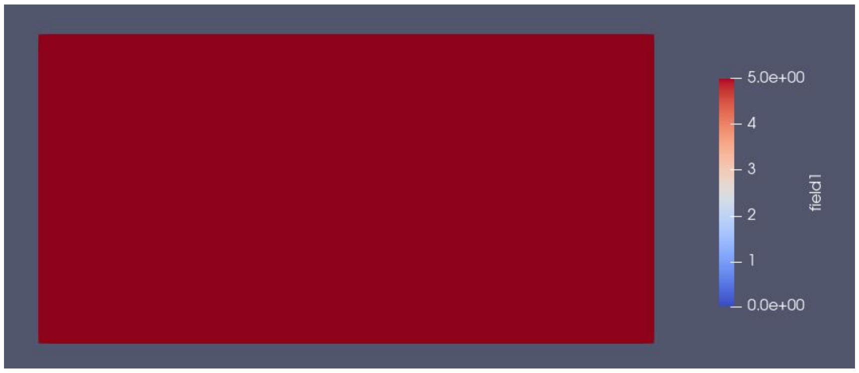


Fig. 23. The most probable result (r = 100)

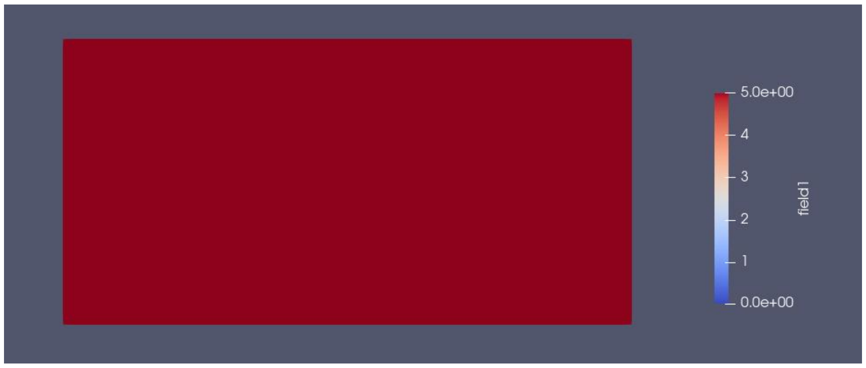


Fig. 24. The most probable result (r = 110)

Finally, with the increase of the number of realizations, actually the average uncertainty has no big changes. It is also logical because the mean can only contain around 50% reliability. It’s shown in Fig. 25.

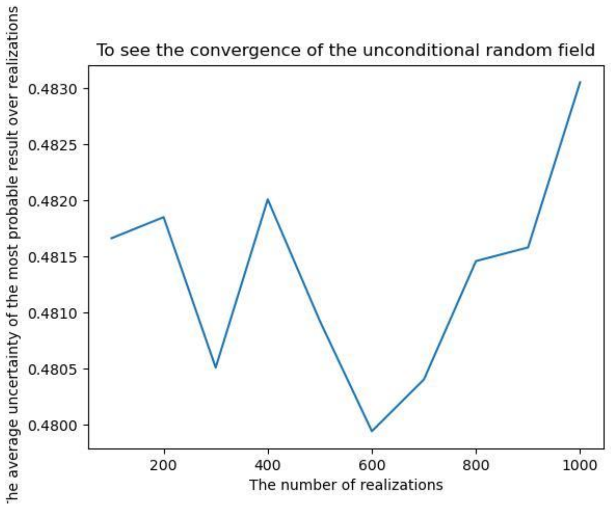


Fig. 25. The average uncertainty in the unconditional random field over realizations

Whereas in the conditional field, because the existence of Kriging interpolation, the classifications in each cell cannot converge no matter how many realizations are. The difference exists permanently, which is shown in Fig. 26.

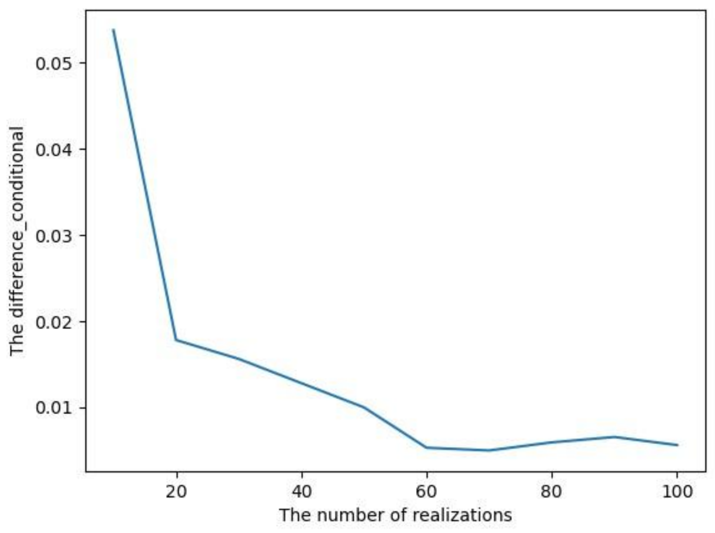


Fig. 26. The difference between the most probable result of classifications of n realizations and n+10 realizations in conditional random field

5. Case Study

5.1 Background

In this part, the proposed conditional random field generator is applied to a real case, which aims to achieve probabilistic soil classifications using three conducted CPTs. Those three CPTs are obtained from https://www.dinoloket.nl/en, respectively named as CPT000000161781, CPT000000161782 and CPT000000161785. Their relative positions are shown in Fig. 27. For simplicity, their names are changed to CPT1, CPT2 and CPT5. All of them are projected to x axis and based on their coordinates, it can be calculated that the spacings are 22 m and 25 m. So in the random field, CPT1 can be approximately regarded as exactly in the middle of CPT2 and CPT5. What those three CPTs look like in the random field is shown in Fig. 28.

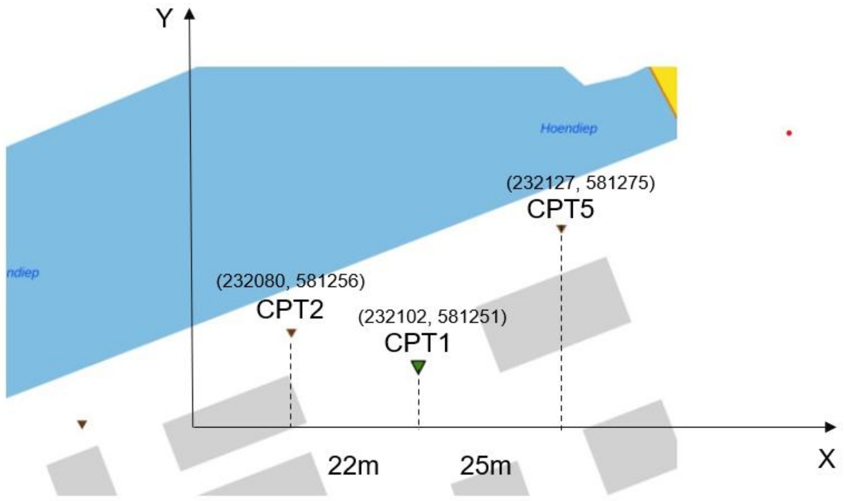


Fig. 27. The real locations of the three CPTs used in case study

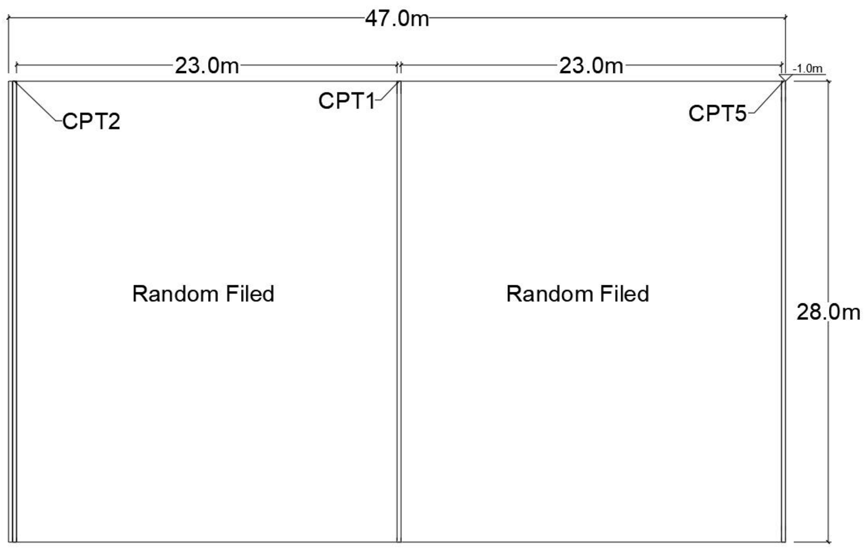


Fig. 28. The schematic diagram for the locations of the three CPTs in the random field

It's worthy noticing that because there are some differences of those CPTs in terms of the initial depth and CPT length, some data has to be deleted in order to make those three profiles uniform. After some adjustments, all CPTs start from one meter below the ground and have a depth of 28 meters. The horizontal size is 47 meters. In all, it is a 47m by 28m domain locating at one meter below the ground.

5.2 Input data

The input mean and standard deviation for this case adopt the second proposal in Section 3.3. The mean is the average values for all CPT profiles. The standard deviation is the average of the standard deviation for each CPT. Those two parameters for both cone resistance and friction ratio are calculated by Python script, which can be seen in Appendix D1. The results are shown in Table 2.

Table 2. Mean and standard deviation calculated from the three CPTs (the second proposal)

	mean	standard deviation
ln(nQt) [MPa]	3.81	1
ln(nFr) [%]	0.36	0.76

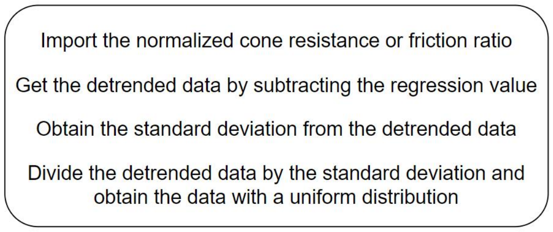
Additionally, as for the vertical scale of fluctuation, it is calculated by trying to find value which fits the Markov correlation function to the experimental correlation function. Those two functions are shown in Eq. (9) and Eq. (10) respectively:

$$\rho(\Delta z) = \exp(-2\frac{\Delta z}{\theta}) \tag{9}$$

$$\hat{\rho}(\Delta z) = \frac{1}{n_{\Delta z}} \sum_{i=1}^{n_{\Delta z}} U(z_i) U(z_i + \Delta z) \tag{10}$$

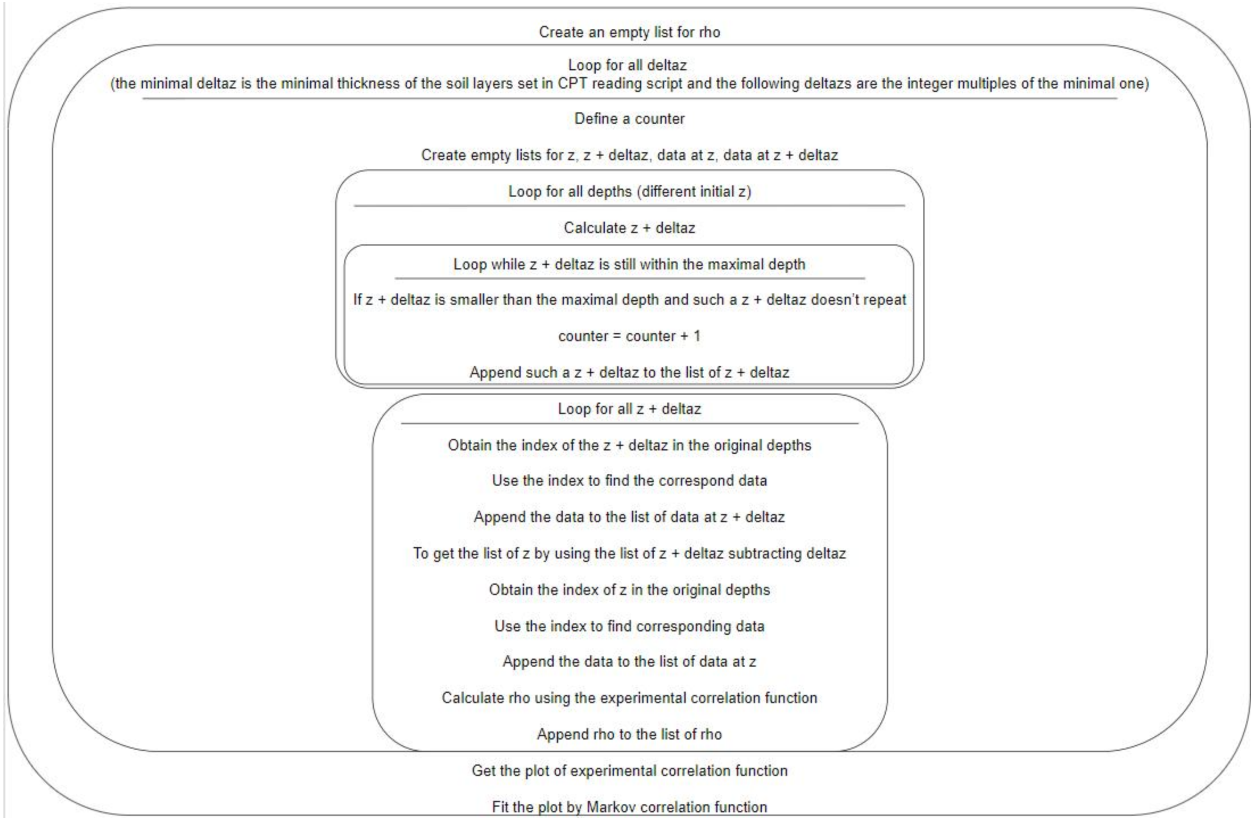
where $U(z_i)$ and $U(z_i + \Delta z)$ are a pair of detrended random field data points at relative distance Δz, $n_{\Delta z}$ is the number of pairs available in the data at a distance Δz and θ is the vertical scale of fluctuation.

How to implement the calculation of vertical scale of fluctuation can be seen in Appendix D2. Fig. 29 and Fig. 30 show the conceptual process. For the horizontal scale of fluctuation, a reasonable anisotropy of heterogeneity is assumed to calculate $\theta_h = \theta_v \times \xi$.



```
Import the normalized cone resistance or friction ratio
Get the detrended data by subtracting the regression value
Obtain the standard deviation from the detrended data
Divide the detrended data by the standard deviation and
obtain the data with a uniform distribution
```

Fig. 29. Pseudo code for calculating the normalized cone resistance or friction ratio with a uniform distribution



```
Create an empty list for rho
    Loop for all deltaz
    (the minimal deltaz is the minimal thickness of the soil layers set in CPT reading script and the following deltazs are the integer multiples of the minimal one)
        Define a counter
        Create empty lists for z, z + deltaz, data at z, data at z + deltaz
            Loop for all depths (different initial z)
                Calculate z + deltaz
                    Loop while z + deltaz is still within the maximal depth
                        If z + deltaz is smaller than the maximal depth and such a z + deltaz doesn't repeat
                        counter = counter + 1
                        Append such a z + deltaz to the list of z + deltaz
            Loop for all z + deltaz
                Obtain the index of the z + deltaz in the original depths
                Use the index to find the correspond data
                Append the data to the list of data at z + deltaz
                To get the list of z by using the list of z + deltaz subtracting deltaz
                Obtain the index of z in the original depths
                Use the index to find corresponding data
                Append the data to the list of data at z
                Calculate rho using the experimental correlation function
                Append rho to the list of rho
Get the plot of experimental correlation function
Fit the plot by Markov correlation function
```

Fig. 30. Pseudo code for calculating the vertical scale of fluctuation of normalized cone resistance and friction ratio

The results of those correlation functions are shown in Fig. 31 and Fig. 32. There is a phenomenon that the latter part of those two experimental correlation function curves reach some negative values. This is the because in the real CPT profile there are many variations. Thus when the relative distance is large, two points in a pair are likely to locate separately in weak zone and strong zone. Moreover, it is also caused by relatively little data and short CPT profiles. Still it can be approximately estimated that when θv = 2 m, Markov correlation function fits best for normalized friction ratio and when θv = 3 m, Markov correlation function fits best for normalized cone resistance. Eventually, the vertical scale of fluctuation used in the conditional random field generator is the average of these two values, 2.5m. Additionally, ξ is set to be 30, so the horizontal scale of fluctuation is 75m.

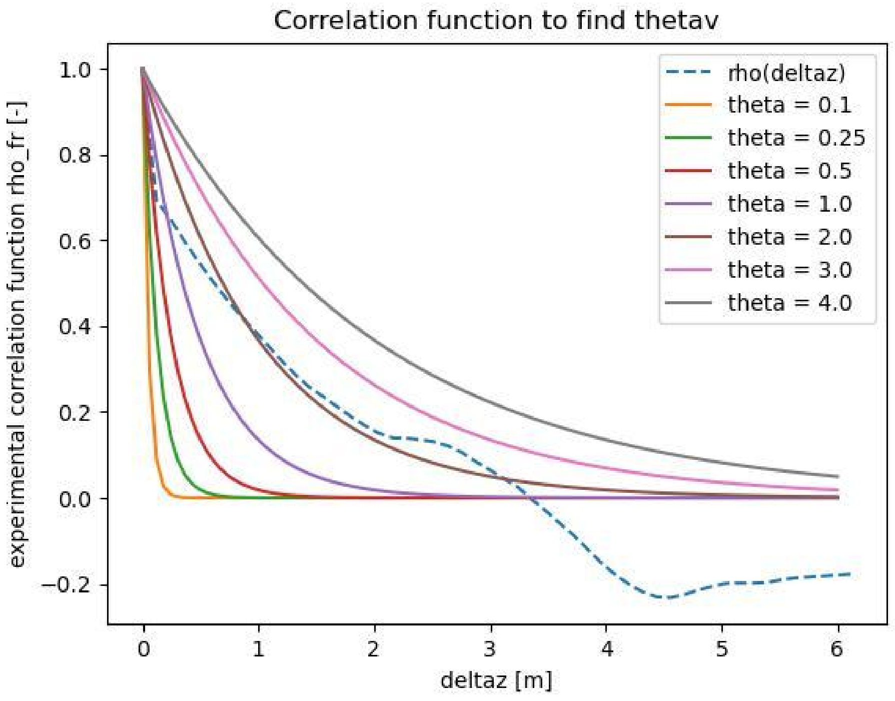


Fig. 31. The correlation function fit for normalized friction ratio

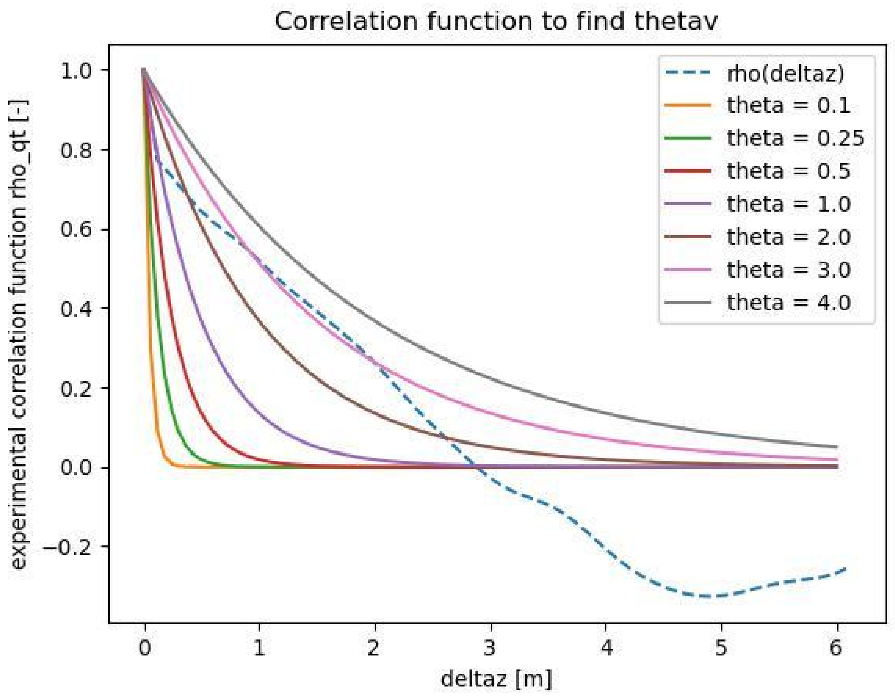
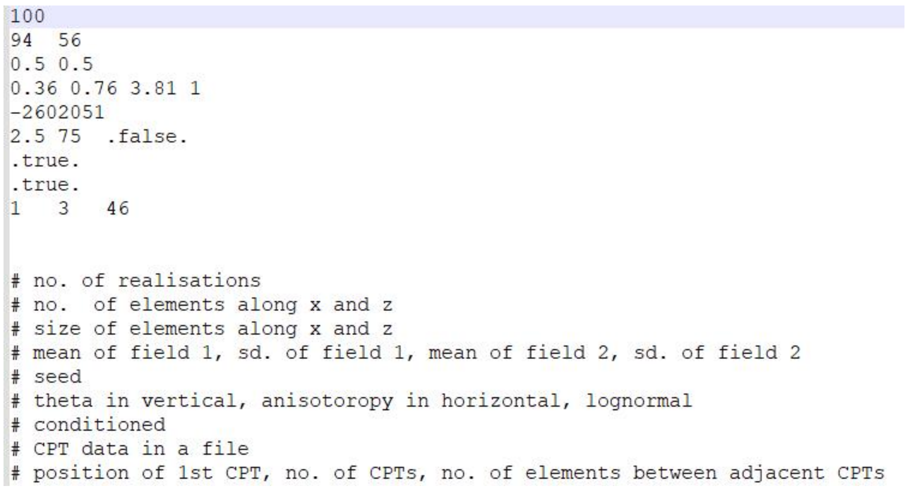


Fig. 32. The correlation function fit for normalized cone resistance

Finally, the cell size is determined by principle that the maximal cell size is no bigger than a quarter of minimal scale of fluctuation, so the cell size is chosen to be 0.5m. All the input is shown in Fig. 33.



```
100
94  56
0.5 0.5
0.36 0.76 3.81 1
-2602051
2.5 75  .false.
.true.
.true.
1   3   46


# no. of realisations
# no.  of elements along x and z
# size of elements along x and z
# mean of field 1, sd. of field 1, mean of field 2, sd. of field 2
# seed
# theta in vertical, anisotoropy in horizontal, lognormal
# conditioned
# CPT data in a file
# position of 1st CPT, no. of CPTs, no. of elements between adjacent CPTs
```

Fig. 33. The input for the case study

5.3 Results and analysis

Similarly, the generated data is processed by the aforementioned Python scripts from Appendix A to Appendix C and Fig. 34 to Fig. 36 show the final results.

5.3.1 Most probable result for classifications and its uncertainty

Fig. 34 illustrates the most probable result for classifications in this case study and it can be seen that type 5 is no longer predominant. This is because with the participation of Kriging interpolation and the real CPTs, the distribution for each cell doesn't conform with Gaussian distribution but is subject to the conducted CPT data. Fig. 35 shows the uncertainty of the most probable result and the average uncertainty is 4.58%. The phenomenon that, the uncertainty is the highest in the middle part of two adjacent CPTs while the uncertainty is the lowest (actually 0%) in the conducted CPT positions, can be also seen in Fig. 35.

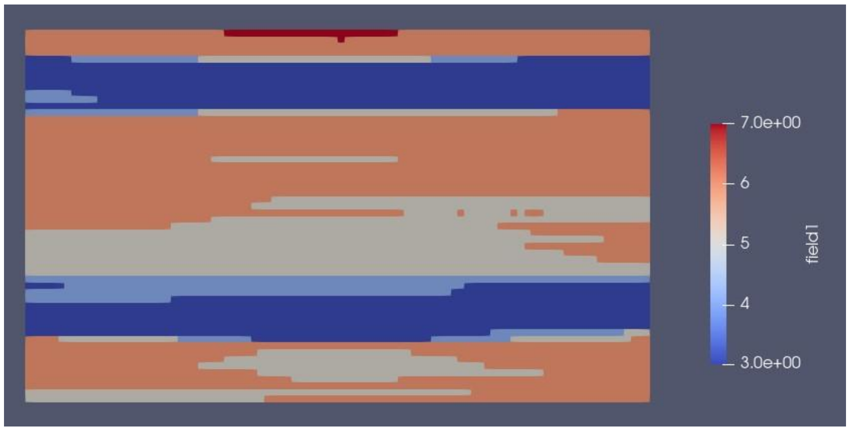


Fig. 34. The most probable result of classifications in the case study

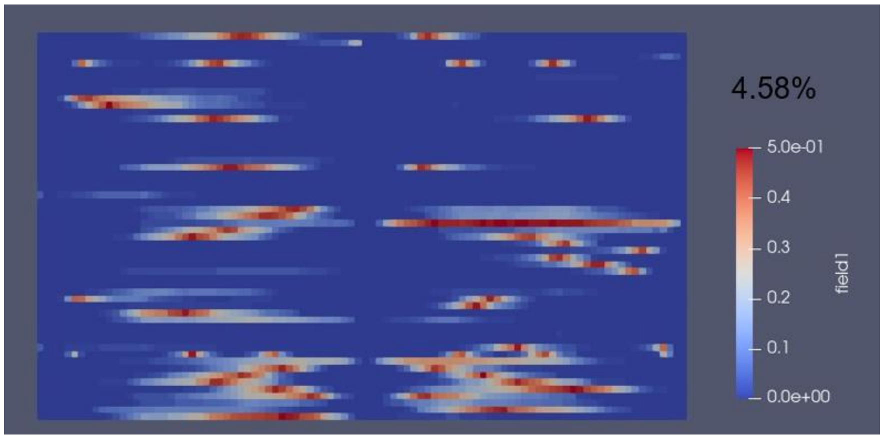


Fig. 35. The uncertainty of the most probable result of classifications in the case study, with an average uncertainty of 4.58%

5.3.2 Comparison between conditional and unconditional random field generator

It has been shown that the column where to set CPT doesn't change over realizations in the conditional random filed in Chapter 4, so it's not repeated here. Only the average uncertainty is compared between conditional and unconditional random. Fig. 36 also illustrates that the uncertainty of unconditional scenario is higher than conditional scenario.

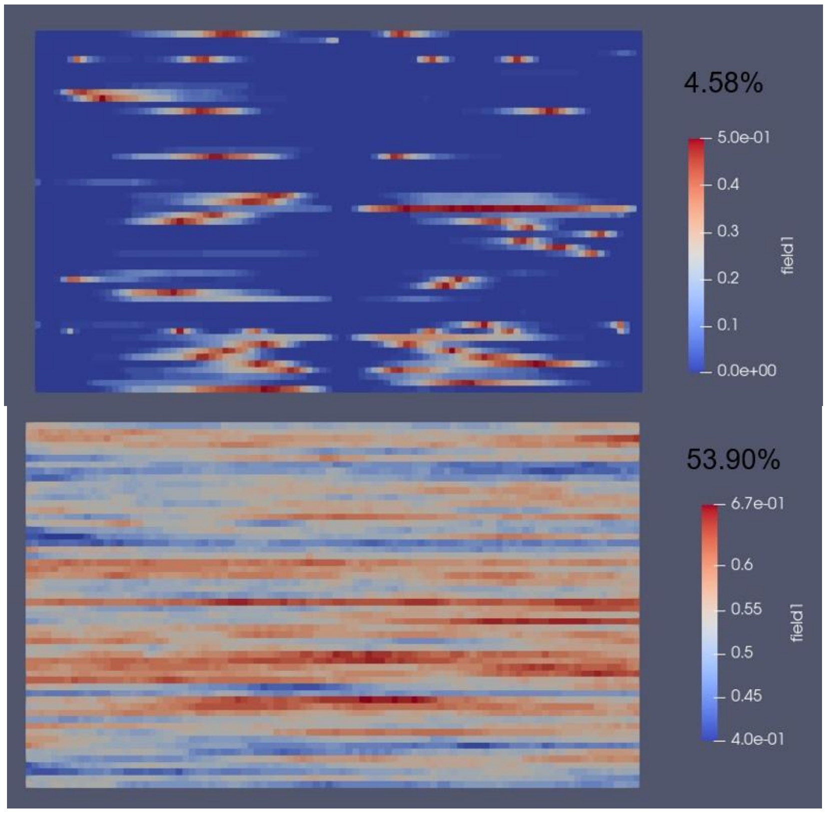


Fig. 36. The average uncertainty comparison between the conditional random field (4.58%, upper) and the unconditional random field (53.9%, lower) in the case study

5.4 Influence of one less CPT

Further study can be done within adjacent CPTs. Specifically, the central CPT (CPT1) is removed from input and other input keep the same. A few analyses are done subsequently to investigate in the influence in terms of: a) the uncertainty change after removing CPT1; b) the distribution of random values in the central column; c) the classifications of the simulated central column. The script to implement analysis b and c is shown in Appendix E.

a) The uncertainty change after removing CPT1

The most probable result of classifications and its uncertainty can be obtained by using the similar code as Appendix C. It can be observed in Fig. 38 that with only two known CPTs, the uncertainty of the classifications is 7.58% which is higher than the uncertainty of three known CPT case 4.58% shown in Fig. 35.

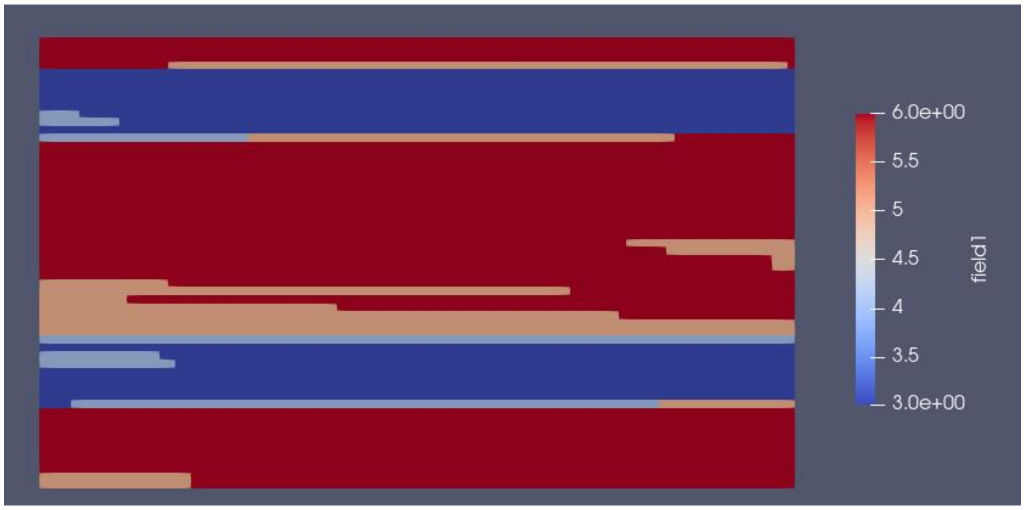


Fig. 37. The most probable result of classifications with CPT2 and CPT5

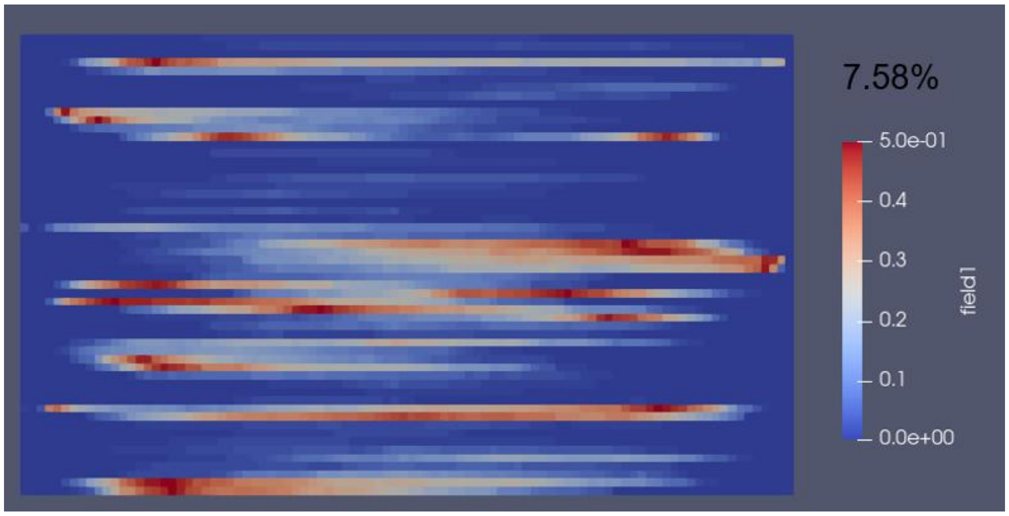


Fig. 38. The uncertainty of the most probable result of classifications with CPT2 and CPT5, with an average uncertainty of 7.58%

b) The distribution of random values in the central column

After running the conditional random field generator with 2 CPTs, the random values in the central column can be extracted over realizations. Based on those values, two post processes are taken. Both of these processes are applied to normalized friction ratio and cone resistance respectively. The first is to find the minimal and maximal value for each cell in the column over realizations. After doing this for all cells, the minimal and maximal boundary of CPT profiles can be obtained, which can be seen in Fig. 39 and Fig. 41. It's observed that the conducted CPT1 is within those two boundaries. The second process is the calculate the point statistics (mean and standard deviation) for each cell. Subsequently the representative point statistics can be calculated by taking the average of point statistics for each cell. It's worthy noticing that indeed the point statistics can vary a lot among different cells, but the coefficient of variance (COV, standard deviation / mean) doesn't vary a lot. So still it's meaningful to take such an average. Moreover, because the generation of the random values is subject to normal distribution, the probability density function can be drawn based on the point statistics and normal distribution, which is shown in Fig. 40 and Fig. 42.

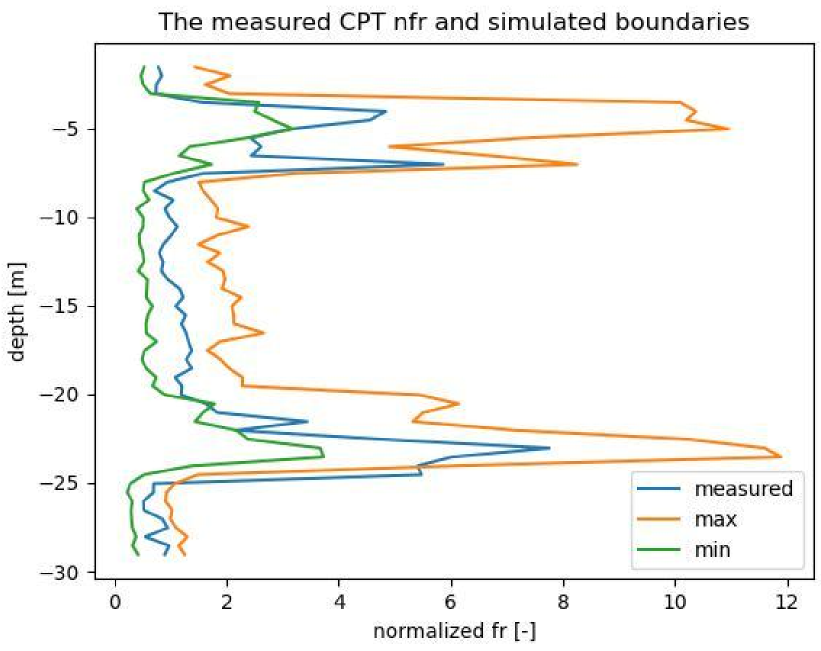


Fig. 39. The simulated boundaries of normalized friction ratio

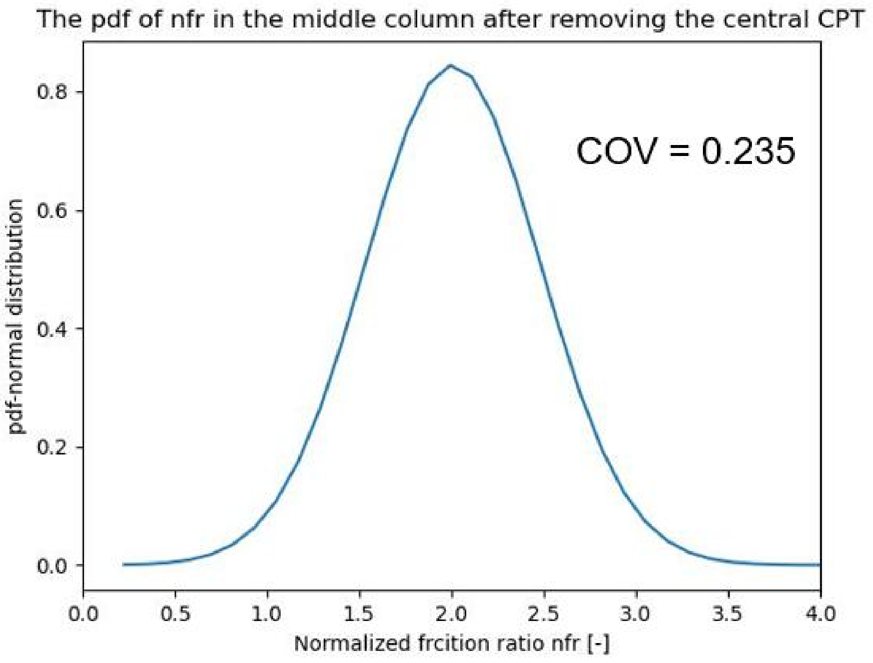


Fig. 40. The pdf of normalized friction ratio in the middle column

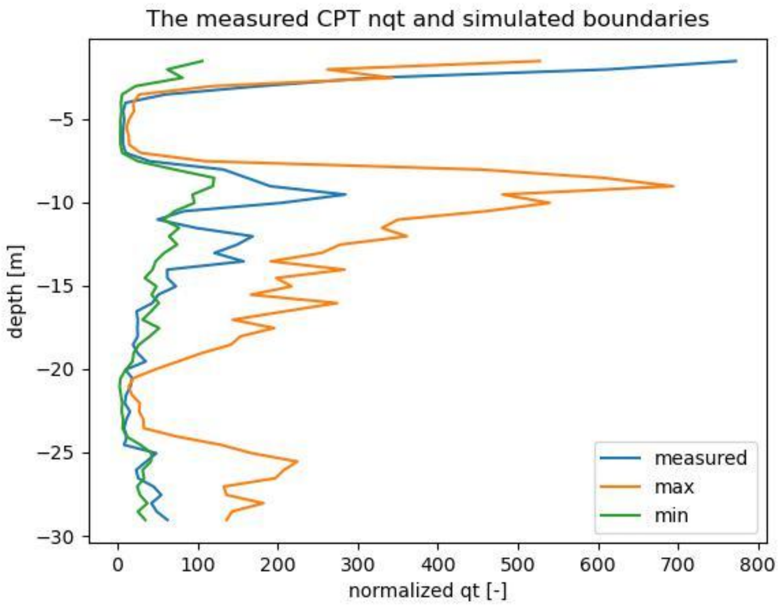


Fig. 41. The simulated boundaries of normalized cone resistance

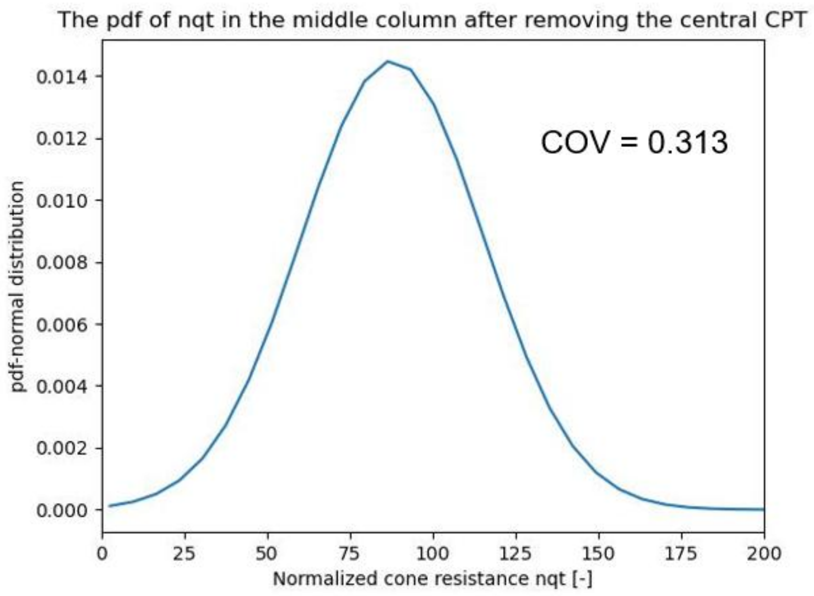


Fig. 42. The pdf of normalized cone resistance in the middle column

c) The classifications of the simulated central column

The random values generated with two CPTs input can be also applied to do classifications and the middle column is extracted to compare with CPT1 classifications. Fig. 43 illustrated the difference is 41.07%.

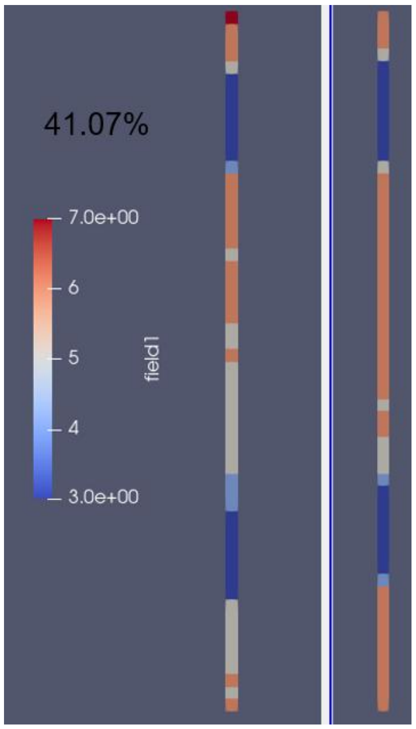


Fig. 43. The comparison between the most probable classifications in the middle column and CPT1

6. Limitation and optimization

6.1 Uncertainty in soil classification model

The proposed method to achieve probabilistic classification doesn't consider the uncertainty in classification model. Actually when using the CPT processing script to generate CPT profiles, it is in a deterministic way, which may ignore the uncertainty in CPT-based soil classification using Robertson's chart. Actually this uncertainty can be explicitly modeled in Bayesian approaches (Wang, Huang & Cao, 2013). Although this investigation is out of the scope of this study, there is an alternative and simper way to include the consideration of uncertainty in soil classification model proposed by Hu & Wang (2020). How to calculate soil behavior type index and how to use SBT index Ic to achieve soil classifications were proposed by Robertson & Wride (1998).

The function is defined as:

$$I_c = \sqrt{(3.47 - \log Q_t)^2 + (\log F_r + 1.22)^2} \tag{11}$$

where Qt and Fr are normalized cone resistance and friction ratio. Soil classifications based on SBT index Ic is shown in Table 3.

Table 3. Soil classifications based on SBT index, Ic (Robertson & Wride, 1998)

Range of SBT index Ic	SBT ID	SBT description
Ic < 1.31	7	Gravelly sand to dense sand
1.31 < Ic < 2.05	6	Sands: clean sand to silty sand
2.05 < Ic < 2.60	5	Sand mixtures: silty sand to sandy silt
2.60 < Ic < 2.95	4	Silt mixtures: clayey silt to silty clay
2.95 < Ic < 3.60	3	Clays: silty clay to clay
Ic > 3.60	2	Organic soil: peats

It can be seen that there are some boundaries for Ic respectively: 1.31, 2.05, 2.60, 2.95, 3.60. Means and standard deviations can be given to them in order to carry on the stochastic approach, which is shown in Table 4.

Table 4. Probabilistic boundaries of SBT index, Ic (Hu & Wang, 2020)

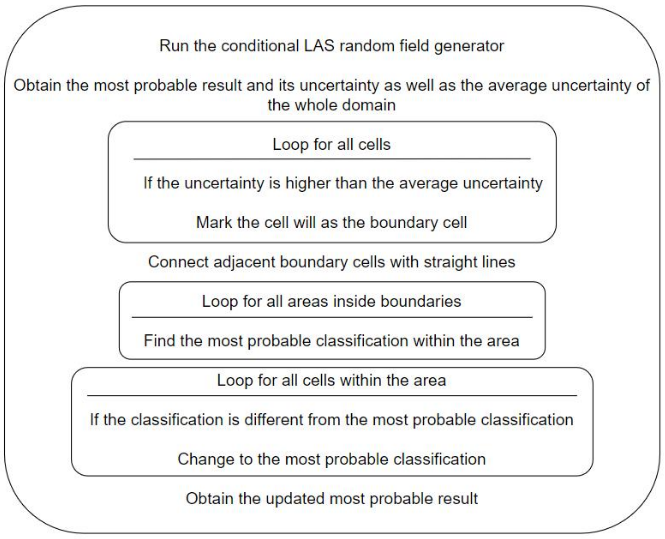
Statistics	B1	B2	B3	B4	B5
Mean	1.31	2.05	2.6	2.95	3.6
Standard deviation	0.1	0.1	0.05	0.05	0.1
Range	[1.01, 1.61]	[1.75, 2.35]	[2.45, 2.75]	[2.8, 3.1]	[3.3, 3.9]

As for how to use it specifically, for example, there are 100 realizations for normalized cone resistance and friction ratio, which later are applied to Eq. (11) to obtain Ic, from the proposed conditional random field generator based on LAS. At the mean time, there can be 100 sets of boundary values for Ic from a simple probabilistic model based on Table 4. Those 100 realizations are classified by corresponding 100 sets of boundaries. As a result, there are 100 probabilistic soil geographies which simultaneously incorporate uncertainties in the soil classification model as well as the uncertainties in the classification result.

6.2 Combination with boundary based model

The generator used in this study is basically category-based but actually it is more precise and realistic to carry on boundary analysis before the conditional random field category analysis. So the proposed method can be optimized by adding a boundary-based model in the random field. This can be achieved by a heuristic model (Xiao, Zhang, Li, & Li, 2017). First, the boundaries at conducted CPT positions can be determined, which will be used as known values for Kriging interpolation and unconditional random field simulation for boundaries in between adjacent CPTs. Notice Kriging interpolation cannot be solely applied because it actually has no random variations. Subsequently those simulated boundaries as well as the real boundaries measured by CPTs are used as constrains for the category based random field analysis. This is to say, with this method, the constrains of the conditional LAS random field generator come from both the boundaries and the conducted 1D CPT data.

There is another method according to the claim that the locations of high uncertainty are generally consistent with the underlying true soil layer boundaries (Hu & Wang, 2020). So a conceptual process is proposed in this study which is shown in Fig. 44.



```
Run the conditional LAS random field generator
Obtain the most probable result and its uncertainty as well as the average uncertainty of the whole domain
Loop for all cells
    If the uncertainty is higher than the average uncertainty
        Mark the cell will as the boundary cell
Connect adjacent boundary cells with straight lines
Loop for all areas inside boundaries
    Find the most probable classification within the area
    Loop for all cells within the area
        If the classification is different from the most probable classification
            Change to the most probable classification
Obtain the updated most probable result
```

Fig. 44. Pseudo code for adding boundary based model

Notice when connecting boundary cells, it may be assisted manually. The most probable result is only one image so it is feasible and in this way prior knowledge and profession knowledge can be utilized.

7. Conclusion

A random field generator based on Local Average Subdivision (LAS) is proposed in this study which aims to achieve probabilistic soil classifications and consider the uncertainty of the geological cross section. Moreover, it has been improved to a conditional random field generator by using Kriging interpolation which exerts more constrains to the random field. The unconditional and conditional generator are tested by an illustrative example and the conditional generator is applied to a case study with three real conducted CPTs. There are a few conclusions can be drawn:

a) It can be evidently seen that the uncertainty of the most probable result of classifications generated by the conditional random field generator (17.2%) is lower than that of the unconditional random field generator (48.2%). The uncertainty is truly reduced by improving the generator and the generation of the classifications are more realistic and meaningful.
b) With enough realizations, the classifications in the unconditional random field converge. In the most probable result, because Gaussian distribution is the only factor considered in the unconditional generation, all classifications become type 5 which is the mean used in the unconditional random field. It proves the sole application of LAS based generator is useless.
c) The uncertainty of the most probable result has been largely reduced to 4.58% in the case study, which may reflect that the proposed generator can be best applied in a large scale of fluctuation scenario.
d) The conditional random field generator can simulate the range of a CPT profile. For example, with two conducted CPTs respectively on the two sides of a problem domain, the maximal and minimal boundary of the CPT profile at the central position within this domain can be simulated. The real profile must be inside those two boundaries, which can be used to verify the reliability of conducted CPTs. In particular, the distribution of the profile of the normalized friction ratio has a COV of 0.235 and the distribution of the profile of the normalized cone resistance has a COV of 0.313.

Reference

Anagnostopoulos, A., Koukis, G., Sabatakakis, N., & Tsiambaos, G. (2003). Empirical correlations of soil parameters based on Cone Penetration Tests (CPT) for Greek soils. *Geotechnical And Geological Engineering*, *21*(4), 377-387. doi: 10.1023/b:gege.0000006064.47819.1a

Chessa, A. (2006). A Markov Chain Model for Subsurface Characterization: Theory and Applications. *Mathematical Geology*, *38*(4), 503-505. doi: 10.1007/s11004-006-9037-9

Cone penetration test - Wikipedia. (2022). Retrieved 19 September 2022, from https://en.wikipedia.org/wiki/Cone_penetration_test

Campanella, R.G., Gillespie, D., Robertson, P.K., 1982. Pore pressures during cone penetration testing. In: Proceedings of the 2nd European Symposium on Penetration Testing, vol. 2, pp. 507e512. ESOPT-2, Amsterdam.

Clayton, C. (2001). *Managing geotechnical risk*. London: Telford.

Griffiths, D. V., Huang, J., & Fenton, G. A. (2015). Probabilistic slope stability analysis using RFEM with non-stationary random fields. In Geotechnical safety and risk V (pp. 704-709). IOS Press.

Fenton, G., & Griffiths, D. (2007). Random Field Generation and the Local Average Subdivision Method. *CISM Courses And Lectures*, 201-223. doi: 10.1007/978-3-211-73366-0_9

Fenton, G., & Griffiths, D. (2008). *Risk assessment in geotechnical engineering*. Hoboken: Wiley.

Fenton, G., & Vanmarcke, E. (1990). Simulation of Random Fields via Local Average Subdivision. *Journal Of Engineering Mechanics*, *116*(8), 1733-1749. doi: 10.1061/(asce)0733-9399(1990)116:8(1733)

Hernández-Lemus, E. (2021). Random Fields in Physics, Biology and Data Science. *Frontiers In Physics*, *9*. doi: 10.3389/fphy.2021.641859

Hicks, M. (2009). *Risk and variability in geotechnical engineering*. London: Thomas Telford Ltd.

Hu, Y., & Wang, Y. (2020). Probabilistic soil classification and stratification in a vertical cross-section from limited cone penetration tests using random field and Monte Carlo simulation. *Computers And Geotechnics*, *124*, 103634. doi: 10.1016/j.compgeo.2020.103634

Jamshidi Chenari, R., & Kamyab Farahbakhsh, H. (2015). Generating non-stationary random fields of auto-correlated, normally distributed CPT profile by matrix decomposition method. *Georisk: Assessment And Management Of Risk For Engineered Systems And Geohazards*, *9*(2), 96-108. doi: 10.1080/17499518.2015.1033429

Liu, Y., Li, J., Sun, S., & Yu, B. (2019). Advances in Gaussian random field generation: a review. *Computational Geosciences*, *23*(5), 1011-1047. doi: 10.1007/s10596-019-09867-y

Lloret-Cabot, M., Hicks, M., & van den Eijnden, A. (2012). Investigation of the reduction in uncertainty due to soil variability when conditioning a random field using Kriging. *Géotechnique Letters*, *2*(3), 123-127. doi: 10.1680/geolett.12.00022

McGann, C., Bradley, B., Taylor, M., Wotherspoon, L., & Cubrinovski, M. (2015). Development of an empirical correlation for predicting shear wave velocity of Christchurch soils from cone penetration test data. *Soil Dynamics And Earthquake Engineering*, *75*, 66-75. doi: 10.1016/j.soildyn.2015.03.023

Montoya-Noguera, S., Zhao, T., Hu, Y., Wang, Y., & Phoon, K. (2019). Simulation of non-stationary non-Gaussian random fields from sparse measurements using Bayesian compressive sampling and Karhunen-Loève expansion. *Structural Safety*, *79*, 66-79. doi: 10.1016/j.strusafe.2019.03.006

Ramsey, N. (2010). Some issues related to applications of the CPT. 2nd International Symposium on Cone Penetration Testing, CA, USA

Robertson, P. (2010). Soil behaviour type from the CPT: an update. 2nd International Symposium on Cone Penetration Testing, CA, USA

Robertson, P. (2015). Soil behavior type using the DMT. Gregg Drilling & Testing Inc., Signal Hill, CA, USA.

Robertson, P. (1990). Soil classification using the cone penetration test. *Canadian Geotechnical Journal*, *27*(1), 151-158. doi: 10.1139/t90-014

Robertson, P., & Wride, C. (1998). Evaluating cyclic liquefaction potential using the cone penetration test. *Canadian Geotechnical Journal*, *35*(3), 442-459. doi: 10.1139/t98-017

Shi, C., & Wang, Y. (2021a). Development of Subsurface Geological Cross-Section from Limited Site-Specific Boreholes and Prior Geological Knowledge Using Iterative Convolution XGBoost. *Journal Of Geotechnical And Geoenvironmental Engineering*, *147*(9). doi: 10.1061/(asce)gt.1943-5606.0002583

Shi, C., & Wang, Y. (2021b). Nonparametric and data-driven interpolation of subsurface soil stratigraphy from limited data using multiple point statistics. *Canadian Geotechnical Journal*, *58*(2), 261-280. doi: 10.1139/cgj-2019-0843

Tillmann, A., Englert, A., Nyari, Z., Fejes, I., Vanderborght, J., & Vereecken, H. (2008). Characterization of subsoil heterogeneity, estimation of grain size distribution and hydraulic conductivity at the Krauthausen test site using Cone Penetration Test. *Journal Of Contaminant Hydrology*, *95*(1-2), 57-75. doi: 10.1016/j.jconhyd.2007.07.013

Wang, X., Wang, H., & Liang, R. (2017). A method for slope stability analysis considering subsurface stratigraphic uncertainty. *Landslides*, *15*(5), 925-936. doi: 10.1007/s10346-017-0925-5

Wang, Y., Huang, K., & Cao, Z. (2013). Probabilistic identification of underground soil stratification using cone penetration tests. *Canadian Geotechnical Journal*, *50*(7), 766-776. doi: 10.1139/cgj-2013-0004

Xiao, T., Zhang, L., Li, X., & Li, D. (2017). Probabilistic Stratification Modeling in Geotechnical Site Characterization. *ASCE-ASME Journal Of Risk And Uncertainty In Engineering Systems, Part A: Civil Engineering*, *3*(4). doi: 10.1061/ajrua6.0000924

Appendices: Python Codes

Appendix A: Main process

```python
import numpy as np
import matplotlib.pyplot as plt
import robertson_new as Frob_new
import Additional_subroutines as subs
from shutil import copyfile
import pandas as pd
from numpy.ma import sin, cos, exp, log
from scipy.stats import norm

'------Process the result file including fr and qc to classify--------'
#Read the fr and qc for all realizations in the feild.RES file
list1 = []
list2 = []
with open("field_co.RES", "r") as f:
    lines = f.readlines()
    for line in lines:
        a = line.split()
        x = a[0]
        y = a[1]
        list1.append(x)
        list2.append(y)

for i in range(len(list1)):
    list1[i] = float(list1[i])
for i in range(len(list2)):
    list2[i] = float(list2[i])

nfr = np.array(list1).reshape(-1, 1) #Generated log normalized friction ratio
nqt = np.array(list2).reshape(-1, 1) #Generated log normalized tip resistance


#Read the number and size of cells in both directions
lines = open('input.DAT').read().splitlines()
no_cell = lines[1].split()
no_cell_h = float(no_cell[0])
no_cell_v = float(no_cell[1])
cell_size = lines[2].split()
cell_size_vh = float(cell_size[1])
cell_oneR = int(no_cell_h * no_cell_v)
realizations = int(lines[0])

thickness = np.ones(nfr.shape) * cell_size_vh #Actually useless and it won't effect the result
matrix_generalized = np.hstack((thickness, nqt, nfr))

polygons = Frob_new.plot_Robertson(matrix_generalized)
matrix_final = Frob_new.deterministic(matrix_generalized, polygons)
classification_final = matrix_final[:, 1]
classification_eachR = np.array_split(classification_final, realizations)

'---------Create .VTK files which can be shown in Paraview---------------'
name = 1

for i in range(int(realizations)):
    i_str = str(name)
    filename = 'realization' + i_str + '.vtk'
    file = open(filename, 'a')
    copyfile('Example.vtk', filename)
    file.write('\r')

    for ii in classification_eachR[i]:
        file.write(str(ii)+'\r')
    file.close()
    name = name + 1
```

Appendix B: Determination of the number of realizations

```
import numpy as np
import Additional_subroutines as subs
import os
import robertson_new as Frob_new
import matplotlib.pyplot as plt

realizations = np.linspace(10, 100, 5)
realizations_additional = realizations + 10

unce_ave1 = np.zeros(realizations.shape)
unce_ave2 = np.zeros(realizations_additional.shape)

for ii, r in enumerate(realizations):
    # Create a temporary .dat file to store modified values
    lines = open('input.dat').read().splitlines()
    lines[0] = '%6.0f'%(r)
    open('input.dat', 'w').write('\n'.join(lines))
    #Run analysis
    os.system(r'@echo | ConditionalRandomFieldGenerator.exe')

    #open the result file and read the generated qc and rf
    list1 = []
    list2 = []
    with open("field_co.RES", "r") as f:
        lines = f.readlines()
        for line in lines:
            a = line.split()
            x = a[0]
            y = a[1]
            list1.append(x)
            list2.append(y)

    for i in range(len(list1)):
        list1[i] = float(list1[i])
    for i in range(len(list2)):
        list2[i] = float(list2[i])

    nfr = np.array(list1).reshape(-1, 1) #Generated log normalized friction ratio
    nqt = np.array(list2).reshape(-1, 1) #Generated log normalized tip resistance

    #Read the number and size of cells in both directions
    lines = open('input.DAT').read().splitlines()
    no_cell = lines[1].split()
    no_cell_h = float(no_cell[0])
    no_cell_v = float(no_cell[1])
    cell_size = lines[2].split()
    cell_size_vh = float(cell_size[1])

    thickness = np.ones(nfr.shape) * cell_size_vh #Actually useless and it won't effect the result
    matrix_generalized = np.hstack((thickness, nqt, nfr))

    polygons = Frob_new.plot_Robertson(matrix_generalized)
    matrix_final = Frob_new.deterministic(matrix_generalized, polygons)
    classification_final = matrix_final[:, 1]

    cell_oneR = int(no_cell_h * no_cell_v)
    r_changing = int(len(classification_final)/cell_oneR)
    unce_ave1[ii] = subs.FindAveUnce(classification_final, cell_oneR, r_changing)

```

```

for ii, r in enumerate(realizations_additional):
    # Create a temporary .dat file to store modified values
    lines = open('input.dat').read().splitlines()
    lines[0] = '%6.0f'%(r)
    open('input.dat', 'w').write('\n'.join(lines))
    #Run analysis
    os.system(r'@echo | ConditionalRandomFieldGenerator.exe')

    #open the result file and read the generated qc and rf
    list1 = []
    list2 = []
    with open("field_co.RES", "r") as f:
        lines = f.readlines()
        for line in lines:
            a = line.split()
            x = a[0]
            y = a[1]
            list1.append(x)
            list2.append(y)

    for i in range(len(list1)):
        list1[i] = float(list1[i])
    for i in range(len(list2)):
        list2[i] = float(list2[i])

    nfr = np.array(list1).reshape(-1, 1) #Generated log normalized friction ratio
    nqt = np.array(list2).reshape(-1, 1) #Generated log normalized tip resistance

    #Read the number and size of cells in both directions
    lines = open('input.DAT').read().splitlines()
    no_cell = lines[1].split()
    no_cell_h = float(no_cell[0])
    no_cell_v = float(no_cell[1])
    cell_size = lines[2].split()
    cell_size_vh = float(cell_size[1])

    thickness = np.ones(nfr.shape) * cell_size_vh #Actually useless and it won't effect the result
    matrix_generalized = np.hstack((thickness, nqt, nfr))

    polygons = Frob_new.plot_Robertson(matrix_generalized)
    matrix_final = Frob_new.deterministic(matrix_generalized, polygons)
    classification_final = matrix_final[:, 1]

    cell_oneR = int(no_cell_h * no_cell_v)
    r_changing = int(len(classification_final)/cell_oneR)
    unce_ave2[ii] = subs.FindAveUnce(classification_final, cell_oneR, r_changing)

Change_over_realizations = abs(unce_ave1 - unce_ave2) / unce_ave1
plt.figure()
plt.plot(realizations, Change_over_realizations)
plt.xlabel('The number of realizations')
plt.ylabel('Percent change in the uncertainty of the most probable result')
```

Appendix C1: Determination of the most probable result and its uncertainty

```
import Additional_subroutines as subs

'---------------Most probable result for classifcations and its uncertainty-------'
#To get the most probable result
most_probable = subs.MostProbableResult(classification_final, cell_oneR, realizations, 'example.vtk')

#To get the uncertainty of the most probable result
prob_most_probable = subs.FindUncertainty(classification_final, cell_oneR, realizations, 'example.vtk')

#To get the average uncertainty
ave_uncertainty = subs.FindAveUnce(classification_final, cell_oneR, realizations)
print(ave_uncertainty)
```

Appendix C2: Determination of the most probable result and its uncertainty (Additional subroutines)

```
import numpy as np
from collections import Counter
from shutil import copyfile

def Classification_SameCell(classification_allR, no_cell_oneR, no_realization):
    period = no_cell_oneR
    length = no_realization
    list_class_SameCell = []

    for i in range(period):
        position = i
        extract = np.zeros(length)
        for ii in range(length):
            extract[ii] = classification_allR[position]
            position += period

        list_class_SameCell.append(extract)

    return list_class_SameCell

#To get the probability of the most probable classification in each cell
def FindUncertainty(classification_allR, no_cell_oneR, no_realization, exampleVTK):
    class_SameCell = Classification_SameCell(classification_allR, no_cell_oneR, no_realization)
    prob_eachcell = []

    for i in range(len(class_SameCell)):
        a = Counter(class_SameCell[i]).most_common(1)[0][0]
        #print(a)
        n = 0

        for ii in range(len(class_SameCell[i])):
            if class_SameCell[i][ii] == a:
                n += 1
        prob = n/len(class_SameCell[i])

        prob_eachcell.append(prob)

    uncertainty_eachcell = 1 - np.array(prob_eachcell)

    filename = 'The uncertainty of the most probable result.vtk'

    file = open(filename, 'a')
    copyfile(exampleVTK, filename)
    file.write('\r')

    for ii in uncertainty_eachcell:
        file.write(str(ii)+'\r')
    file.close()
    return uncertainty_eachcell

def MostProbableResult(classification_allR, no_cell_oneR, no_realization, exampleVTK):
    class_SameCell = Classification_SameCell(classification_allR, no_cell_oneR, no_realization)
    mostprobable = np.zeros(no_cell_oneR)
    for i in range(len(class_SameCell)):
        a = Counter(class_SameCell[i]).most_common(1)[0][0]
        mostprobable[i] = a

    filename = 'Most probable result.vtk'

    file = open(filename, 'a')
    copyfile(exampleVTK, filename)
    file.write('\r')

    for ii in mostprobable:
        file.write(str(ii)+'\r')
    file.close()

    return mostprobable

def MostProbableResult_Array(classification_allR, no_cell_oneR, no_realization):
    class_SameCell = Classification_SameCell(classification_allR, no_cell_oneR, no_realization)
    mostprobable_array = np.zeros(no_cell_oneR)
    for i in range(len(class_SameCell)):
        a = Counter(class_SameCell[i]).most_common(1)[0][0]
        mostprobable_array[i] = a
    return mostprobable_array

def FindAveUnce(classification_allR, no_cell_oneR, no_realization):
    class_SameCell = Classification_SameCell(classification_allR, no_cell_oneR, no_realization)
    prob_eachcell = []

    for i in range(len(class_SameCell)):
        a = Counter(class_SameCell[i]).most_common(1)[0][0]
        #print(a)
        n = 0

        for ii in range(len(class_SameCell[i])):
            if class_SameCell[i][ii] == a:
                n += 1
        prob = n/len(class_SameCell[i])

        prob_eachcell.append(prob)

    uncertainty_eachcell = 1 - np.array(prob_eachcell)

    unce_ave = np.mean(uncertainty_eachcell)

    return unce_ave
```

Appendix D1: Determination of statistics of case study (mean and standard deviation)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from shutil import copyfile
from numpy import polyfit, poly1d
from numpy.ma import sin, cos, exp, log

'---------------Read the results after using the CPT reading scirpt--------'
df81 = pd.read_excel('CPT81_FindStatistics.xlsx')
depth81 = -df81.values[:, 0]
qt81 = df81.values[0:, 1]
fs81 = df81.values[:, 2]
sigv81 = df81.values[:, 5]
sigv_eff81 = df81.values[:, 6]
nqt81 = (qt81 - sigv81)/sigv_eff81
nfr81 = (fs81*100)/(qt81 - sigv81)
log_nqt81 = np.log(nqt81)
log_nfr81 = np.log(nfr81)

df82 = pd.read_excel('CPT82_FindStatistics.xlsx')
depth82 = -df82.values[:, 0]
qt82 = df82.values[0:, 1]
fs82 = df82.values[:, 2]
sigv82 = df82.values[:, 5]
sigv_eff82 = df82.values[:, 6]
nqt82 = (qt82 - sigv82)/sigv_eff82
nfr82 = (fs82*100)/(qt82 - sigv82)
log_nqt82 = np.log(nqt82)
log_nfr82 = np.log(nfr82)

df85 = pd.read_excel('CPT85_FindStatistics.xlsx')
depth85 = -df85.values[:, 0]
qt85 = df85.values[0:, 1]
fs85 = df85.values[:, 2]
sigv85 = df85.values[:, 5]
sigv_eff85 = df85.values[:, 6]
nqt85 = (qt85 - sigv85)/sigv_eff85
nfr85 = (fs85*100)/(qt85 - sigv85)
log_nqt85 = np.log(nqt85)
log_nfr85 = np.log(nfr85)

log_nfr81 = np.round(log_nfr81, 3)
log_nfr82 = np.round(log_nfr82, 3)
log_nfr85 = np.round(log_nfr85, 3)

log_nqt81 = np.round(log_nqt81, 3)
log_nqt82 = np.round(log_nqt82, 3)
log_nqt85 = np.round(log_nqt85, 3)

'--------To get the mean, std and scale of fluctuation for the input of random field-------------'
'To get the mean'
meanfr_log_norm = np.mean((log_nfr81, log_nfr82, log_nfr85))
meanqt_log_norm = np.mean((log_nqt81, log_nqt82, log_nqt85))

'To get the std'
stdfr_81 = np.std(log_nfr81)
stdfr_82 = np.std(log_nfr82)
```

```
stdfr_85 = np.std(log_nfr85)
stdfr = (stdfr_81 + stdfr_82 + stdfr_85) / 3

stdqt_81 = np.std(log_nqt81)
stdqt_82 = np.std(log_nqt82)
stdqt_85 = np.std(log_nqt85)
stdqt = (stdqt_81 + stdqt_82 + stdqt_85) / 3
```

Appendix D2: Determination of statistics of case study (scale of fluctuation)

```
'To get the scale of fluctuation'
#Remove some abnormal values
nqt81 = nqt81[9:-1]
nfr81 = nfr81[9:-1]
nqt82 = nqt82[9:-1]
nfr82 = nfr82[9:-1]
nqt85 = nqt85[9:-1]
nfr85 = nfr85[9:-1]

depth81 = depth81[9:-1]
depth82 = depth82[9:-1]
depth85 = depth85[9:-1]

'-----------------Get theta from normalized cone resistance-----------------'
##-------------For normalized qt-----------------------------
####Remove the trend first
##To get the regression function of the fr/qc
depth_total = np.hstack((depth81,depth82, depth85)) #the depth now is negative
nqt_total = np.hstack((nqt81, nqt82, nqt85))

plt.figure()
plt.plot(nqt_total , depth_total, 'rx', label = 'data')
plt.xlabel('fr [-]')
plt.ylabel('depth [m]')

coeff = polyfit(-depth_total, nqt_total, 1)
x = -depth81
y = coeff[0]*x + coeff[1]
plt.plot(y, -x, label = 'regression')
plt.legend()
plt.show()

nqt_regression = y

##Get the standard devaition from the detrended values
eps81 = nqt81 - nqt_regression
eps82 = nqt82 - nqt_regression
eps85 = nqt85 - nqt_regression
eps_total = np.hstack((eps81, eps82, eps85))
n_std_square = 0.0

plt.figure()
plt.plot(eps82, -depth82)

##The detrended nqt
nqt_detrended = (eps81 + eps82 + eps85)/3
print('mean(nqt_detrended):', np.mean(nqt_detrended))##Show the detrended values have a mean of 0

for eps in eps_total:
    n_std_square += (eps - np.mean(nqt_detrended))**2
std_detrended_nqt = (n_std_square/len(eps_total))**0.5

print('std_detrended_nqt:', std_detrended_nqt)

##Convert the detrended values to uniform distribution
nqt_unifrom = nqt_detrended/std_detrended_nqt
test_std = np.std(nqt_unifrom)
print('test_std:', test_std) ##Show that now nqt is uniform distribution and std is 1
## Plot the experimental correlation function
```

```
##There are two key points for the correlation function
##1. The minimal deltaz is chosen as the minimal thickness in the CPT reading script.
The following deltazs are the interger multiples of minimal deltaz
##2. In order to extract all the available pairs in the profile, the initila depth can be all depths in the profile.
standard_depth = -depth81
standard_depth = np.round(standard_depth, 2)
qt_toGetTheta = nqt_unifrom

list_rho_all = [1]
deltaz_index = np.linspace(0, 50, 51)

for i in deltaz_index:

    deltaz = (i + 1)*0.12
    ndeltaz = 0
    z_sum_deltaz = deltaz

    list_qt_z = []
    list_qt_z_sum_deltaz = []
    list_z = []
    list_z_sum_deltaz = []

    for ii in standard_depth:
        initial_depth = ii
        z_sum_deltaz = (initial_depth + deltaz).round(2)

        z_sum_deltaz_GETd = z_sum_deltaz

        for iii in range(300):
            if z_sum_deltaz_GETd <= max(standard_depth) and z_sum_deltaz_GETd not in list_z_sum_deltaz:
                ndeltaz += 1
                list_z_sum_deltaz.append(z_sum_deltaz_GETd.round(2))

            z_sum_deltaz_GETd += deltaz
            z_sum_deltaz_GETd = z_sum_deltaz_GETd.round(2)

    for iv in range(len(list_z_sum_deltaz)):

        #To get the index of z+deltaz and the corresponding fr
        index_z_sum_deltaz = np.where(standard_depth == list_z_sum_deltaz[iv])
        list_qt_z_sum_deltaz.append(qt_toGetTheta[index_z_sum_deltaz])

        #To get the index for the z and the corresponding fr
        array_z_sum_deltaz = np.array(list_z_sum_deltaz)
        array_z = array_z_sum_deltaz - deltaz
        array_z = np.round(array_z, 2)
        list_z = list(array_z)

        index_z = np.where(standard_depth == list_z[iv])
        list_qt_z.append(qt_toGetTheta[index_z])

    rho_deltaz = sum(np.array(list_qt_z) * np.array(list_qt_z_sum_deltaz))/ ndeltaz
    #rho_deltaz = rho_deltaz[0]

    list_rho_all.append(rho_deltaz)
array_deltaz_all = np.linspace(0, 51*0.12, 52)
list_deltaz_all = list(array_deltaz_all)

plt.figure()
plt.plot(list_deltaz_all, list_rho_all, linestyle = '--', label = 'rho(deltaz)')
plt.xlabel('deltaz [m]')
```

```python
plt.ylabel('experimental correlation function rho_qt [-]')
plt.title('Correlation function to find thetav')


#Fit a theoretical Markov correlation function
theta = np.array([0.1, 1/4, 1/2, 1, 2, 3, 4])
deltaz_markov = np.linspace(0, 6, 100)
for i in range(len(theta)):
    roum = exp(-2*deltaz_markov/theta[i])
    plt.plot(deltaz_markov, roum, label = f'theta = {theta[i]}')
plt.legend()
plt.show()
'---------------Do the same thing to friction ratio----------------'
depth_total = np.hstack((depth81,depth82, depth85)) #the depth now is negative
nfr_total = np.hstack((nfr81, nfr82, nfr85)) #the fr has been detrended

plt.figure()
plt.plot(nfr_total , depth_total, 'rx', label = 'data')
plt.xlabel('fr [-]')
plt.ylabel('depth [m]')

coeff = polyfit(-depth_total, nfr_total, 1)
x = -depth81
y = coeff[0]*x + coeff[1]
plt.plot(y, -x, label = 'regression')
plt.legend()
plt.show()

nfr_regression = y

##Get the standard devaition from the detrended values
eps81 = nfr81 - nfr_regression
eps82 = nfr82 - nfr_regression
eps85 = nfr85 - nfr_regression
eps_total = np.hstack((eps81, eps82, eps85))
n_std_square = 0.0

plt.figure()
plt.plot(eps82, -depth82)

##The detrended nfr
nfr_detrended = (eps81 + eps82 + eps85)/3
print('mean(nfr_detrended):', np.mean(nfr_detrended))##Show the detrended values have a mean of 1

for eps in eps_total:
    n_std_square += (eps - 0)**2
std_detrended_nfr = (n_std_square/len(eps_total))**0.5

print('std_detrended_nfr:', std_detrended_nfr)

##Convert the detrended values to uniform distribution
nfr_unifrom = nfr_detrended/std_detrended_nfr
test_std = np.std(nfr_unifrom)
print('test_std:', test_std)


# Plot the experimental correlation function
standard_depth = -depth81
standard_depth = np.round(standard_depth, 2)
fr_toGetTheta = nfr_unifrom

```

```python
list_rho_all = [1]
deltaz_index = np.linspace(0, 50, 51)

for i in deltaz_index:

    deltaz = (i + 1)*0.12
    ndeltaz = 0
    z_sum_deltaz = deltaz

    list_fr_z = []
    list_fr_z_sum_deltaz = []
    list_z = []
    list_z_sum_deltaz = []

    for ii in standard_depth:
        initial_depth = ii
        z_sum_deltaz = (initial_depth + deltaz).round(2)

        z_sum_deltaz_GETd = z_sum_deltaz

        for iii in range(300):
            if z_sum_deltaz_GETd <= max(standard_depth) and z_sum_deltaz_GETd not in list_z_sum_deltaz:
                ndeltaz += 1
                list_z_sum_deltaz.append(z_sum_deltaz_GETd.round(2))

            z_sum_deltaz_GETd += deltaz
            z_sum_deltaz_GETd = z_sum_deltaz_GETd.round(2)

    for iv in range(len(list_z_sum_deltaz)):

        #To get the index of z+deltaz and the corresponding fr
        index_z_sum_deltaz = np.where(standard_depth == list_z_sum_deltaz[iv])
        list_fr_z_sum_deltaz.append(fr_toGetTheta[index_z_sum_deltaz])

        #To get the index for the z and the corresponding fr
        array_z_sum_deltaz = np.array(list_z_sum_deltaz)
        array_z = array_z_sum_deltaz - deltaz
        array_z = np.round(array_z, 2)
        list_z = list(array_z)

        index_z = np.where(standard_depth == list_z[iv])
        list_fr_z.append(fr_toGetTheta[index_z])

    rho_deltaz = sum(np.array(list_fr_z) * np.array(list_fr_z_sum_deltaz))/ ndeltaz
    #rho_deltaz = rho_deltaz[0]

    list_rho_all.append(rho_deltaz)

array_deltaz_all = np.linspace(0, 51*0.12, 52)
list_deltaz_all = list(array_deltaz_all)

plt.figure()
plt.plot(list_deltaz_all, list_rho_all, linestyle = '--', label = 'rho(deltaz)')
plt.xlabel('deltaz [m]')
plt.ylabel('experimental correlation function rho_fr [-]')
plt.title('Correlation function to find thetav')


#Fit a theoretical Markov correlation function
theta = np.array([0.1, 1/4, 1/2, 1, 2, 3, 4])
deltaz_markov = np.linspace(0, 6, 100)
```

```
for i in range(len(theta)):
    roum = exp(-2*deltaz_markov/theta[i])
    plt.plot(deltaz_markov, roum, label = f'theta = {theta[i']')
plt.legend()
plt.show()
```

Appendix E: Investigation in one less CPT situation

```python
import numpy as np
import matplotlib.pyplot as plt
import robertson_new as Frob_new
import Additional_subroutines as subs
from shutil import copyfile
import pandas as pd
from numpy.ma import sin, cos, exp, log
from scipy.stats import norm

'------Process the result file including fr and qc to classify--------'
#Read the fr and qc for all realizations in the feild.RES file
list1 = []
list2 = []
with open("field_co.RES", "r") as f:
    lines = f.readlines()
    for line in lines:
        a = line.split()
        x = a[0]
        y = a[1]
        list1.append(x)
        list2.append(y)

for i in range(len(list1)):
    list1[i] = float(list1[i])
for i in range(len(list2)):
    list2[i] = float(list2[i])

nfr = np.array(list1).reshape(-1, 1) #Generated log normalized friction ratio
nqt = np.array(list2).reshape(-1, 1) #Generated log normalized tip resistance


#Read the number and size of cells in both directions
lines = open('input.DAT').read().splitlines()
no_cell = lines[1].split()
no_cell_h = float(no_cell[0])
no_cell_v = float(no_cell[1])
cell_size = lines[2].split()
cell_size_vh = float(cell_size[1])
cell_oneR = int(no_cell_h * no_cell_v)
realizations = int(lines[0])
'-----------Compare the middle column in terms of classifications--------'
##Extract the middle column and compare
mostprobable_array = subs.MostProbableResult_Array(classification_final, cell_oneR, realizations)
middlecolumn = []

middle_index = 48
for i in range(int(no_cell_v)):
    middlecolumn.append(mostprobable_array[int(middle_index)])
    middle_index += no_cell_h

##write in VTK file to visualize
filename = 'middlecolumn.vtk'
file = open(filename, 'a')
copyfile('ClassificationExample.vtk', filename)
file.write('\r')

for i in middlecolumn:
    file.write(str(i)+'\r')
file.close()
```

```

##read the actual CPT1 and compare the difference
real_CPT = []
with open('CPT81_classification.dat', 'r') as f:
    lines = f.readlines()
    for line in lines:
        CPT1_classification = float(line)
        real_CPT.append(CPT1_classification)

n = 0
for ii in range(len(real_CPT)):
    if middlecolumn[ii] != real_CPT[ii]:
        n += 1
Percent_diff = n/len(real_CPT)

'-----------Compare the middle column in terms of fr and qt--------'
##Get the log_nqt and log_nfr for each realization
log_nqt_eachR = np.array_split(nqt, realizations)
log_nfr_eachR = np.array_split(nfr, realizations)
##Get the middle column for each realization, stored in list
middlecolumn_lognqt = []
middlecolumn_lognfr = []

##First for log_nqt
for i in range(realizations):
    middlecolumn_qtcontainer = np.zeros(int(no_cell_v))
    middle_index = 48
    for ii in range(int(no_cell_v)):
        middlecolumn_qtcontainer[ii] = log_nqt_eachR[i][int(middle_index)]
        middle_index += no_cell_h
    middlecolumn_lognqt.append(middlecolumn_qtcontainer)

##Second for log_nfr
for i in range(realizations):
    middlecolumn_frcontainer = np.zeros(int(no_cell_v))
    middle_index = 48
    for ii in range(int(no_cell_v)):
        middlecolumn_frcontainer[ii] = log_nfr_eachR[i][int(middle_index)]
        middle_index += no_cell_h
    middlecolumn_lognfr.append(middlecolumn_frcontainer)

##Get normalized qt and normalized fr at the same cell for each realization
middleC_nqt_eachR = []
middleC_nfr_eachR = []

for i in range(int(no_cell_v)):
    nqtcontainer_realizations = np.zeros(realizations)
    for ii in range(realizations):
        nqtcontainer_realizations[ii] = exp(middlecolumn_lognqt[ii][i]) ##Notice the expotential Func
    middleC_nqt_eachR.append(nqtcontainer_realizations)

for i in range(int(no_cell_v)):
    nfrcontainer_realizations = np.zeros(realizations)
    for ii in range(realizations):
        nfrcontainer_realizations[ii] = exp(middlecolumn_lognfr[ii][i]) ##Notice the expotential Func
    middleC_nfr_eachR.append(nfrcontainer_realizations)

##mean for nqt and nfr
mean_eachC_nqt = []
for i in range(int(no_cell_v)):
    mean_tem = np.mean(middleC_nqt_eachR[i])
    mean_eachC_nqt.append(mean_tem)

mean_eachC_nfr = []
for i in range(int(no_cell_v)):
    mean_tem = np.mean(middleC_nfr_eachR[i])
    mean_eachC_nfr.append(mean_tem)

#std for nqt and nfr
std_eachC_nqt = []
for i in range(int(no_cell_v)):
    std_tem = np.std(middleC_nqt_eachR[i])
    std_eachC_nqt.append(std_tem)

std_eachC_nfr = []
for i in range(int(no_cell_v)):
    std_tem = np.std(middleC_nfr_eachR[i])
    std_eachC_nfr.append(std_tem)

##The final version of the mean and std are the average values over depth
final_mean_middleC_nqt = sum(mean_eachC_nqt)/(len(mean_eachC_nqt))
final_mean_middleC_nfr = sum(mean_eachC_nfr)/(len(mean_eachC_nfr))

final_std_middleC_nqt = sum(std_eachC_nqt)/(len(std_eachC_nqt))
final_std_middleC_nfr = sum(std_eachC_nfr)/(len(std_eachC_nfr))


##--------Plot the simulated range and measured CPT---------##

##Get the max and min values to get the range plot
max_middleC_nqt = []
for i in range(int(no_cell_v)):
    max_tem = np.max(middleC_nqt_eachR[i])
    max_middleC_nqt.append(max_tem)

max_middleC_nfr = []
for i in range(int(no_cell_v)):
    max_tem = np.max(middleC_nfr_eachR[i])
    max_middleC_nfr.append(max_tem)


min_middleC_nqt = []
for i in range(int(no_cell_v)):
    min_tem = np.min(middleC_nqt_eachR[i])
    min_middleC_nqt.append(min_tem)

min_middleC_nfr = []
for i in range(int(no_cell_v)):
    min_tem = np.min(middleC_nfr_eachR[i])
    min_middleC_nfr.append(min_tem)

###########The pdf based on the above mean and standard deviation########
plt.figure()
x_axis_nqt = np.linspace(min(min_middleC_nqt), max(max_middleC_nqt), 100)
plt.plot(x_axis_nqt, norm.pdf(x_axis_nqt, final_mean_middleC_nqt, final_std_middleC_nqt))
plt.xlim(0, 200)
plt.xlabel('Normalized cone resistance nqt [-]')
plt.ylabel('pdf-normal distribution')
plt.title('The pdf of nqt in the middle column after removing the central CPT')

plt.figure()
x_axis_nfr = np.linspace(min(min_middleC_nfr), max(max_middleC_nfr), 100)
```

```python
plt.plot(x_axis_nfr, norm.pdf(x_axis_nfr, final_mean_middleC_nfr, final_std_middleC_nfr))
plt.xlim(0, 4)
plt.xlabel('Normalized frcition ratio nfr [-]')
plt.ylabel('pdf-normal distribution')
plt.title('The pdf of nfr in the middle column after removing the central CPT')

##Read CPT81 which is the conducted CPT in the middle and compare
df81 = pd.read_excel('CPT81.xlsx')
depth81 = -df81.values[:, 0]
qt81 = df81.values[0:, 1]
fs81 = df81.values[:, 2]
sigv81 = df81.values[:, 5]
sigv_eff81 = df81.values[:, 6]
nqt81 = (qt81 - sigv81)/sigv_eff81
nfr81 = (fs81*100)/(qt81 - sigv81)
log_nqt81 = np.log(nqt81)
log_nfr81 = np.log(nfr81)
##Remove some abnormal values
# nqt81 = nqt81[9:-1]
# nfr81 = nfr81[9:-1]
# depth81 = depth81[9:-1]

###########The range of simulated results and real condition########
plt.figure()
plt.plot(nqt81, depth81, label = 'measured')
plt.plot(max_middleC_nqt, depth81, label = 'max')
plt.plot(min_middleC_nqt, depth81, label = 'min')
plt.xlabel('normalized qt [-]')
plt.ylabel('depth [m]')
plt.legend()
plt.title('The measured CPT nqt and simulated boundaries')

plt.figure()
plt.plot(nfr81, depth81, label = 'measured')
plt.plot(max_middleC_nfr, depth81, label = 'max')
plt.plot(min_middleC_nfr, depth81, label = 'min')
plt.xlabel('normalized fr [-]')
plt.ylabel('depth [m]')
plt.legend()
plt.title('The measured CPT nfr and simulated boundaries')
```