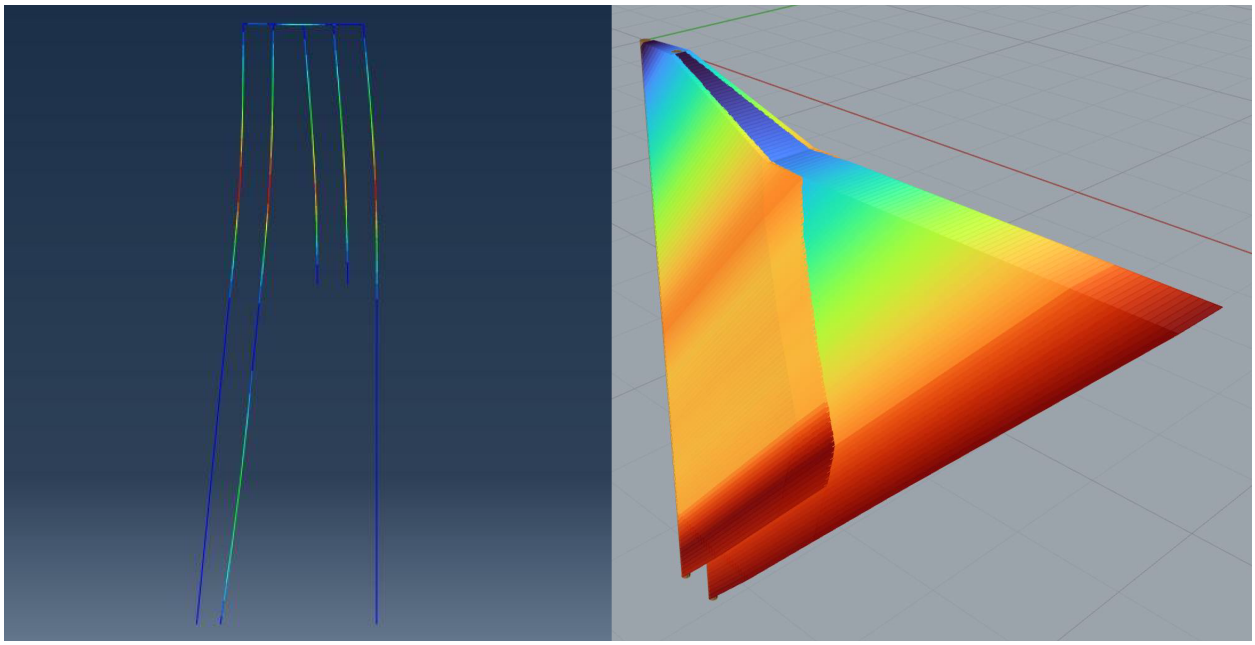
CHALMERS
UNIVERSITY OF TECHNOLOGY



Analysis of historical quay wall structures

A Sensitivity Study to Develop a Method of Analysing Existing Structures

Master's Thesis in the Master's Programme Structural Engineering and Building Technology

Juan José Arriola Meza
Tobias Stridh

DEPARTMENT OF ARCHITECTURE AND CIVIL ENGINEERING
DIVISION OF STRUCTURAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Master's thesis ACEX30
Gothenburg, Sweden 2025

MASTER'S THESIS 2025

Analysis of historical quay wall structures

A sensitivity study to develop a method of analysing existing structures

Juan José Arriola Meza
Tobias Stridh

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Architecture and Civil Engineering
Division of Structural engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025

Analysis of historical quay wall structures
A sensitivity study to develop a method of analysing existing structures
Juan J. Arriola
Tobias Stridh

Supervisor: Jelke Dijkstra, Department of Architecture and Civil Engineering
Co-Supervisor: Joosef Leppänen, Department of Architecture and Civil Engineering
Examiner: Jelke Djikstra, Department of Architecture and Civil Engineering

Master's Thesis 2025
Department of Architecture and Civil Engineering
Division of Structural Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover:
Left: A picture from the Abaqus model, showing the stress distribution, Right: A picture of contour volumes from two reduced piles, utilising the wedge model.
Department of Architecture and Civil Engineering
Gothenburg, Sweden, 2025

Analysis of historical quay wall structures
A sensitivity study to develop a method of analysing existing structures
Master's Thesis in the Master's Programme Structural Engineering and Building Technology
Juan José Arriola Meza, Tobias Stridh
Department of Architecture and Civil Engineering
Chalmers University of Technology

Abstract

This thesis investigates methods for assessing the structural capacity of historic quay walls in central Gothenburg, many of which are in need of renovation. A theoretical study of typical failure mechanisms was carried out, and analysis methods previously applied to similar quay walls in Amsterdam were studied to create a finite element (FE) model in Abaqus. This model was validated against existing Dutch models. Additionally, a reduction model was developed in Grasshopper to account for geometric constraints on soil capacity. Parametric studies were conducted to evaluate key factors influencing structural integrity, resulting in a practical tool for assessing historical quay wall structures.

For validation, the developed model was compared with results from the Dutch case. The simulations included various combinations: assessing the impact of the reduction factor, comparing drained and undrained conditions in the wedge model, and evaluating the results against hand calculations.

In the Gothenburg case study, two models were developed to analyse the effect of different structural layouts. The FE modelling considered different conditions, resulting in 96 simulations providing data to conduct a sensitivity study. Approximately $10\,\%$ of these combinations were found to be in a critical state.

The study identified that the front pile and pile diameter, in combination with at-rest earth pressure, where the most critical cases. The status of the filling, in the top soil layers beneath the timber deck, was the most critical condition to determine. These requires consideration when assessing the structural integrity of historical quay walls.

It was proven that a parametric finite element model, implementing a soil wedge reduction model with visual programming, can be used to conduct a sensitivity study on historical quay walls.

Keywords: Quay walls, Soil-pile interaction, Parametric modelling, FE-modelling, Abaqus, Historic structures, Pile-foundations, Grasshopper.

Analys av historiska kajmurskonstruktioner
En känslighetsstudie för att utveckla en metod för att analysera befintliga konstruktioner
Examensarbete inom Masterprogrammet Konstruktionsteknik och Byggnadsteknologi
Juan José Arriola Meza, Tobias Stridh
Institutionen för Arkitektur och Samhällsbyggnadsteknik
Chalmers Tekniska Högskola

Sammanfattning

Denna uppsats undersöker metoder för att bedöma bärförmågan hos historiska kajmurskonstruktioner i centrala Göteborg, där många är i behov av renovering. En teoretisk studie av typiska brottmekanismer genomfördes och analysmetoder, som tidigare tillämpats på liknande kajmurar i Amsterdam, studerades för att skapa en finita elementmodel (FE-modell) i Abaqus. Denna modell validerades mot befintliga holländska modeller. Dessutom utvecklades en modell i Grasshopper för att ta hänsyn till hur jordens kapacitet reduceras till följd av geometriska begränsningar. Parametriska studier genomfördes för att utvärdera nyckelfaktorer som påverkar den strukturella integriteten, vilket resulterade i ett praktiskt verktyg för att bedöma historiska kajmurskonstruktioner.

För valideringen jämfördes den utvecklade modellen med resultat från det holländska fallet. Simuleringarna inkluderade olika kombinationer: bedömning av reduktionsfaktorns påverkan, jämförelse mellan dränerade och odränerade förhållanden, och utvärdering av resultaten mot handberäkningar.

I Göteborgfallstudien utvecklades två modeller för att analysera effekten av olika konstruktiva utformningar. FE-modelleringen beaktade olika förhållanden vilket resulterade i 96 simuleringar som gav data för att genomföra känslighetsstudien. Ungefär $10\,\%$ av dessa kombinationer befanns vara i ett kritiskt tillstånd.

Studien identifierade att den främre pålen och påldiametern, i kombination med vilojordtryck, var de mest kritiska fallen. Statusen på fyllnaden i de översta jordlagren under trädäcket var det mest kritiska tillståndet att fastställa. Dessa parametrar måste beaktas vid bedömning av den strukturella integriteten hos historiska kajmurar.

Det bevisades att en parametrisk finit elementmodell, som implementerar en reduktionsmodell med visuell programmering, kan användas för att genomföra en känslighetsstudie på historiska kajmurskonstruktioner.

Nyckelord: Kajmurar, Interaktion jord-påle, Parametrisk modellering, FE-modellering, Abaqus, Historiska byggnadsverk, Pålgrundläggning, Grasshopper.

Contents

Acknowledgements

The work was carried out as part of the master's program at Chalmers University of Technology, within the Department of Structural Engineering, and in collaboration with the engineering firm Sweco in Gothenburg.

We would like to express our sincere gratitude to our academic supervisors, Joosef Leppänen and Jelke Dijkstra, for their invaluable guidance during these couple of months. An extra thank to Jelke Dijkstra for also being our examinator.

We would like to acknowledge the research conducted at TU Delft, which provided the foundation for this thesis. A special thanks to Dr Mandy Korff at Delft University of Technology, for agreeing to meet with us and providing your valuable expertise on the topic of historical quay walls.

To the engineers at Sweco, Gothenburg, we are deeply grateful for your support throughout this project. In particular, we extend heartfelt thanks to the departments *Anläggningskonstruktion* and *Geoteknik*, without whose expertise and assistance, this thesis would not have reached its current form.

Gothenburg, June, 2025
Juan Jose Arriola Meza,
Tobias Stridh

List of Figures

List of Tables

1 Introduction

1.1 Background

Quay walls have existed ever since man started travelling on water and the structural design have varied over time (Sharma et al., 2023). In Amsterdam, the Netherlands, a type of quay wall structure with a Masonry superstructure started being erected during the 17th century.

In Gothenburg, masonry quay walls support the roads that are situated along the channels in the city. The capacity and lifespan of the structure, including the pile foundation, are difficult to assess and because of this they are often replaced by new concrete retaining walls that have a known safety factor. There is a gap in knowledge regarding how the masonry superstructure behaves when loaded by traffic loading, which is the case for the walls in Gothenburg that are loaded by trams, buses, and cars in the central parts of the city.

Whereas, the lifespan of the concrete walls is limited to 120 years (L100), existing walls have been around for a significantly longer time and are still standing today. However, there is a limited knowledge regarding how long their function will remain intact. Inspections have concluded that the historic walls of the harbour are in poor shape and need renovation (Göteborg stad, 2025).

This thesis will analyse the capacity of these walls, focusing on the wall and the pile foundation below. The focus is on developing a method of assessing the capacity/stability and evaluating the most important factors affecting the capacity/stability. This was done by first benchmarking the method against a similar case in Amsterdam in the Netherlands, where a collapse of an existing historical quay wall was observed and an analytical model was created to evaluate the failure mechanism of the wall.

1.2 Aim and objective

The aim of this thesis is to find the critical parameters affecting the capacity/stability of a typical historic quay on timber piles found in Gothenburg and to find a method to assess their structural integrity.

The following objectives are identified:

- Find the governing failure mechanisms for quay walls near the Gothenburg canal.
- Investigate which parameters are critical to the integrity of the structure.
- By incorporating known theory and previous research on similar conditions, develop hand-calculations and numerical models and benchmark those with a reference case from the Netherlands, where a new analytical model was presented.
- Apply the method in one case study of the Gothenburg canal retaining walls to assess the capacity / stability of existing walls including their pile-foundation.

1.3 Limitations and Delimitations

The following limitations are made for this thesis:

- Only characteristic parameters of soil, loads and material properties will be used.
- One reference case is studied and the model is only applied in one case study in

Gothenburg with one set of geometry and site conditions.

The following delimitations are made for this thesis:

- The ultimate limit state will be analysed and no major focus is put on the serviceability limit state.
- Results are evaluated for the foundation only and no major focus is put on the masonry structure, since it was found in the study by Korff et al. (2022) that this is not the main failure mechanism.
- Dynamic effects are not considered.

1.4 Method

The thesis starts with a literature review on the governing failure mechanisms of the type of quay retaining walls like those found along the Gothenburg canals. This is followed by a short summary of available finite-element (FE) software, Abaqus, FEM-Design and Plaxis, to understand how they work and how they can be applicable for our case when modelling piles and masonry type structures.

The information obtained is used to create a calculation method to model the structural behaviour of the retaining walls. To validate the method it is applied in an identified reference case, in which a newly developed analytical model was utilised to analyse the capacity on a quay structure in Amsterdam, the Netherlands, with a structure similar to those along the Gothenburg canals (Korff et al., 2022). Based on comparisons of the results, between our model and the reference case, the method is iteratively refined until the results align with the presented data from the reference case.

Finally, the model is implemented to investigate the capacity of the quay retaining walls along the Gothenburg canals and a parametric study is conducted.

1.5 Social, ethical, and ecological aspects

The following socio-economic and ecological benefits may be gained if a calculation method is found that allows designers to conclude that the structural integrity of existing structures is sufficient, to prevent the structures from being replaced.

- No need to replace the existing walls. Replacing the existing walls will lead to a demand of new materials, increasing the environmental impact.
- The execution of construction works would lead to a substantial traffic impact which will affect commuters and visitors to the city centre. This, on the other hand, needs to be balanced with the impact of maintenance work of the walls.
- The existing walls are an important historical aspect for the city of Gothenburg whereas replacing them removes an important piece of history.

2 Theoretical Framework for Quay Wall Analysis

This chapter will elaborate on the theory behind the historical quay walls that is necessary to reach the objectives of this thesis. Geometry and characteristics for gravitation retaining quay wall structures will be investigated and calculation methods applicable for analysing these type of structures will be studied.

2.1 Geometry and characteristics of gravitational quay walls

2.1.1 Background

Since the stratigraphy of Amsterdam generally consists of soft clay; pile foundations are necessary to carry the superstructure (Sharma et al., 2023). An example is shown in Figure 2.1 where photos of the superstructure and pile-foundation are presented along with a 2D-section showing the structural representation.

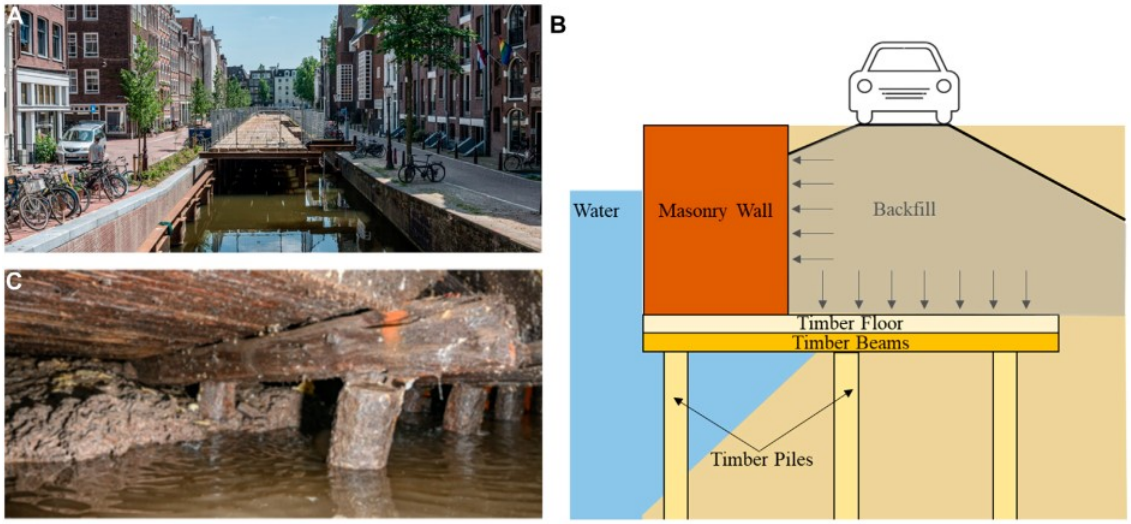


Figure 2.1: An example of gravitational quay wall structure in Amsterdam, the Netherlands A) Superstructure, B) Structural 2D representation, C) Pile foundation (Sharma et al., 2023).

The city of Amsterdam has 600 km of historic quay wall structures (Sharma et al., 2023). Typically in major European cities the quay walls are loaded by traffic coming from pedestrians and vehicles which creates vertical- and horizontal loads affecting the structure as indicated in Figure 2.1, C.

Information regarding the geometry and strength characteristics of the structural elements is difficult to obtain (Luongo et al., 2025). The documentation of the walls is limited and proper inspections of the structure are difficult and expensive to conduct. Guidelines for how to properly assess the capacity and structural integrity are also challenging to find. Several historical quay walls have collapsed in Amsterdam in recent years whereas the need for maintenance work is apparent. Out of the 200 km walls in Amsterdam where a risk of collapse have been observed, more than 50% does not comply with the requirements for safety and durability.

According to (Erling Reinius, 1963), it was earlier common to build quay retaining wall

structures as gravitational walls with masonry of stone or bricks. In modern days, this has been replaced with other building materials, such as reinforced concrete, due to lower costs.

Gothenburg have similar gravitational walls as in the Netherlands, partly built by the Dutch in the 17th century (Göteborg stad, 2025). The structures are in bad shape and are in need of renovations. One such wall, found at the central station in the city of Gothenburg, is shown in Figure 2.2 where the photo was taken more than 100 years ago.

Figure 2.2: An example of a gravitational retaining wall found in Gothenburg, Sweden close to the Centralstation taken around 1900 (Vårt Göteborg, Göteborgs stads tidning, 2023).

The walls have been around for a long time, still serving their function. The same wall as shown in Figure 2.2 can be seen in Figure 2.3, but as it looks in modern days with trams passing along the walls. Supported by the structure is the tram-station named *Centralstationen* which is located at *Drottningtorget*.

Figure 2.3: The quay wall around the central station as it looks in 2025 (Göteborg stad, 2025).

In some parts of the city, for example at *Stora hamnkanalen* in Gothenburg, Sweden, the older walls, originally made out of timber, got replaced somewhere around the 19th century with new retaining wall structures with a superstructure in stone masonry (Exploateringsförvaltningen, Göteborgs stad, 2024). Information regarding the characteristics of these walls are hard to come by but an example in Figure 2.4 shows an assumed section from an old

drawing of the quay wall at the location called *Packhuskajen*. The figure is presented in a geotechnical study for a new pedestrian bridge connecting the island of *Hisingen* with the central part of Gothenburg.

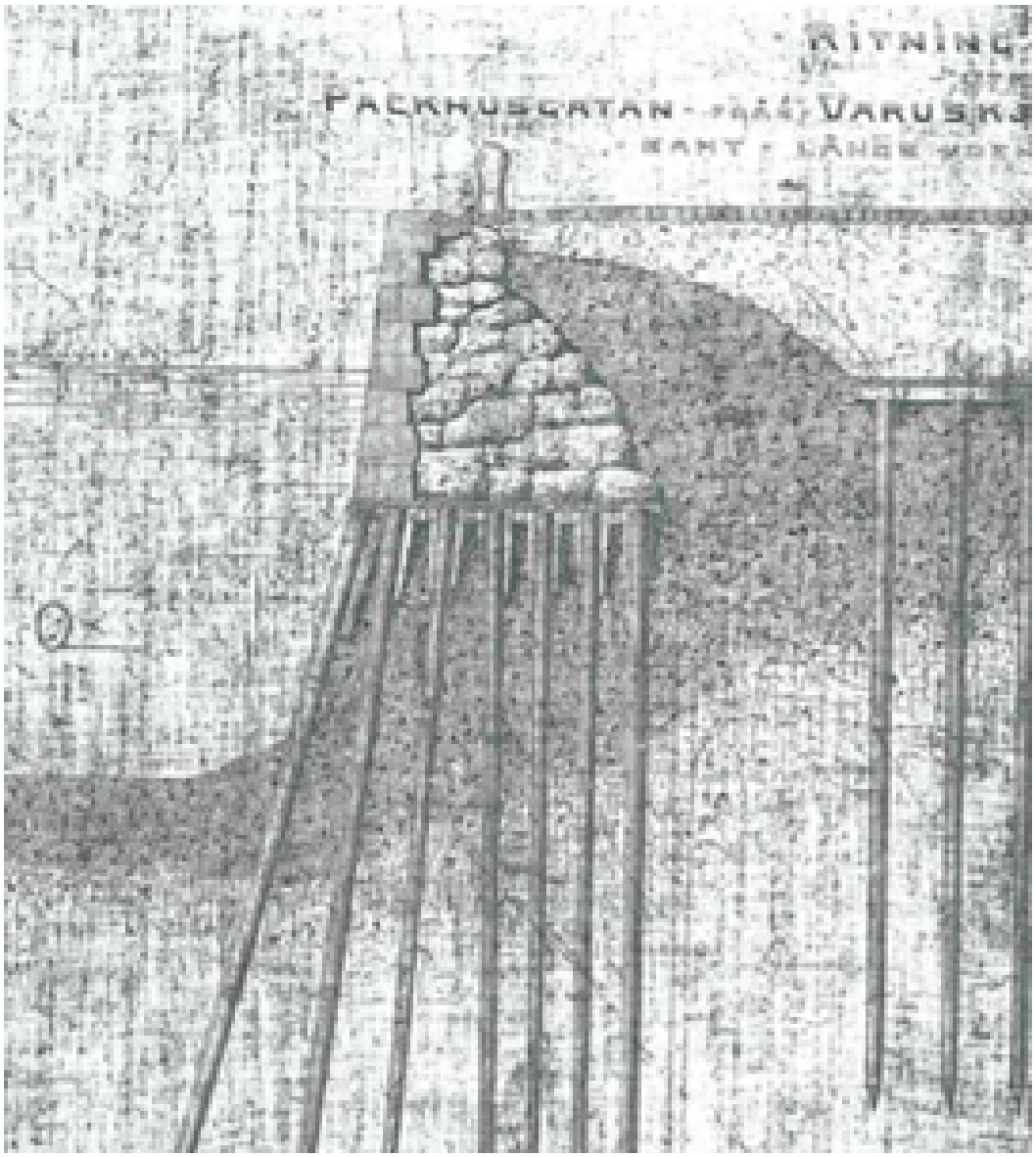


Figure 2.4: Assumed cross-section of an old gravitational wall with Masonry superstructure at *Packhuskajen*, Sweden (Exploateringsförvaltningen, Göteborgs stad, 2024).

This specific wall is founded on timber piles and the superstructure, a stone masonry wall, is supported on a timber deck. The type of engineering solution is consistent to the example from Amsterdam, compare Figure 2.1, with a major difference regarding the amount of piles and size of the superstructure itself. Due to excessive settlements over time the quay wall was reinforced in 1997 (Exploateringsförvaltningen, Göteborgs stad, 2024) and a more detailed drawing, containing geometrical measurements, was presented as seen in Figure 2.5.

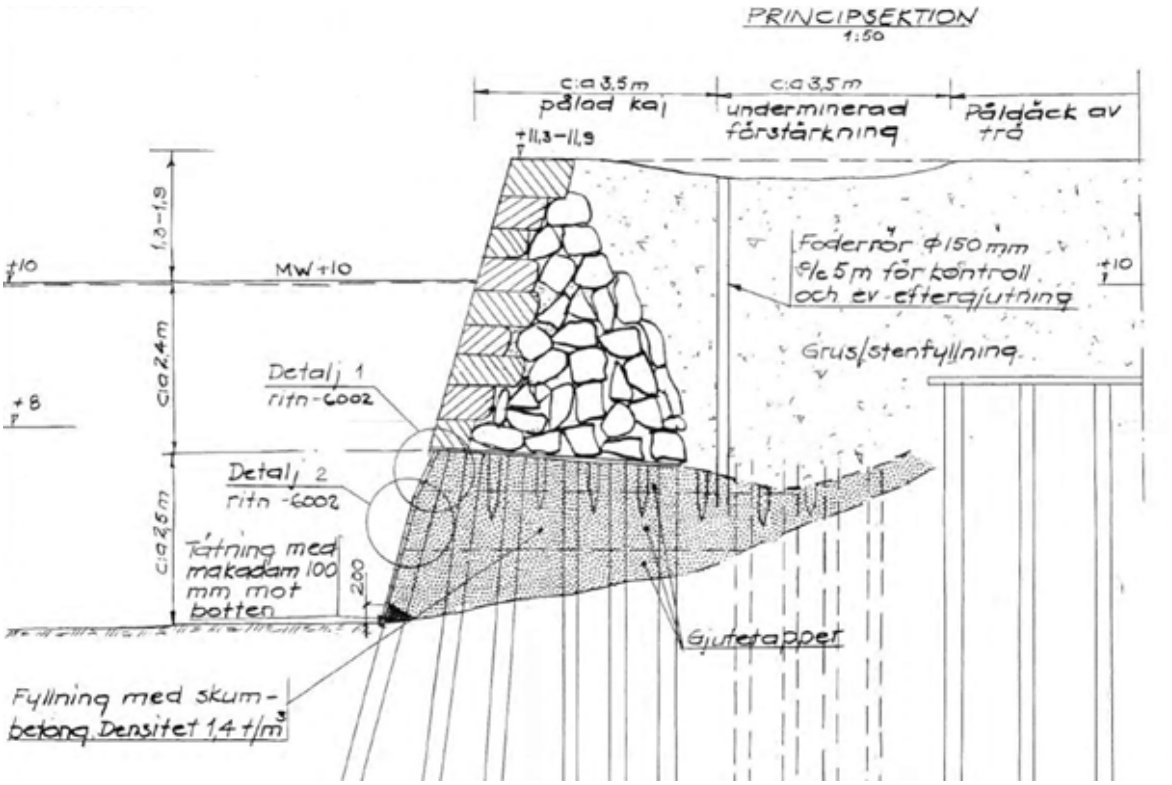


Figure 2.5: Old wall at *Packhuskajen*, Sweden, with more detailed measurements (Exploateringsförvaltningen, Göteborgs stad, 2024).

The wall at *Packhuskajen* was demolished and a new reinforced concrete structure, founded on steel piles, was built in it's place around 2020. A picture from the construction of this wall is shown in Figure 2.6.

Figure 2.6: Picture from the erection of the new retaining wall structure at *Packhuskajen* (Sweco AB, 2020).

The concrete slab was cast at the bottom of the excavation which was supported by two sheet-pile walls, one as protection from the water and one supporting the soil. The piles are visible and the falsework is currently being built for the wall-segment. This construction method is complex where aspects such as limited workspace on land and other interests must be considered (Luongo et al., 2025).

A positive aspect in favour of replacing the walls is that the need for correct data monitoring (for example settlements and horizontal movements) and evaluation of results from structural inspections decrease when a new, fully known structure, is built (Goldbohm et al., 2018).

2.1.2 Piles

Timber piles for marine structures can be used when the pile is completely covered in soil or when the pile head, of a pile standing free in fresh water, is located below the low mean water level (Erling Reinius, 1963). Otherwise, the piles risk damage caused by rot. When piles are installed in salt water with a high saline content they risk being attacked by pile worm which causes decomposition of the timber. This type of damage can be prevented by ensuring that the piles are completely covered in soil, creating a dark environment and reduces the amount of plankton in the water. This environment is none suitable for the worms which prevents these attacks from occurring.

In Amsterdam, the Netherlands, Luongo et al. (2025) mentions that deterioration from bacteria, fungi, water level variations, and more can cause a decrease in their load bearing capacity. This effects lead to more stresses being absorbed by the soil due to a decrease of pile stiffness, causing larger displacements which increase the stress in the piles.

According to the Swedish road administration (*Trafikverket*), the timber strength of an existing pile shall be chosen as K24 if no information regarding the pile characteristics are available (Trafikverket, 2023). This leads to a bending strength of $24\,\mathrm{MPa}$, a compressive strength of $20.9\,\mathrm{MPa}$ and a mean modulus of elasticity of $11\,\mathrm{GPa}$.

In the study conducted by Hemel (2023) the effect of deterioration of the timber piles where analysed by conducting several bending strength test on existing piles. It was found that the mean strength of the piles, when determined from the external pile diameter, is $10.6\,\mathrm{MPa}$ which is referred to as the modulus of rupture (MOR). It is stated that yielding starts above stress values of $MOR/1.7$ and that full yielding of timber piles occurs at roughly $1.7 \times MOR$. This "$90\,\%$ confidence interval" is shown in Table 2.1.

Table 2.1: Strength parameters of existing piles in Amsterdam (Hemel, 2023)

Yielding starts MPa	**Mean strength** MPa	**Full yielding** MPa
5	10.6	18.2

The strength parameters obtained from the timber tests, conducted by Hemel (2023), are low compared to the obtained mean values of the MOR from studied dry spruce specimens in Europe, where the mean MOR was found as $40.2\,\mathrm{MPa}$.

2.1.3 Gothenburg soil characteristics

The geotechnical properties of the soil around the quay walls vary but, as seen in the figures above, the soil often consists of filling material close to the surface and deeper down of clay. As seen in Figure 2.7 on the mainland side, south of the river, the soil consists of clay (*lera*).

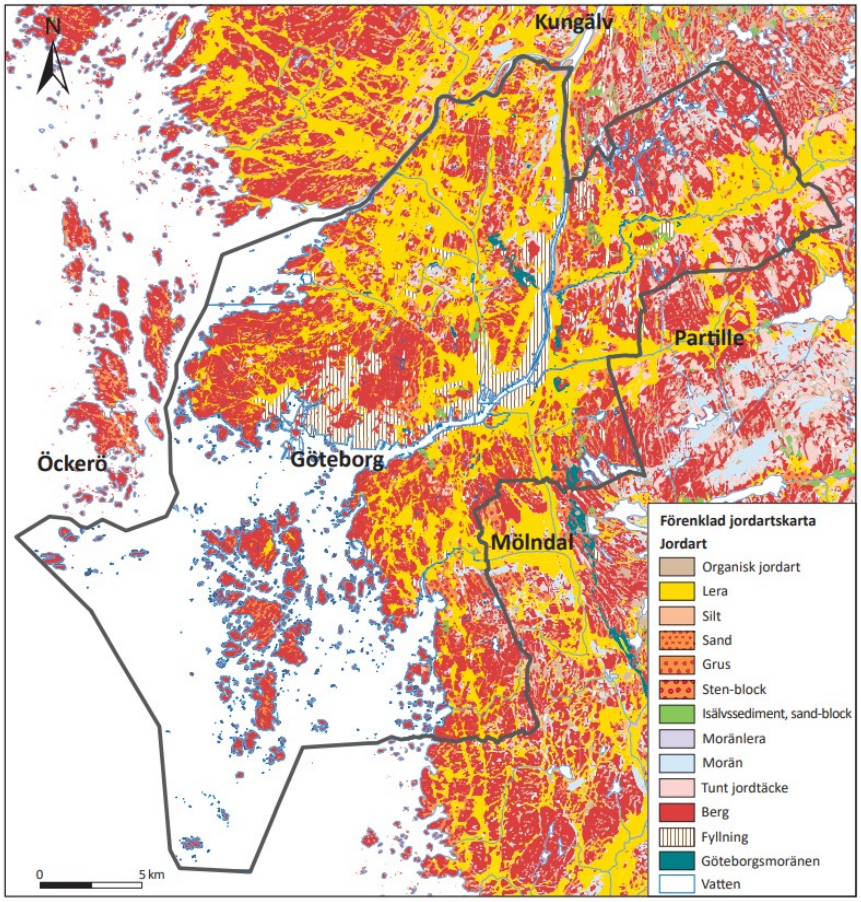


Figure 2.7: Simplified map of soil types in Gothenburg municipality (Bergström et al., 2022).

Geotechnical parameters used in design for frictional materials, such as the filling material, should be determined in a drained analysis. For clay, drained or undrained conditions can be applicable depending on the loading condition (Larsson, 2007). The undrained shear strength of clays can be determined based on results from site investigations. Soil properties of clay in a drained analysis is hard to determine but a general assumption for over consolidated Swedish clays with elastic deformations gives a frictional angle of $30°$ and an effective cohesion that depends on the over consolidation pressure as $c' = 0.03 \times \sigma_p$.

2.1.4 Stone masonry

According to Soutsos and Domone (2017) there are certain methods to achieve a stable masonry structure. For dry stone walls, a common technique is to place irregularly shaped and sized blocks by hand and link them together in a large mass. The main function of the mortar is to fill the gaps between the stones rather than act as a method to connect the stones in tension. Looking at Figure 2.5 it's probable that this type of technique was used to build the historical quay walls in Gothenburg.

2.2 Retaining walls

Retaining structures are utilised to support lateral loads from soils and rocks in slopes and embankments (Avilés L., 2015). They are also used to support vertical surcharge loads

acting on the soil behind it originated from various sources. A good description of the function of these structures can be found in the Mexican design guidelines (*Manual de Diseño de Obras Civiles de la CFE*). Some sources of lateral loading conditions are exemplified below:

- Soil-structure interaction on the interface between the wall and soil, due to friction
- Water pressure
- Surcharge loading
- Dynamic forces

Surcharge loading can have a varying layout such as a point load, uniformly distributed pressure, with a limited or unlimited distribution, and as a linear load distribution along the wall length (Avilés L., 2015). For existing structures it's important to define the geotechnical properties of the natural soils where the quay-wall is located. The following is necessary to determine:

- The profile and characteristics of the soils below the foundation.
- The profile and characteristics of the soils to be supported by the retaining wall (including permeability).
- The water table and site's hydrological conditions.

From this the following geotechnical parameters should be determined:

- Cohesion "c"
- Friction angle "ϕ"
- Permeability coefficient "k"

A retaining wall structure can be defined as a wall developing it's resistance against lateral loading mainly due to it's self-weight (Avilés L., 2015). Typical structures relevant to discuss in this thesis is gravitational walls (which the historical quay walls can be defined as) and cantilever walls, which is the main choice for new concrete retaining wall structures.

2.2.1 Gravitational walls

These type of retaining wall structures only utilise their self weight to carry lateral loads (Jaime P. & Fernández O., 2015). They are usually massive made out of rock, stone-, or brick masonry. An exemplification can be seen in Figure 2.8. The forces that may act on the gravitational walls can be seen in Figure 2.9, where the previously mentioned lateral loads are shown.

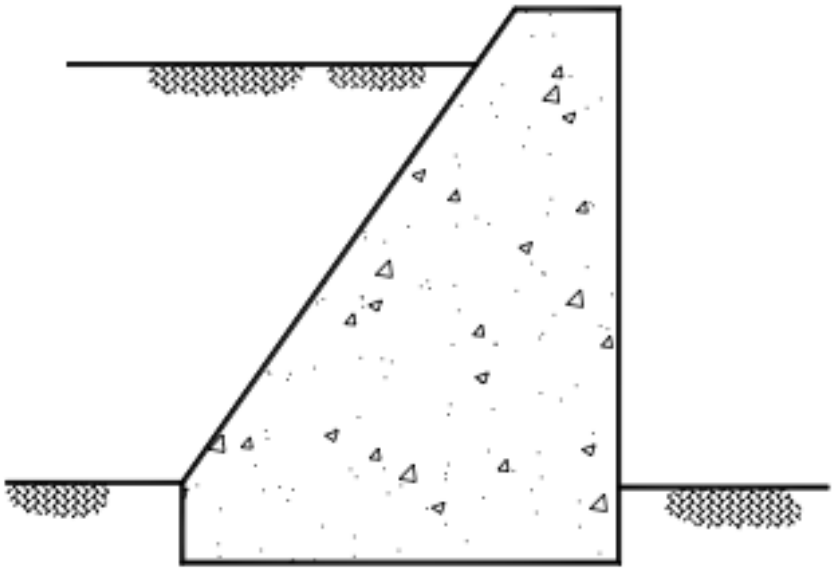


Figure 2.8: Exemplification of a gravitational wall (Jaime P. & Fernández O., 2015).

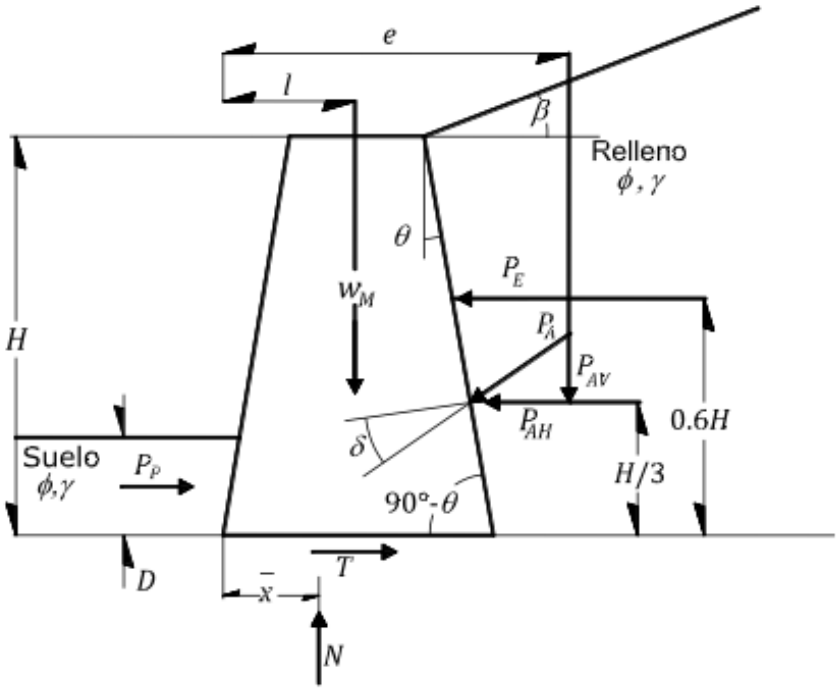


Figure 2.9: Scheme of forces acting against the quay wall (Jaime P. & Fernández O., 2015).

Where:

- Overturning forces:
  - P_A = Horizontal earth pressure from the soil
  - P_E = Horizontal earth pressure from surcharge loading
- Stabilizing forces:
  - W_M = Self-weight of the wall

- P_p = Horizontal earth pressure from the soil in front of the wall

To obtain equilibrium the forces with it's respective eccentricity needs to be in balance to prevent the wall from tilting and to make sure that the vertical stress in the supporting soil is less than it's bearing capacity. The horizontal component K is determined based on the soil state, active, at-rest or passive state (Avilés L., 2015). When the back of the wall is tilted against the backfill, the horizontal earth pressure is divided in a horizontal and vertical component according to Equation 2.1 and 2.2.

$$P_{AH} = P_A \times cos(\delta) \tag{2.1}$$

$$P_{AV} = P_A \times sin(\delta) \tag{2.2}$$

Where δ is the frictional component between the wall and backfill. The equilibrium condition is similar to what is presented in the Swedish handbook *Handboken Bygg* but there, the horizontal and vertical component are also considering the inclination (θ) of the wall (Algers et al., 1961b), (Algers et al., 1961a). The expression inside the parenthesis becomes $(\theta + \delta)$ and the friction between the wall and soil can be set to a maximum value of $2/3 \times \phi$ for coarse wall surfaces.

2.2.2 Cantilever concrete wall

As shown in Figure 2.10 these walls are composed by a concrete stem and foundation system creating an inverted "*T* " (Jaime P. & Fernández O., 2015). The T-shaped base helps balancing the structure and the base needs to be reinforced to resist the resulting moment-, and shear forces.

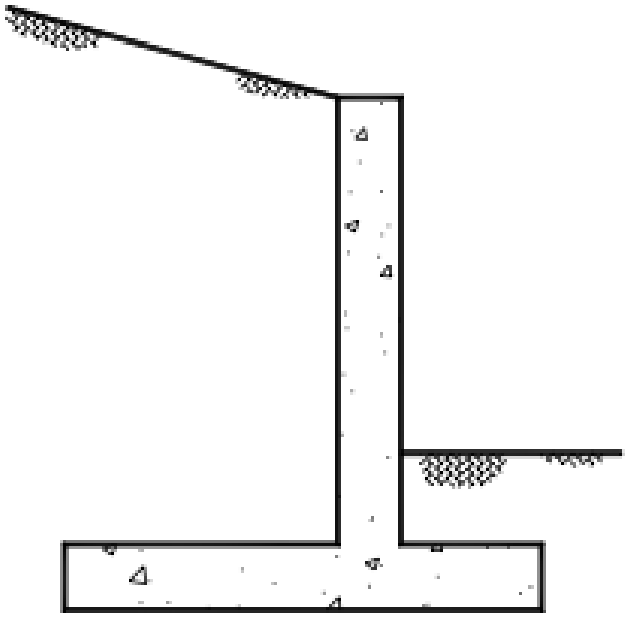


Figure 2.10: Cantilever concrete wall (Jaime P. & Fernández O., 2015).

2.3 Laterally loaded piles

2.3.1 Background

A common method to model the behaviour of laterally loaded piles is to model each pile as a beam on an elastic foundation with a bilinear approximation (Hemel, 2023). The lateral bearing soil-pile interaction are modelled by a series of independent lateral springs (p-y spring) with a plastic stress limit after which the soil starts to yield. Each spring has its own p-y curve based on the soil properties, depth, and pile dimensions.

Other methods exist to predict the p-y curves for the soil discretization such as guidelines from the Swedish Road administration (Trafikverket, 2023).

In the study of the Grimburgwal collapse in Amsterdam, the Netherlands (Hemel, 2023), the plastic limit was calculated by utilizing the theories presented by Brinch-Hansen. The plastic limit is reduced by applying the semi empirical strain wedge model (SW model), as explained by Ashour and Norris (2000), including group effects and geometrical boundaries such as sloping ground surfaces. See Figure 2.11.

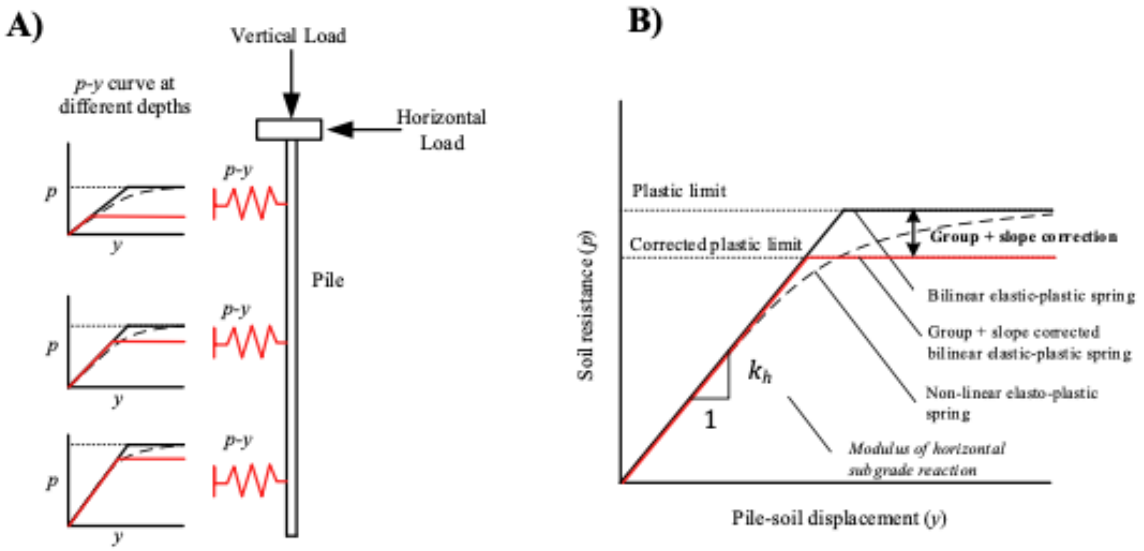


Figure 2.11: Diagrammatic illustration of the suggested BEF approach. A) When the pile is laterally supported by p-y springs, B) Close-up view of a p-y diagram and springs on the pile representation, where the black and red line represents the non-linear behaviour without and with reduction (Hemel, 2023).

Laterally loaded piles resist most of the horizontal forces in the top soil layers in the range of 10-15 pile diameter depth (Rollins et al., 1998), whereas it's especially important to properly define the soil-pile interaction in this soil segment. When a pile group is loaded laterally, the displacement increase with a higher load magnitude compared to the displacement of a single pile resisting the same average load. It's therefore important to consider group effects.

2.3.2 Lateral soil stiffness from the study of the Grimurgwal collapse

In 2020 a quay wall, around 150 years old, collapsed in the city center of Amsterdam, the Netherlands (Korff et al., 2022). A study of the collapse mechanism of the structure was

conducted by Delft University of Technology. The above mentioned analytical model was utilised and it was determined that the governing failure mechanism was horizontal bending of the piles in the pile group.

The horizontal elastic stiffness of the soil can, according to Hemel (2023), be expressed with the horizontal subgrade reaction. This is denoted as k_h, representing the elastic behaviour of the soil. This is derived by utilizing Ménard modulus of subgrade reaction according to Equation 2.3.

$$\frac{1}{k_h} = \begin{cases} \frac{1}{3E_m}\left[1.3R_0\left(2.65\frac{R}{R_0}\right)^{\alpha} + \alpha R\right] & \text{if } R \geq R_0 \\ \frac{2R}{E_m} \times \frac{4(2.65)^{\alpha}+3\alpha}{18} & \text{if } R < R_0 \end{cases} \tag{2.3}$$

The pressiometric modulus (E_m) is obtained by correlating it to the cone resistance obtained from pressuremeter-tests. The term R is half the diameter of the pile and R_0 a constant of 0.3. The values of the rheological coefficient α can be found in Appendix A. The modulus of subgrade reaction is measured in $\mathrm{kN/m^3}$ and depends on the pile diameter. To obtain the spring stiffness per meter pile length, k_h is multiplied with the pile-diameter.

The plastic limit of the soil at a certain depth, *z*, according to the Brinch-Hansen method for drained soil conditions can be determined according to Equation 2.4, which is based on Rankine's theory and experimental results (Christensen & Hansen, 1961).

$$\sigma_p = K_q \sigma'_v + K_c c \tag{2.4}$$

Where:
σ_p = Pressure at soil yielding per unit area [$\mathrm{kN/m^2}$]
σ'_v = Effective overburden pressure [$\mathrm{kN/m^2}$]
c = Effective cohesion [$\mathrm{kN/m^2}$]
K_q and K_c = Bearing factors considering overburden pressure / cohesion

Determination of the bearing factors can be found in Appendix B.

In the model presented by Hemel et al. (2022), pile group effects where taken into consideration with the **soil-wedge model**. With it, the plastic limit of the soil along the length of the pile is reduced, considering geometrical boundaries and pile group effects.

When several soil layers are involved, the pile is discretised in constant *dz* [m]. At each depth, the soil develops a failure plane based on a base angle β_m and a fan angle φ_m. The relation between φ_m and β_m is shown in equation 2.5.

$$\beta_m = 45° + \frac{\varphi_m}{2} \tag{2.5}$$

β_m express how the failure slice evolves from a studied depth "z" along the pile towards the surface in the x-z plane, and φ_m the propagation in the x-y plane. In the case of a quay wall, φ_m can be translated as expressing how the wedge propagates in the longitudinal direction of the wall.

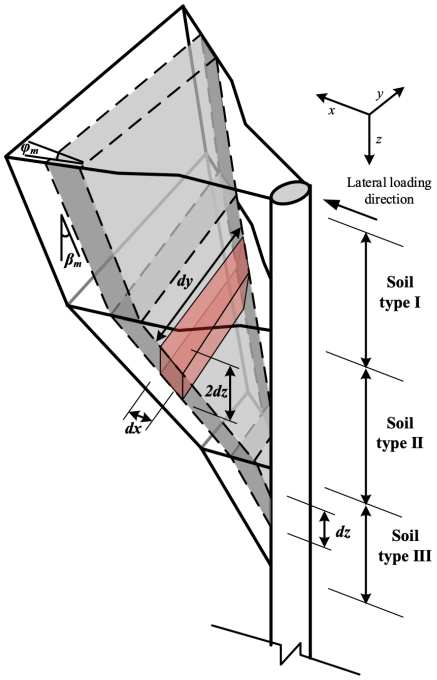


Figure 2.12: Three dimensional geometry of passive wedge in layered soil (Hemel, 2023).

If the friction angle (and accordingly the base-, and fan-wedge angle) changes in each soil layer, the shape of the wedge varies through the different layers (Ashour & Norris, 2000). The failure wedge of a single pile is exemplified in Figure 2.12.

Each failure wedge represents the plastic limit of the soil at the discretized depths along the pile length. When studying a pile group, the wedges from neighbouring piles collide which reduces the total horizontal capacity compared to a single pile. Since the described wedges represent the failure plane, the plastic limit of the soil is reduced and not the elastic behaviour.

The theory is derived under the assumption of freely developing failure planes, not considering the influence of nearby piles, sloping ground surface or other geometrical boundaries (Hemel, 2023). This can be accounted for by reducing the bearing factors according to Equation 2.6.

$$\sigma_p = K_q \Psi_\gamma \sigma'_v + K_c \Psi_c c \tag{2.6}$$

Where:
Ψ_γ = Correction factor for overburden pressure
Ψ_c = Correction factor for the cohesion

The correction factors are calculated according to Equations 2.7 and 2.8 below where the sub-indexes *"cpw"* stands for "corrected passive wedge".

$$\Psi_\gamma(z) = 1 - \frac{W - W_{cpw}}{W} = \frac{W_{cpw}}{W} \tag{2.7}$$

$$\Psi_c(z) = 1 - \frac{\tau - \tau_{cpw}}{\tau} = \frac{\tau_{cpw}}{\tau} \tag{2.8}$$

The correction factors represent the difference in weight and shear resistance along the sliding plane for a failure slice at a certain depth. The difference is measured between a pile in a pile group and the same pile considered as a single pile. The example in Figure 2.13 shows how the failure wedge is reduced by the pile in front of it.

The weight of each wedge and shearing resistance is determined based on Equations 2.9 and 2.10. For each wedge the volume and area of the shear plane of the soil, in each layer, is multiplied with it's corresponding density and effective shear strength. The contribution for each layer is summed up giving us the weight / shear strength of each wedge.

$$W(z) = \sum_{j=1}^{n} dz_j \times dx_j \times dy_j \times \gamma'_j \tag{2.9}$$

$$\tau(z) = \sum_{j=1}^{n} \frac{dz_j}{\cos(\beta_{m,j})} \times dy_j \times c'_j \tag{2.10}$$

Where:
W = Total weight of a failure slice developed at depth z [kN]
τ = Total friction of a failure plane developed at depth z [kN]
j = A range from 1 to n where n is the amount of different soil layers
γ' = effective weight of the soil at depth z [$\mathrm{kN/m^3}$]
c' = cohesion of the soil at depth z [$\mathrm{kN/m^2}$]
β_m = mobilized base angle at depth z [deg]

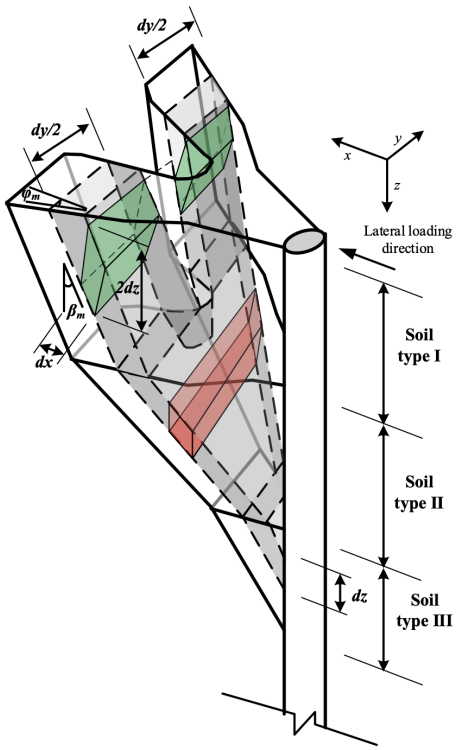


Figure 2.13: Passive wedge failure plane after reduction (Hemel, 2023).

To obtain the plastic limit of the soil along the length of one pile, the steps below are followed:

1. For each discretized spring, at depth "z":
   (a) Utilise Equation 2.9 and 2.10 for the unreduced failure wedges.
   (b) Repeat the previous step for the same pile but considering geometrical boundaries and pile group effects (reduced).
2. Calculate the reduction factors according to Equation 2.7 and 2.8.
3. Determine the plastic limit according to Equation 2.6.

The steps above are repeated, for each pile in the pile group, to obtain the ultimate soil capacity in the analysis.

2.3.3 Broms' method

Another present method to determine the plastic limit of laterally loaded soil was presented by Broms in the 1960's. Broms method considers only if the soil is either purely cohesive or frictional and assumed a perfect plastic behaviour of the soil-pile interaction (Broms & Asce, 1964), (Broms, 1964). The soil's ultimate resistance, p_{lim}, is mobilized at a critical displacement. When analysing laterally loaded piles in undrained cohesive soils, the pressure against the pile is assumed as 0 at the ground surface until the depth of 1-1.5 pile diameters, and then with a constant relation to the soil cohesion below this depth. See Figure 2.14, where the undrained shear strength is denoted by c_u. The assumption of the rectangular distribution is according to Broms on the safe side compared to the probable pressure distribution at failure.

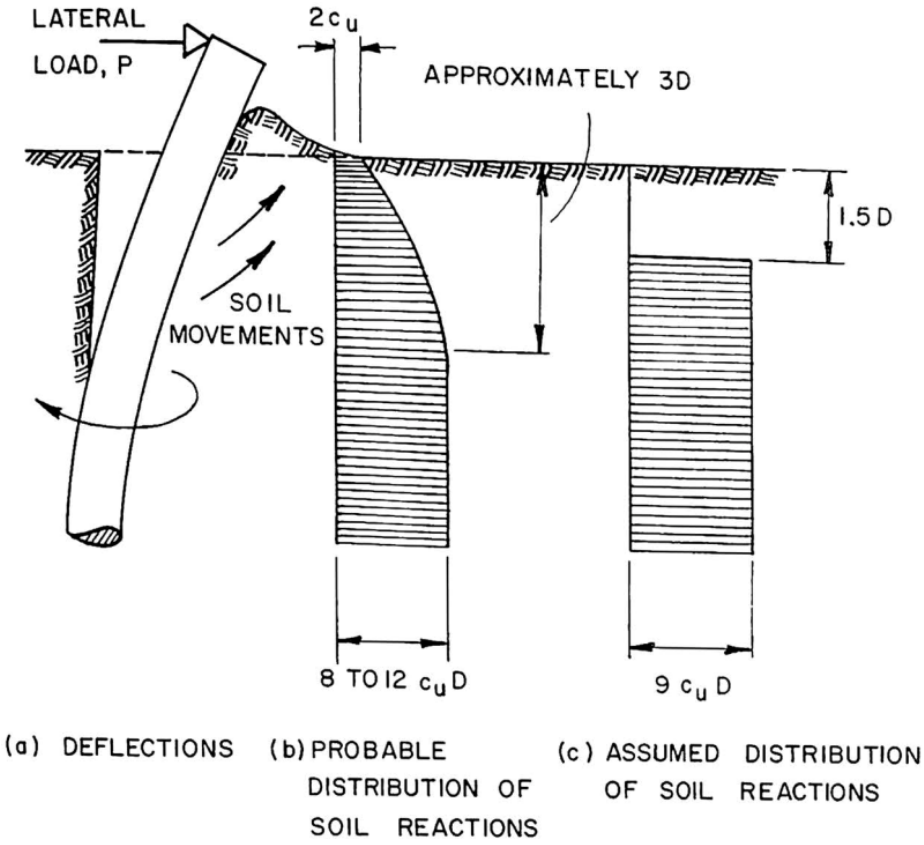


Figure 2.14: Assumption of pressure distribution for long piles (Broms & Asce, 1964).

The governing failure mechanism for the pile is assumed to occur when a plastic hinge is formed in the pile at a certain depth below the surface. This is explained by Broms and Asce (1964), and summarized in Figure 2.15. When this theoretical approach is applied for a quay retaining wall, failure may occur for a lower load than required to reach failure in the pile, for instance due to excessive deformations, leading to failure in the masonry structure.

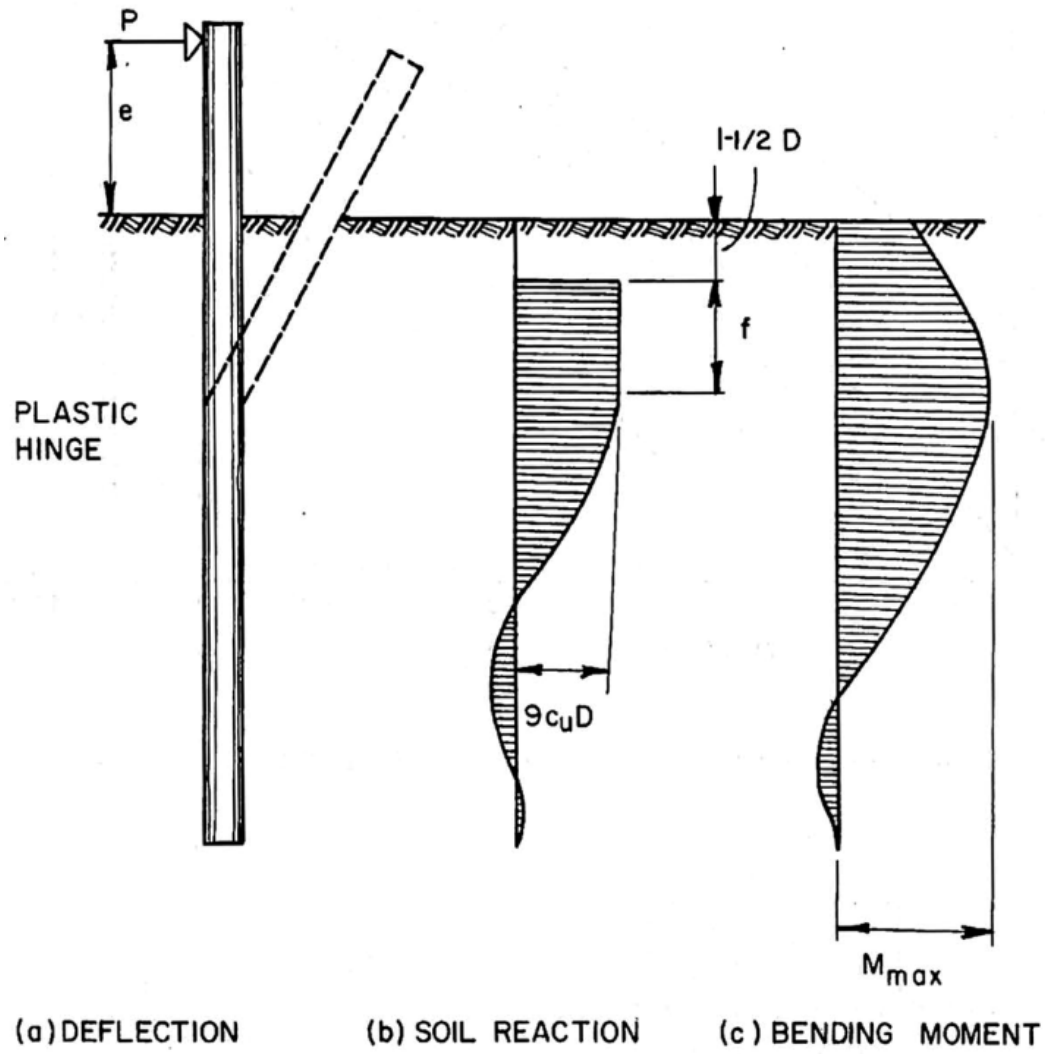


Figure 2.15: Failure mechanism, long pile in clay (Broms & Asce, 1964).

When determining the elastic lateral soil stiffness acting on the pile in non-cohesive soils it is assumed by Broms and Asce (1964) that it's increasing with the depth, z, according to the relationship $n_h \times \frac{z}{D}$. n_h is the coefficient of subgrade reaction for a long strip and D is the pile diameter. It's assumed that n_h is not dependent on the geometry or stiffness of the pile. This method is applied for non-cohesive soils by the Swedish road administration (Trafikverket, 2023).

The behaviour of the pile is related to dimensionless embedment depth ηL where $\eta = (n_h/EI)^{1/5}$ (Broms & Asce, 1964). It's concluded that a long pile is characterised as $\eta L > 4$, and increasing the embedment depth further won't decrease horizontal deflections of the pile at the surface. Increasing the stiffness of the pile on the other hand, reduce the displacements which have an effect on the behaviour of supported structure. The assumptions presented by Broms for laterally loaded piles is conservative based on comparisons between calculated displacements and measured values from real tests.

2.3.4 Broms extension method

Cecconi et al. (2019) propose a method to extend the theory presented by Broms to evaluate the ultimate soil pressure in drained soil condition, considering both the contribution of friction and cohesion ($c-\phi$ soils). The 3D effects, incorporated by Broms, have been used to create the profile of the horizontal soil pressure acting on the pile in the ultimate state when evaluating drained loading conditions. The theory utilises the well-known Rankine theory to determine the ultimate resistance of the soil in drained conditions, while the maximum pressure in the undrained state is identical to the theory presented by Broms. The soil resistance against the pile, comparing Broms' theory with the extended theory, is shown in Figure 2.16 where the soil profile according to Cecconi et al. (2019) is shown in subfigure d.

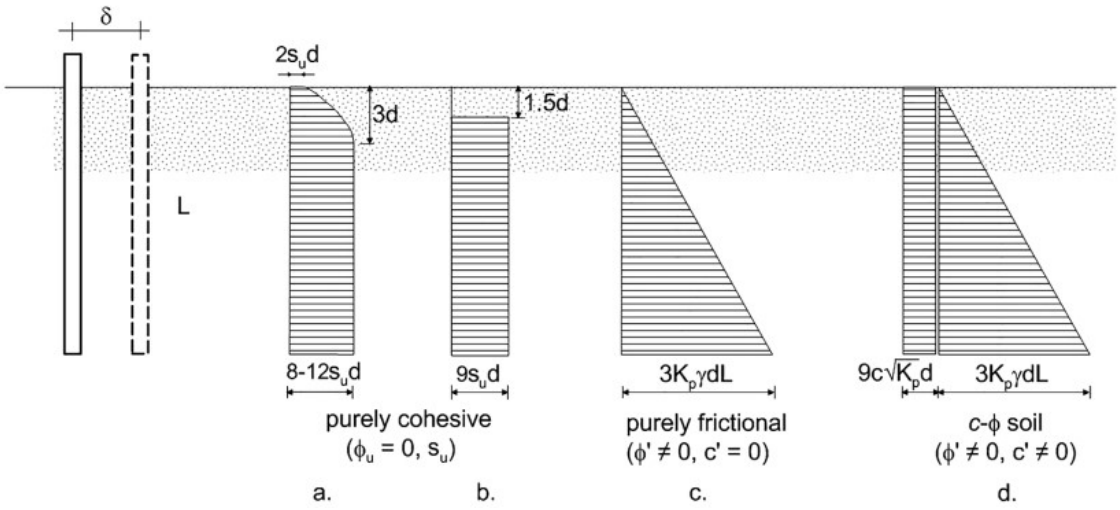


Figure 2.16: Comparison of lateral soil resistance against a pile with Broms'-, and extended method (Cecconi et al., 2019).

The final approximations of the ultimate pressure is presented in Equation 2.11 and 2.12, for drained and undrained soil response respectively, which for the drained state is shown in d in Figure 2.16. K_p denotes the passive earth pressure coefficient of the soil.

$$p_{lim} = (\sigma_v \times 3 \times K_p + 9 \times c' \times \sqrt{K_p}) \times d \tag{2.11}$$

$$p_{lim} = 9 \times c_u \times d \tag{2.12}$$

Cecconi et al. (2019) presents methods to determine the bending moment in piles with different lengths and boundary conditions. For a long pile, free to deform at the pile top, the equilibrium equations for horizontal and rotational moment can be solved by utilizing Equation 2.13 and 2.14 respectively when the horizontal load is applied at the pile head.

$$H(L) = \frac{3}{2} \times K_p \times \gamma \times d \times L + 9 \times c' \times \sqrt{K_p} \times d \times L \tag{2.13}$$

$$M(L) = H \times L - \frac{3}{2} \times K_p \times \gamma d \times \frac{L^3}{3} - 9 \times c' \times \sqrt{K_p} \times d \times \frac{L^2}{2} \tag{2.14}$$

Similar to Broms' method the theory assume plastic behaviour of the soil until the pile is assumed to have a rigid boundary condition at depth L (Cecconi et al., 2019). Comparisons between the extended Broms' theory and field tests show that the resulting horizontal capacity is in better agreement with the real behaviour, compared to the classic theory by Broms. At the same time, it has been proven that the classic theory provides a horizontal capacity determined on the safe side. Small values of the effective cohesion gives a significant increase of the horizontal capacity of the pile. It's important to chose this parameter with caution if the designer wish to utilise the extended Broms' theory.

2.4 Construction of *p-y* curves

The load transfer mechanism for laterally loaded piles is substantially more complex than axially loaded piles (Reed L. & Dawkins, 2000). In this case, the ultimate side friction at the pile-soil interface depends primarily on the soil shear strength at each point along the pile . The relationship between the soil resistance and pile displacement can be represented with the non-linear p-y curve (p-u curve), exemplified in Figure 2.17.

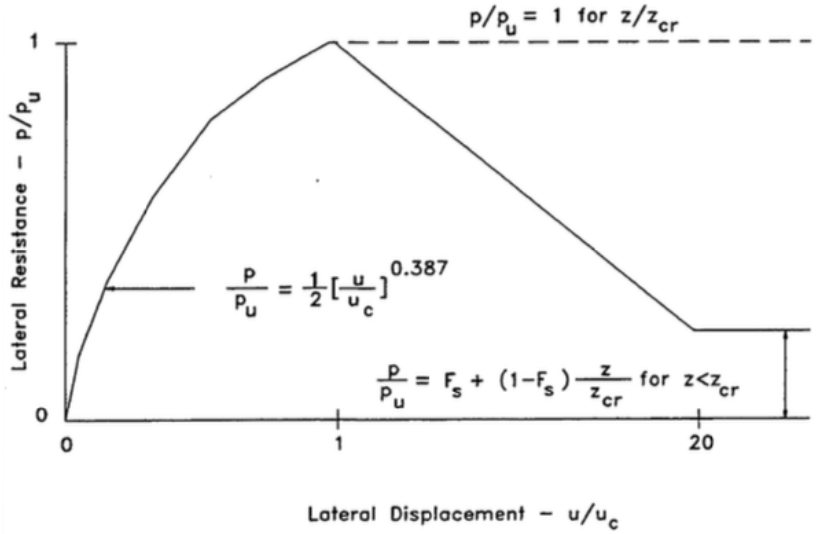


Figure 2.17: p-u curve for static loading (Reed L. & Dawkins, 2000).

In Figure 2.17, the load is normalized with the load ratio applied at depth "z" and the maximum load that the soil can resist at that depth, being when the soil starts to yield and show plastic behaviour.

In Reed L. and Dawkins (2000) various methods to build the *p-y* curves are stated and many, as in the Winkler assumption, assume that the lateral resistance *p*, at depth "z", is a function of the lateral displacement *y* only. If the pile is modelled as shown in Figure 2.18, the governing analytical equation is shown in Equation 2.15.

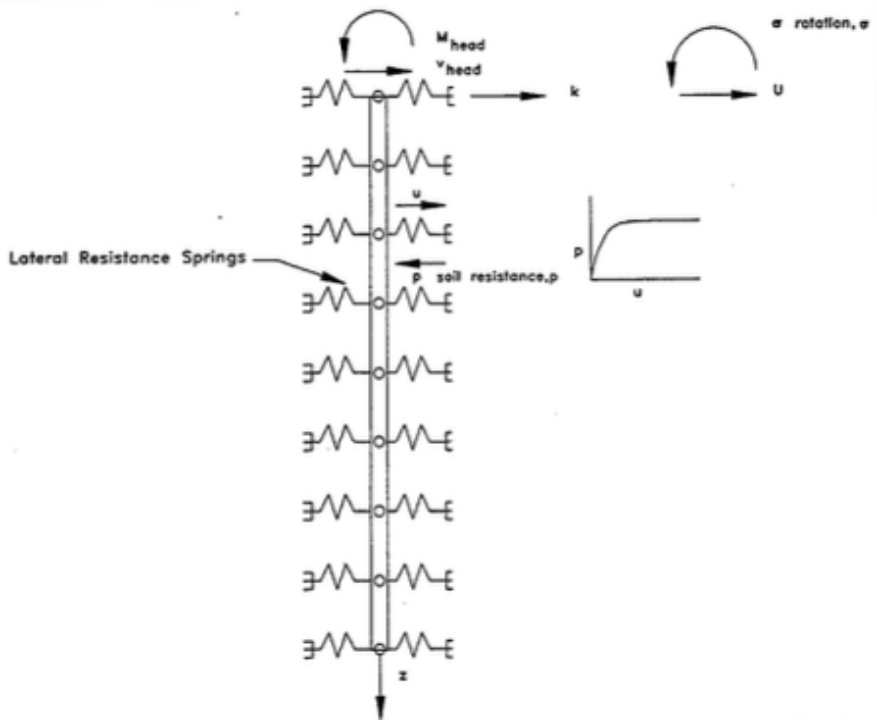


Figure 2.18: Model of laterally loaded piles supported by springs (Reed L. & Dawkins, 2000).

$$\frac{d^2}{dz^2}\left(EI\frac{d^2u}{dz^2}\right) + \frac{d}{dz}\left[P(z)\frac{du}{dz}\right] - p(z,u) = 0 \tag{2.15}$$

Where:
E = Modulus of elasticity of the pile.
I = Moment of Inertia of pile cross section about an axis perpendicular to the *x*-*z* plane.
P(z) = Axial compressive force in the pile at *z*.
p(z,u) = Lateral resistance which is a function of both position *z* on the pile and the lateral displacement *u* at *z*.

As mentioned before, to obtain the lateral resistance *p* the displacements *y* must be known (Reed L. & Dawkins, 2000). Therefore, iterative numerical calculations are needed. To do so, the non-linear *p*-*y* curves are in the first iterations replaced by equivalent elastic springs. The obtained force in the springs after each iteration is evaluated based on the solved displacement. If the force is lower than the plastic limit p_u the elastic stiffness is updated based on the secant of the *p*-*y* curve. If the force is beyond p_u the spring is replaced by the corresponding plastic force that the spring can resist.

It is worth to mention that the resistance increase with depth and therefore the displacements decrease in the same way (Reed L. & Dawkins, 2000). Consequently, if the distribution of soil stiffnesses along the pile can be determined, the behaviour may be evaluated for working loads without the need for iterative solutions. The second term in equation 2.15 represents the "beam-column" effect which is the interaction between the lateral load/displacements and the axial load of the pile. In most cases this effect is not substantial and can be obtained

by taking it as the force applied at the head of the pile.

2.5 Soil-Pile interaction

The stiffness of the pile have a large influence on the ultimate stress resisted by the soil, when the plastic limit is reached (Ashour & Norris, 2000). Generally, a stiffer pile will lead to the plastic limit not being reached. For flexible timber piles, large displacements are required to reach the ultimate stress. This is because of the slow growth of the passive wedge. The modelling of piles and their soil interaction is often conducted as a beam on an elastic foundation. The soil is represented by non-linear springs with a corresponding p-y curve for different depths. The non-linear behaviour originates from the plastic limit for each spring, depending on depth and soil-pile characteristics. In reality the soil behaves non-linearly before the plastic limit is reached. According to Broms and Asce (1964), the ultimate resistance of the pile is reached when the pile deflection at the ground surface is about $20\,\%$ of the pile diameter.

Boundary conditions for the pile head have a large significance when determining the depth of the passive wedge (Ashour & Norris, 2000). A pile with a fixed head will reach the ultimate load at a lower soil displacement compared to that of a free-headed pile, meaning that a larger passive wedge have been developed. The geometry of the pile will also influence the resistance where a square pile have a higher soil-pile resistance compared to a circular pile.

As stated by Broms and Asce (1964), failure of a free-headed pile occurs where either the soil along the pile exceeds it's ultimate limit and the pile rotates as a whole, or when the bending resistance of the pile have been reached. For a long pile, it's logical to assume that the soil will have enough capacity to resist the transversal load and the bending moment resistance of the pile will determine the ultimate load that the pile can carry.

2.6 FE-modelling tools

Finite-element (FE) software offer different methods of modelling the structural elements (Sharma et al., 2023). For example beam-, and shell elements can be used to model structural component. Beam elements can be utilised for modelling the piles and the timber flooring, supporting the masonry wall. The masonry wall can be modelled with 20-noded isoparametric quadratic solid elements where an advanced material model is required to properly capture the rock-rock-, and rock-timber interaction. This method of using solid elements requires advances FE-softwares. Modelling beam elements in FE-softwares can be done with different methods where two common options are "Euler-Bernoulli"-, and "Timoshenko"-type beams (Beck & Da Silva, 2011). The Euler-Bernoulli beams are suitable when modelling slender beams while the "Timoshenko"-beams can be used for both thick and slender beams.

An example of available FE-softwares are the following:

- FEM Design - Structural analysis software
- Plaxis - Geotechnical engineering software
- Simulia Abaqus - Simulation software used to analyze complex events realistically

Both FEM Design and Abaqus are FE-tools mainly used to model the structural components while Plaxis also offers the possibility of modelling the soil behaviour. All the above mentioned software offers the possibility of creating parametric models where the analysis runs through a script, making it possible to conduct a sensitivity study.

According to Sharma et al. (2023), the soil-, and structural models can effectively be decoupled. Incorporating this when modelling, only the pile foundation is modelled with finite elements or an analytical model. The load distribution from the wall, due to self weight and horizontal forces from the soil, is determined and the pressure distribution is applied to the pile foundation. The soil can be represented with linear elastic lateral springs with a defined limiting soil pressure when yielding occurs.

An important aspect to consider is non-linear effects, amplifying the stresses in the structure. This can occur due to displacements of the structure when the load is applied, or from non-linear behaviour of the material. This can be incorporated in the presented softwares by updating the displacement in each load increment, and introducing non-linear material models (“PLAXIS 2D 2024.3”, 2024), (Hougaard, 2024), (“Nonlinearity in Abaqus”, 2006).

2.7 Surcharge loads

As previously stated the quay walls in both the Netherlands and Sweden are affected by surcharge loading conditions from pedestrians and vehicles. In Gothenburg, Sweden, the quay walls are, in many cases, also loaded by trams passing along the quays.

The surcharge load from pedestrians and vehicles are, according to *Göteborg stad - Teknisk handbok*", determined from Eurocode (Göteborg stad, 2024). The magnitude when considering pedestrian loads are $5\,\mathrm{kPa}$ for the uniform load but there is also an accidental loading condition for service vehicles (“SS-EN 1991-2:2003”, 2011). Loads from vehicles are presented with various magnitudes and with different layouts which all need to be considered in the design.

3 Modelling of pile groups in quay wall structures

The capacity of the piles for the quay retaining walls in Gothenburg will be analysed by utilizing a developed method from TU Delft and with Swedish methods. To verify that the model is utilised correctly in this thesis, the result obtained from the analysis of the Grimburgwal collapse, made by Hemel (2023), will be recreated before applying it to the Gothenburg case study.

This comparison will be performed by using the commercially available FE-software Abaqus, to analyse the stresses and displacements of the foundation of the walls, and the software Grasshopper with Rhinoceros 3D to adapt the soil wedge model. Abaqus is believed to be suitable to fulfil the aim of the thesis and offers various methods of capturing the behaviour of the non-linear springs (NL-springs) and a possibility of creating parametrised models.

Hand calculations are also to be conducted by utilizing known theory. Since the analysis of transversally loaded pile is a non-linear problem due to the plastic limit of the soil (Hemel et al., 2022), hand calculations serves as a comparison of the results, evaluating the FE-model behaviour and the analysis of the masonry wall.

Due to the complexity of modelling the behaviour of the masonry structure the method of decoupling the soil and loads, according to Sharma et al. (2023), is utilised. The pressure from the wall is obtained and applied in the models of the pile foundation with the methods presented by Avilés L. (2015).

3.1 Structural geometry and material properties

For the comparison with the reference study in Amsterdam the proposed geometry, and material properties of the structure and soil according to Hemel (2023) will be utilised in the case study of the Grimburgwal collapse. The masonry structure is similar to the existing structures in Gothenburg which are also founded on timber piles supporting a timber deck. The difference is the masonry type structure where brick material was used in the Amsterdam case while the masonry structures existing in Gothenburg are mainly built with stone.

For the case study in Gothenburg assumptions regarding the geometry, and material properties will be made in the thesis. For example, degradation of the timber piles and flooring need to be considered according to inspections made on the quay walls in Amsterdam (Sharma et al., 2023).

3.2 Plastic-limit stress of soil springs

To obtain the plastic-limit stress for the lateral-soil springs in each soil layer, a Python code was developed. To obtain the corrected limits the soil-wedge model was implemented according to Hemel et al. (2022), where the unreduced and reduced volume and shearing surface, of each soil wedge, is calculated. The reduced values are calculated to account for the sloping ground surface in the canal and due to pile group effects.

The procedure is visualized in Figure 3.1. The reduced wedge is marked in yellow where the pink part, which contributes to the unreduced wedge, is excluded.

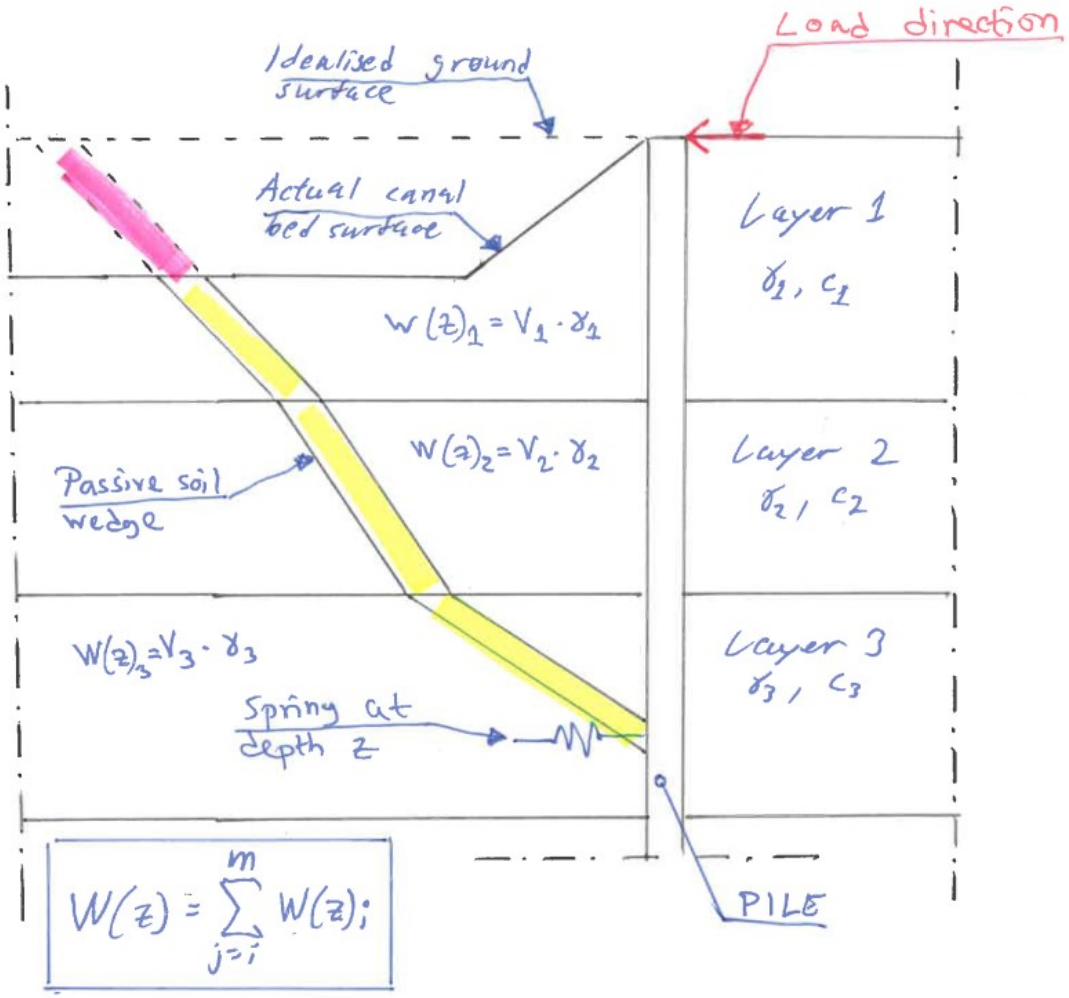


Figure 3.1: Passive soil wedge, layer contribution.

The coordinates in x, y and z at each new soil layer, for each wedge at a certain depth z along the pile depth, was primarily calculated with a Python script. Utilizing the software Rhinoceros together with Grasshopper the volumes of the wedges and areas of the shearing surfaces were obtained, both for the unreduced and reduced wedges.

Each wedge has a contribution to the ultimate resistance based on the soil characteristics from each layer it passes. Therefore the total volume and shearing surface, belonging to each wedge, is divided into the parts passing through each soil layer and multiplied with the density and cohesion of that specific soil layer. The contributions to the plastic limit of the soil springs, due to overburden pressure and effective cohesion, is calculated according to Equation 2.6. The contributions are reduced with the reduction factors according to Equation 2.7 and 2.8 respectively.

3.2.1 Limit Stress

To obtain the plastic limit of the soil, a Python code was created implementing the theory presented by Hemel et al. (2022) or Cecconi et al. (2019), depending on the case study. The code is shown in the appendix for the respective case studies.

The Python code requires the following input in a drained analysis:

- Depth of different soil layers
- Value of vertical spring spacing (dz)
- Pile Diameter (D)
- Characteristics for each soil layer:
  - Volumetric weight (γ)
  - Effective cohesion (c')
  - Internal friction angle (ϕ')

For an undrained analysis, the undrained shear strength of the soil, c_u, is required instead of c'.

Overburden pressure is calculated in the script by utilizing the following Equation:

$$\sigma_{vi} = P_0 + \gamma_i' * (d_{i+1} - d_i)$$

As an example, the plastic limit for different depths is shown in Figure 3.2, utilizing the soil conditions from the *Grimburgwal* case. The plastic limit is shown for both the Brinch-Hansen-, and Broms' extension method.

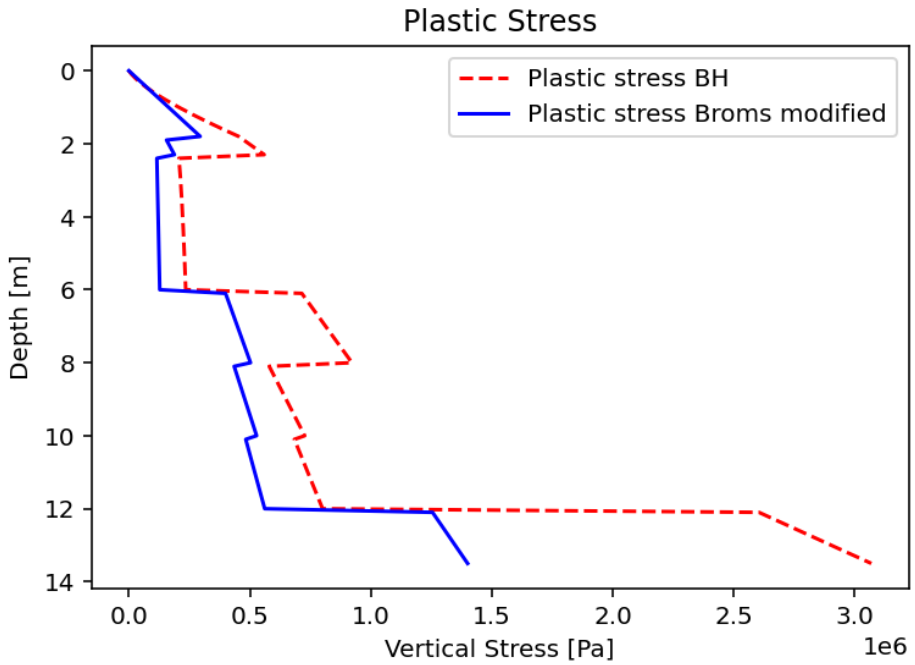


Figure 3.2: Brinch-Hansen and Broms' extension comparison.

3.2.2 Reduction Factor

In order to consider pile group effects and the slope of the canal, with the soil wedge reduction model according to Equation 2.6, two reduction factors where determined for each spring position. The factors consider the contribution to the limit stress from overburden pressure and cohesion. To obtain these areas and volumes, the visual programming software Grasshopper was used with Rhinoceros 3D software. This aspect will be discussed in Section 3.3.

To obtain the reduction factors, Ψ_γ and Ψ_c, a Python code was developed that extracts the output volumes and shear areas, for the unreduced and reduced wedges, from Grasshopper. The complete Python code can be found in Appendix C. Since the reduction factors are different for each pile and condition, the output from the Grasshopper model is arranged as a function of pile, soil type, reduced or unreduced shape, and if it's related to the weight or cohesion. To avoid confusion due to this matter, the following suffix are created:

- P1 $\rightarrow$ Pile one (back-row pile)
- P2 $\rightarrow$ Pile two (front-row pile)
- R $\rightarrow$ Reduced
- UR $\rightarrow$ Unreduced
- A $\rightarrow$ Area
- V $\rightarrow$ Volume

When the contributions from each soil type, pile and type of wedge is included, the respective ψ-factors can be calculated, according to the Equation 2.8 and 2.7. When the reduction factors are obtained, these are exported to a *csv* file which is introduced in the Abaqus model to obtain the corrected plastic limit of the lateral soil springs.

3.3 Grasshopper

3.3.1 General Model

Grasshopper was used to model the soil-wedges and obtain the data necessary to calculate the reduction factors. Grasshopper is a visual programming tool integrated with Rhinoceros 3D, used for generative design. It lets users create complex forms and parametric models without traditional coding.

As already mentioned the coordinates for each soil wedge, for each dz position along the pile, to the ground surface, was obtained from the Python code. The coordinates represent the propagation of the wedge, in a Cartesian coordinate system in three dimensions, based on the β-, and φ- angle shown in section 2.3. This code was inserted in a Python module in Grasshopper. Some parameters were also added outside the code in order to facilitate the modification of the Grasshopper model in the future, see Figure 3.3. More detail can be found in Appendix D.

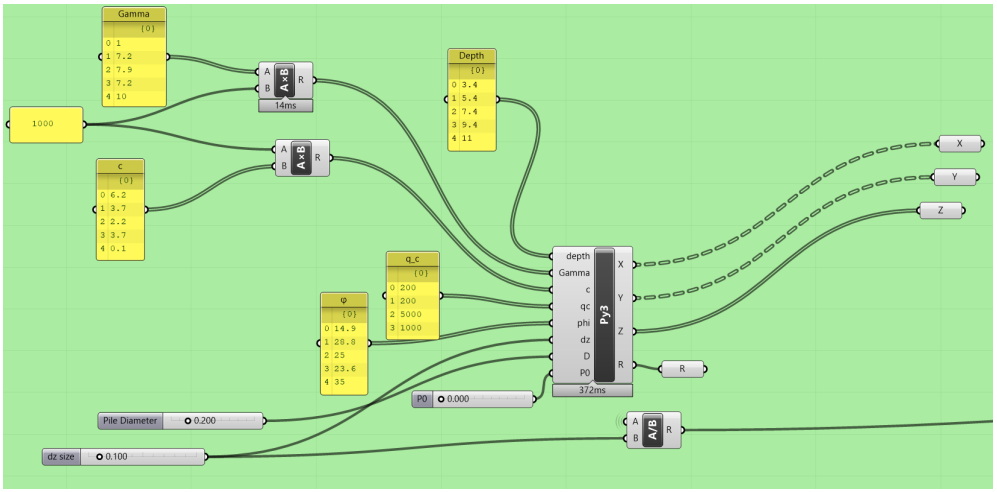


Figure 3.3: Python module in Grasshopper used to obtained the wedges.

Initially the piles are modelled, serving as a guide for the soil-wedges and later as a base to verify the obtained shapes. The piles are shown in 3.4.

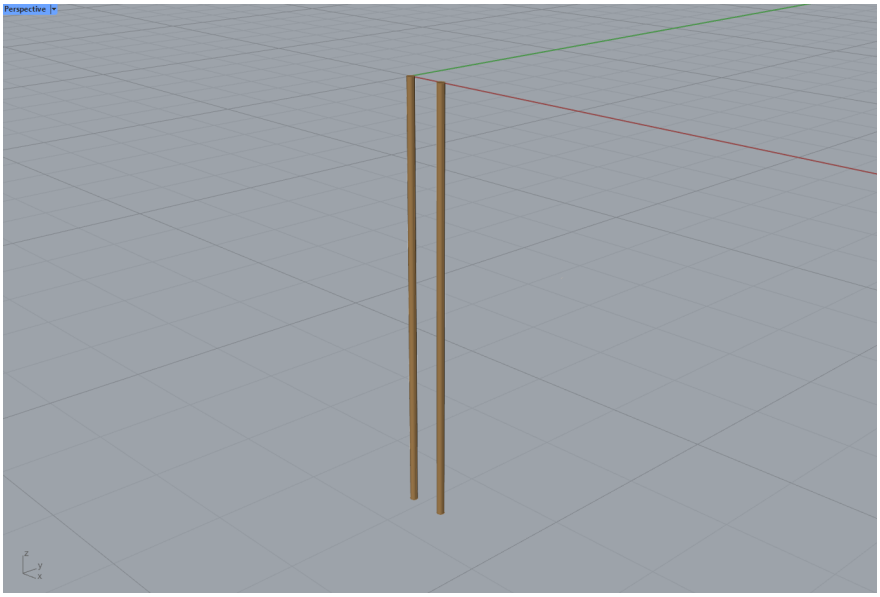


Figure 3.4: Pile model.

From the coordinates, the outer surfaces were created in a local ZX-plane representing the outer boundaries of the wedges. These surfaces where joined together to form the volume of the wedges and the shear surfaces. In Figure 3.5 the wedges are shown, where the contour plot displays the volume change of each wedge.

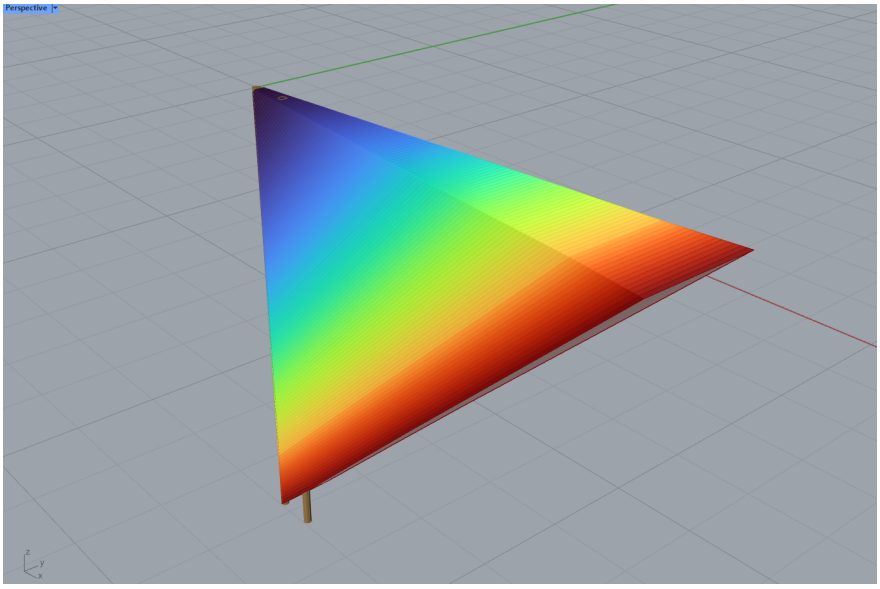


Figure 3.5: Contour wedges volumes for Pile 1, not including group effects or geometrical boundaries (unreduced).

Up to this point, the wedge is identical and unreduced for each pile with the difference in the origin of the curves. The reduced wedges for each pile, created with Grasshopper, starts from this model.

3.3.2 Reduced Model

The first reduction was made the due to the slope on the canal side. To do so, a solid with the idealized canal shape was modelled, as shown in Figure 3.6.

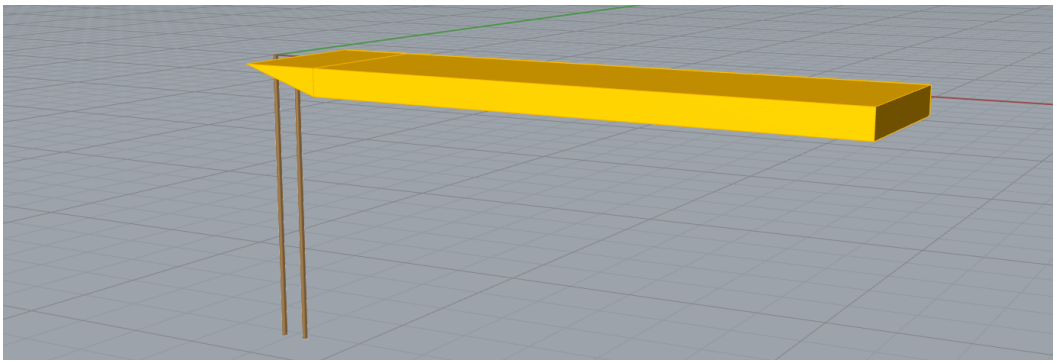


Figure 3.6: Solid representing the volume that is removed from the initial wedge due to canal topography.

The volume from the canal model needs to be subtracted from the unreduced wedge. The result from this subtraction can be seen in Figure 3.7, which also shows the wedge from a trailing pile.

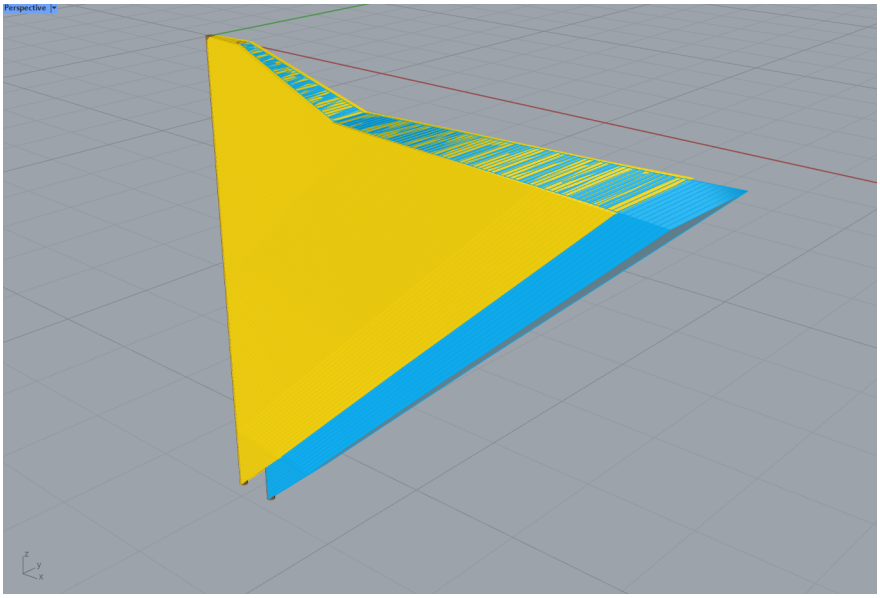


Figure 3.7: Combined volume-wedges from Piles 1 and 2, reduced due to the canal.

The reduction due to pile group effects considers spacing of the piles, perpendicular to the loading direction, and intersection of volumes due to trailing pile rows. The wedges, after this is implemented, can be seen in Figure 3.8.

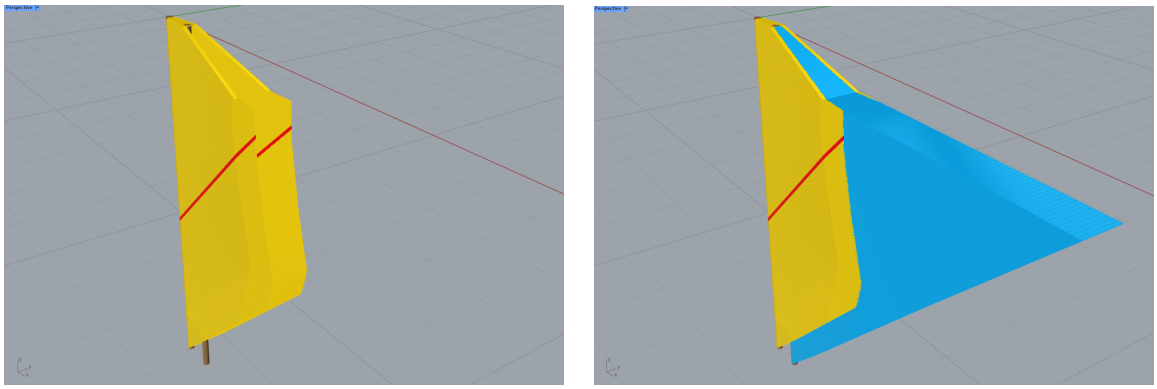


Figure 3.8: Left: All wedges along P1 and a random wedge at a certain depth (in red). Right: Reduced volumes, P1 and P2, showing the same random wedge from the left figure.

From Figure 3.8 it's clear that a large reduction occurs for the trailing pile, and for the top soil layers against the front pile. This is the final general volume/shape of the wedges. With this, it is shown how the sketch from Figure 3.1 looks in the Grasshopper model. It also highlights the importance of considering pile group effects.

A comparison of how the wedges change between the unreduced and reduced model is shown in Figure 3.9.

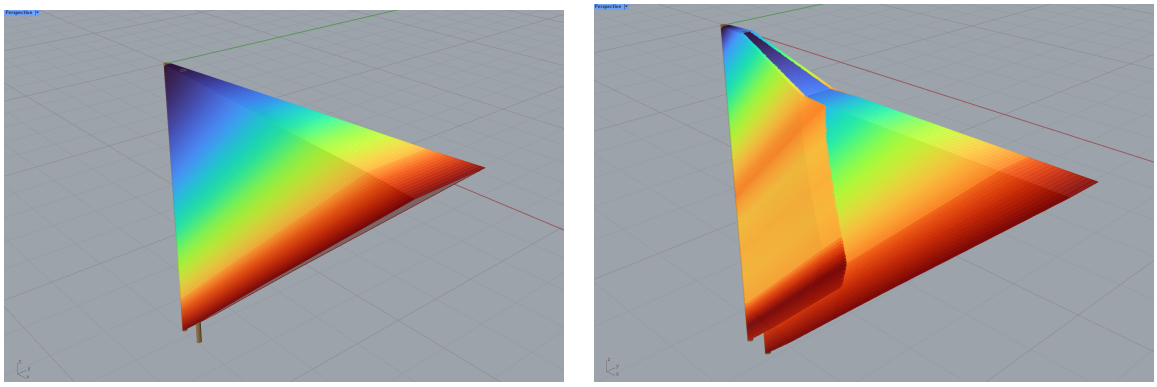


Figure 3.9: Comparison of soil volume-wedges of two piles, where the blue colour represents smaller volumes and red larger volumes. Left: unreduced model, Right: reduced model.

To account for varying soil properties in the different soil layers, the volumes and shear surfaces are split with a plane surface, at the depths where a new soil layer starts. The reduced, layered volume and shear surfaces are shown in Figure 3.10 and 3.11 respectively.

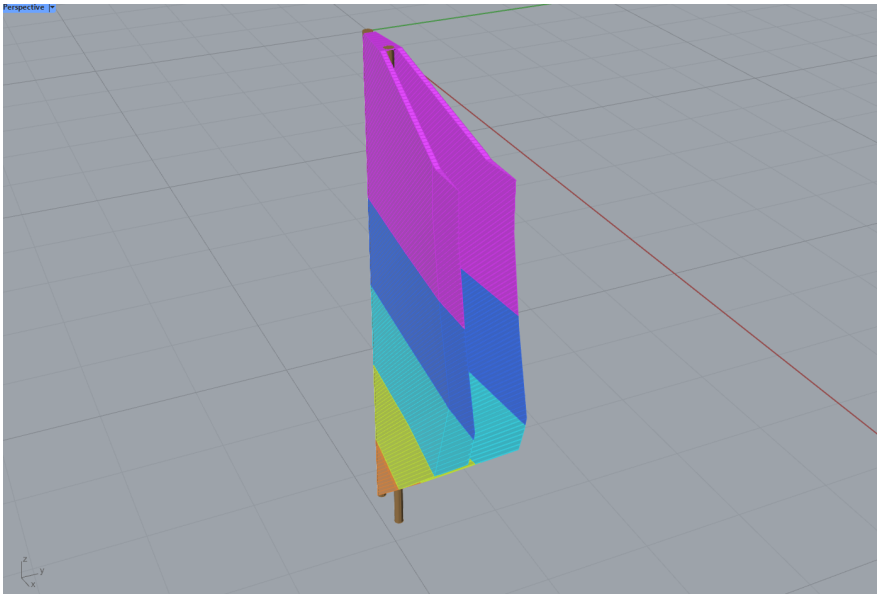


Figure 3.10: Reduced volume from P1 with soil layers.

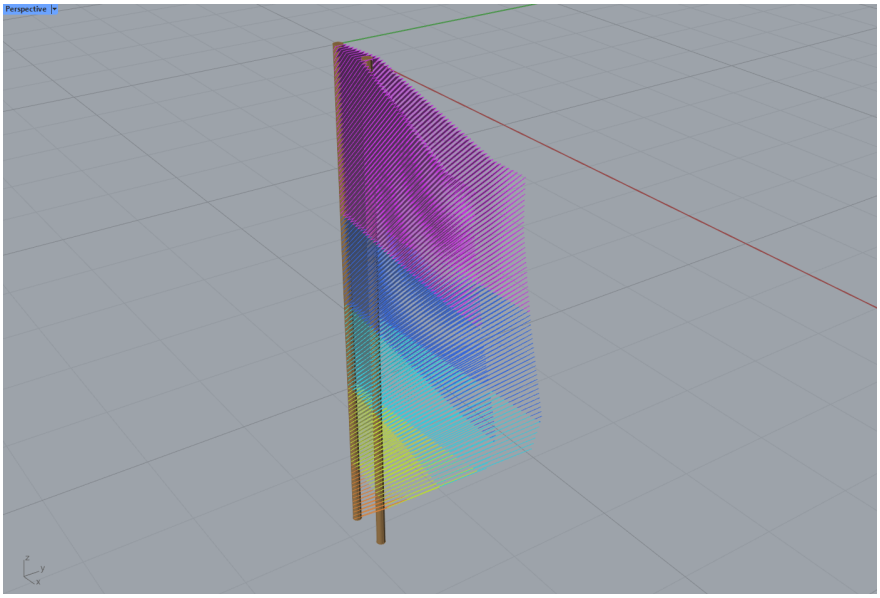


Figure 3.11: Reduced Surfaces from P1 with soil layers.

For each soil wedge, the volume and shear area, for each unique layer, is multiplied with the corresponding weight and shear strength. The contribution from each layer is summed up, resulting in the final weight and shear resistance of each soil wedge for the reduced and unreduced model.

As a comparison, the same wedge as pointed out in Figure 3.8 is highlighted in Figure 3.12, showing how it passes through different layers.

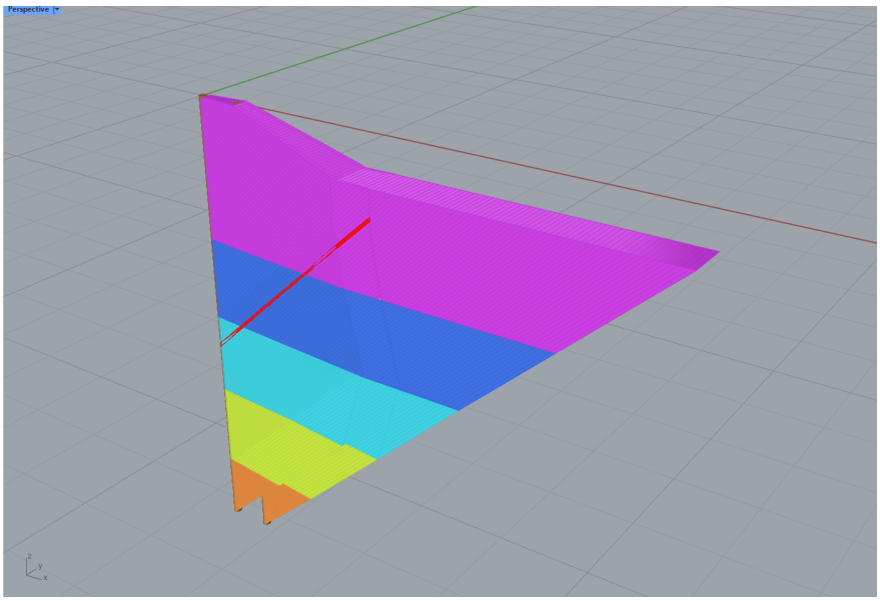


Figure 3.12: Random wedge from P1 (red), passing through different soil layers.

The objective of creating these models is to obtain the corresponding reduction for each wedge. These reduction factors are calculated in Python, utilizing Equation 2.7 and 2.8, comparing the weight and shear resistance, of each wedge, between the unreduced and reduced models.

3.4 Loads

An explanatory figure of how the loads are applied to the pile foundation is shown in the paper presented by Hemel (2023) and shown in Figure 3.13.

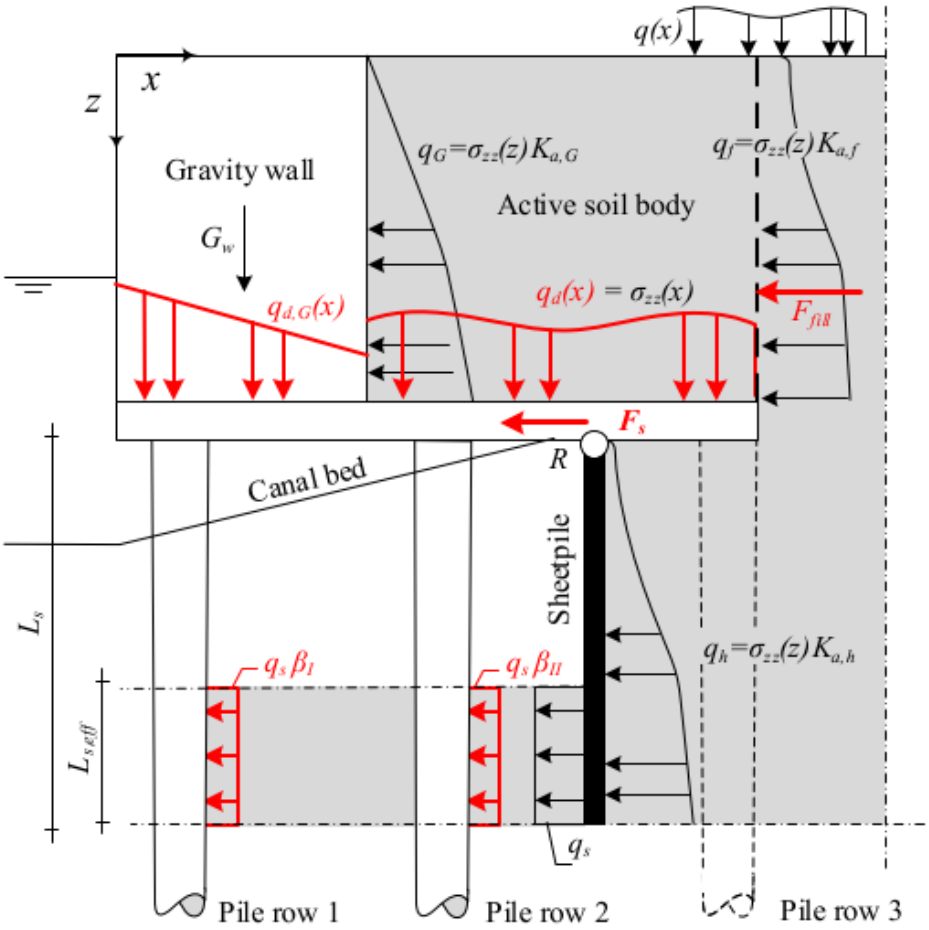


Figure 3.13: Modelling loads acting on the pile foundation (Hemel, 2023).

The vertical loads acting on the timber deck, that are later transferred down to the piles, originate from the following sources:

- Vertical stress from soil on top of the timber deck.
- Self weight of the masonry wall.
- Vertical surcharge load, applied to the ground surface and transferred through the soil, down to the wall and timber deck.

If the surcharge load, from traffic or pedestrians, is assumed to act on the whole ground surface behind the retaining wall structure, the same vertical stress acting on the ground surface will be distributed down to the timber deck.

Horizontal forces originate from earth pressure and surcharge load, transferred through the soil, supported by the gravity wall and the soil body located on top of the timber deck.

A simplification can be made when modelling the resulting forces on the timber deck beneath the masonry wall. The eccentricity of the vertical load, from the gravity wall when

affected by the overturning moment from the soil and surcharge load, can be calculated according to EQ 3.1. *M* is the overturning moment and *V* the vertical force from the gravity wall.

$$e = \frac{M}{V} \tag{3.1}$$

The vertical stress beneath the wall can then be calculated according to the Swedish handbook *Plattgrundläggning*. There it is stated that the distribution depends on the location of the resultant of the vertical force beneath the wall (Bergdahl et al., 1993). The theory is presented for slab foundations but is assumed to be valid for the pressure distribution beneath a quay wall as well.

An eccentricity larger than $B/6$, where B is the width of the wall, leads to lifting at the back of the wall since the connection between the wall and timber deck cannot withstand tensile forces. The stress beneath the wall is determined based on Equation 3.2, considering a longitudinal strip of $1\,\mathrm{m}$.

$$\sigma_{max} = \frac{4 \times V}{3} \times \frac{1}{(B - 2 \times e)} \tag{3.2}$$

If the eccentricity is less than $B/6$, no lifting occurs but the stress distribution on the edge in the direction of the overturning moment is larger. The stress distribution, at the respective edges of the wall, is described by Equation 3.3.

$$\sigma_{max/min} = \frac{V}{B} \pm \frac{6 \times M}{B^2} \tag{3.3}$$

3.5 Abaqus FE-model

Abaqus allows different approaches to model the quay structure, both in 2D and 3D. To be able to validate the model against the results obtained in the case study of the Grimburgwal a 2D model is chosen, since the same approach was utilised there.

3.5.1 Piles

For a 2D model the piles can be modelled as beam elements allowing for in-plane and rotational degrees of freedom (DOFs). The soil-pile interaction is modelled by utilizing springs, containing the elastic behaviour of the soil for the different soil layers as well as the plastic limit. The soil is discretized and represented by springs located at a specified distance. As an example, Hemel (2023) applied springs against the pile beam elements where each spring represented 0.1 m soil thickness while for the pile foundation of the pier barrier protection system of Hisingsbron, each spring represented $1\,\mathrm{m}$ soil thickness (Aygül et al., 2017).

The springs in Abaqus can be modelled with the spring to ground tool. The spring is assigned to a specific node along the length of the beam element and assigned an axial stiffness in specified DOFs. For laterally loaded piles, the DOF in the horizontal direction requires a specified stiffness and it may or may not be applicable to assign a DOF in the vertical direction, depending on if the pile is resting on a firm base (stiff sand layer or bedrock for example), is floating in the soil, or a combination of both conditions.

If the pile is short and resting on a firm base, it's applicable to not assign a DOF in the vertical direction and the opposite goes for when the pile is long. In this thesis the piles are, as a simplification similar to what was conducted by Hemel (2023), assumed to have a fixed vertical support at the pile base.

The beam element and springs have been verified in a small scale model verification which can be seen in Appendix E.

3.5.2 Timber deck

The timber deck is chosen to be modelled as a beam element to be able to capture the stresses in the structure and to properly capture the force distribution through the deck down to the piles, for both vertical and horizontal forces.

The connection of the piles to the timber deck is important to consider, as mentioned by Hemel (2023), and will be modelled as a hinged connection, transferring vertical and horizontal forces while allowing rotations. This can be done in Abaqus by modelling a "Tie" connection between the beam elements and setting the constraint ratio to 0. To validate this chosen method of modelling the timber deck, a verification of the results obtained in Abaqus was conducted which can be seen in Appendix E.

3.5.3 Masonry structure

The masonry structure requires a more advanced model than a beam element to properly capture the friction interface between the masonry, such as conducted by van Hulten (2021), when analysing historical quay walls in Amsterdam. These type of models will not be utilised in this thesis, instead the load from the masonry wall will be applied directly to the timber deck.

4 Case study 1 - Benchmark against the Grimburgwal

To verify that the Abaqus FE-model functions properly, the models created by Korff et al. (2022) for the Grimburgwal are recreated. The geometry of the wall is described by Hemel (2023) and shown in 4.1.

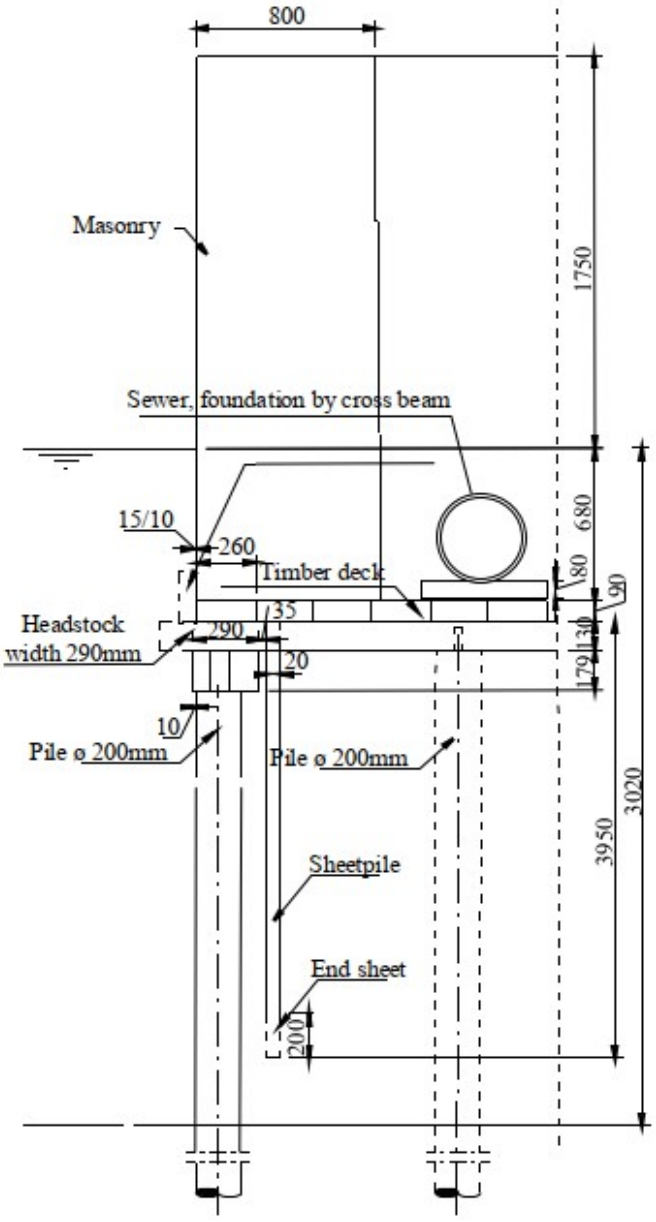


Figure 4.1: Geometry of the Grimburgwal in Amsterdam (Hemel, 2023).

The stresses in the piles from the Grimburgwal reference case, which the FE-model will benchmark against, are displayed in Table 4.2, showing the stresses for various combinations of pile diameters, canal bed levels, and number of piles. The canal bed level is represented with NAP, the Dutch national reference system for measuring elevations and depths.

Canal bed level with respect to NAP->		-1.2m	-1.6m	-2.0m	-2.4m	-2.8m	-3.2m
Model / Bending stress ->		[N/mm²]	[N/mm²]	[N/mm²]	[N/mm²]	[N/mm²]	[N/mm²]
1. D = 0.2m	2 piles	10.3	16.1	20.6	24.8	28.6	31.3
2. D = 0.2m with N	2 piles	12.0	22.8	32.4	43.3	55.9	68.5
3. D = 0.25m	2 piles	4.7	6.6	8.8	11.3	13.5	15.2
4. D = 0.25m with N	2 piles	4.9	7.1	10.1	13.3	16.3	17.9
5. D = 0.2m with N	3 piles	5.1	5.2	5.1	6.5	7.0	8.0
6. D = 0.2m with N	4 piles	3.0	3.2	3.0	2.8	3.0	3.3

Figure 4.2: Obtained stresses from the Grimburgwal reference case (Hemel, 2023).

As a limitation, the case when NAP is located at the top of the piles is studied (NAP = -1.2) with included normal force, amplifying bending moments due to second-order effects. A comparison is conducted for both D = $0.2\,\mathrm{m}$ and D = $0.25\,\mathrm{m}$. Maximum stress from the setups, used in the comparison, can be seen in Table 4.1.

Table 4.1: Model setups from the Grimburgwal reference case.

D [m]	**Number of piles**	**Max Stress** [MPa]
0.2	2	12
0.25	2	4.9
0.2	3	5.1
0.2	4	3

In the structural model, created by Hemel (2023), the soil was represented by a series of lateral-soil springs, assigned an elasto-plastic behavior. The structural model can be seen in Figure 4.3.

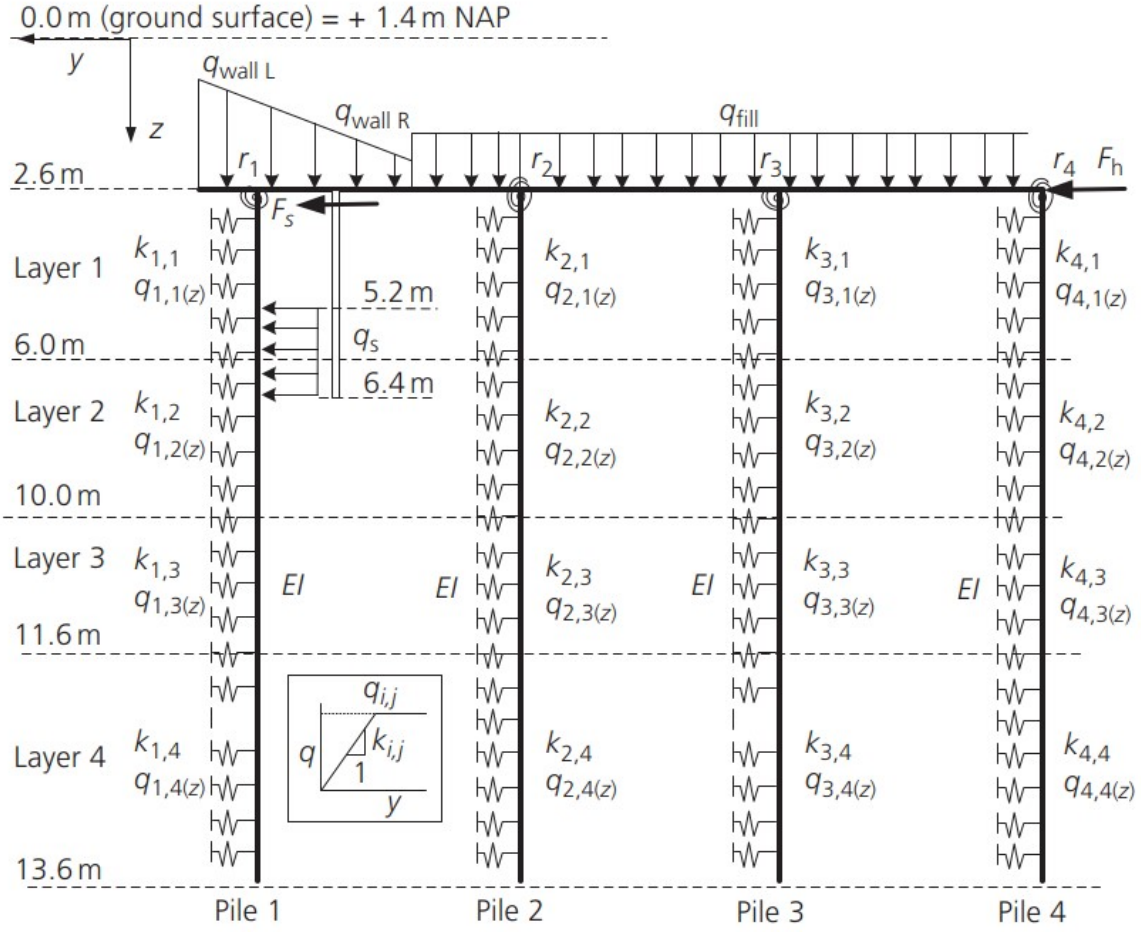


Figure 4.3: Structural representation of the Grimburgwal (Korff et al., 2022).

The soil properties at the location of the Grimburgwal, as stated in Hemel (2023), are shown in Table 4.2. The vertical stress field with these soil conditions is shown in Figure 4.4.

Table 4.2: Soil layer properties at the Grimburgwal (Hemel, 2023).

Depth [m]	**Sort** [-]	γ' [kN/m^3]	q_c [kPa]	c [kN/m^2]	φ [deg]
0.0 – 1.8	Sand dry	18.0	-	0.0	32.5
1.8 – 2.4	Sand wet	9.0	-	0.0	32.5
2.4 – 6.0	Peat	1.0	200	6.2	14.9
6.0 – 8.0	Sea clay	7.2	200	3.7	28.8
8.0 – 10.0	Wadsand	7.9	5,000	2.2	25.0
10.0 – 12.0	Clay	7.2	1,000	3.7	23.6
12.0 – 20.0	Sand	10.0	10,000	0.1	35

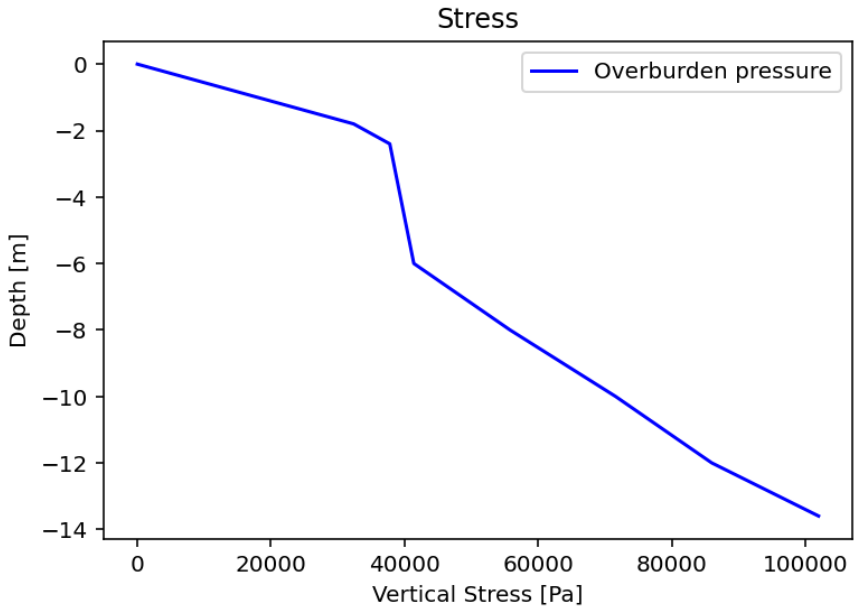


Figure 4.4: Overburden pressure from the reference case.

4.1 Verification model

The interpretation of the model, presented in the study of the Grimburgwal, in this work can be seen in Figures 4.5 and 4.6 where the superstructure and pile foundation, with the loads acting against the sheet-pile wall, are shown respectively.

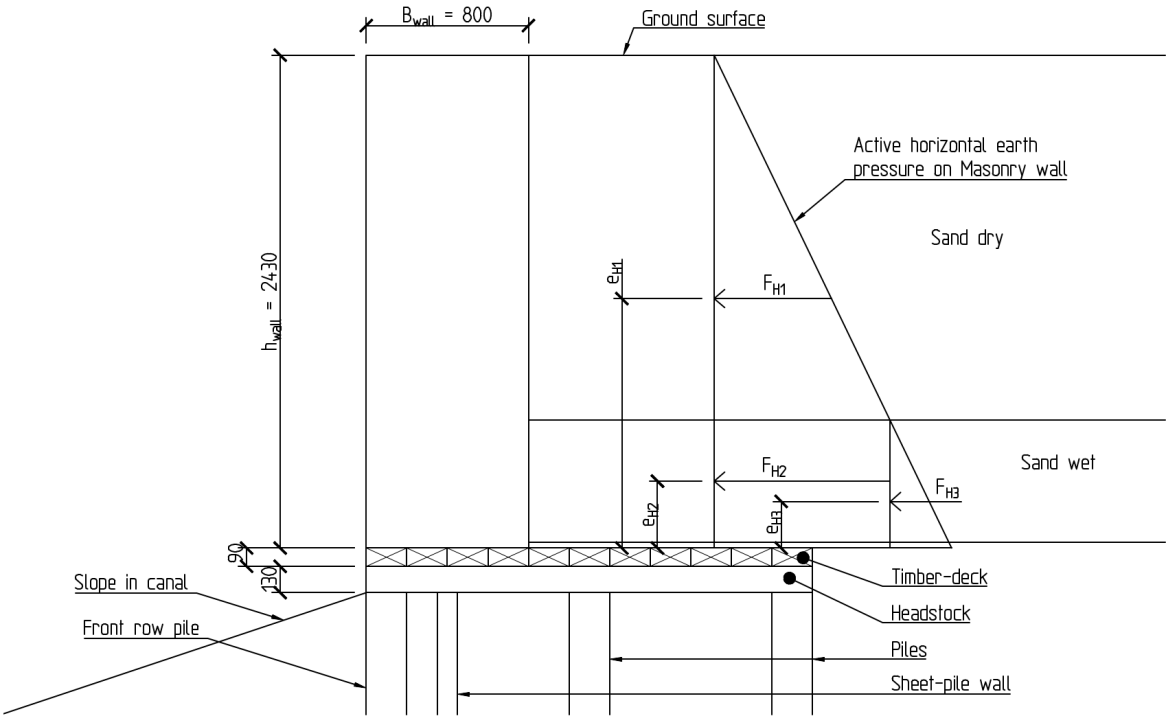


Figure 4.5: Interpretation of the Grimburgwal superstructure with applied loads.

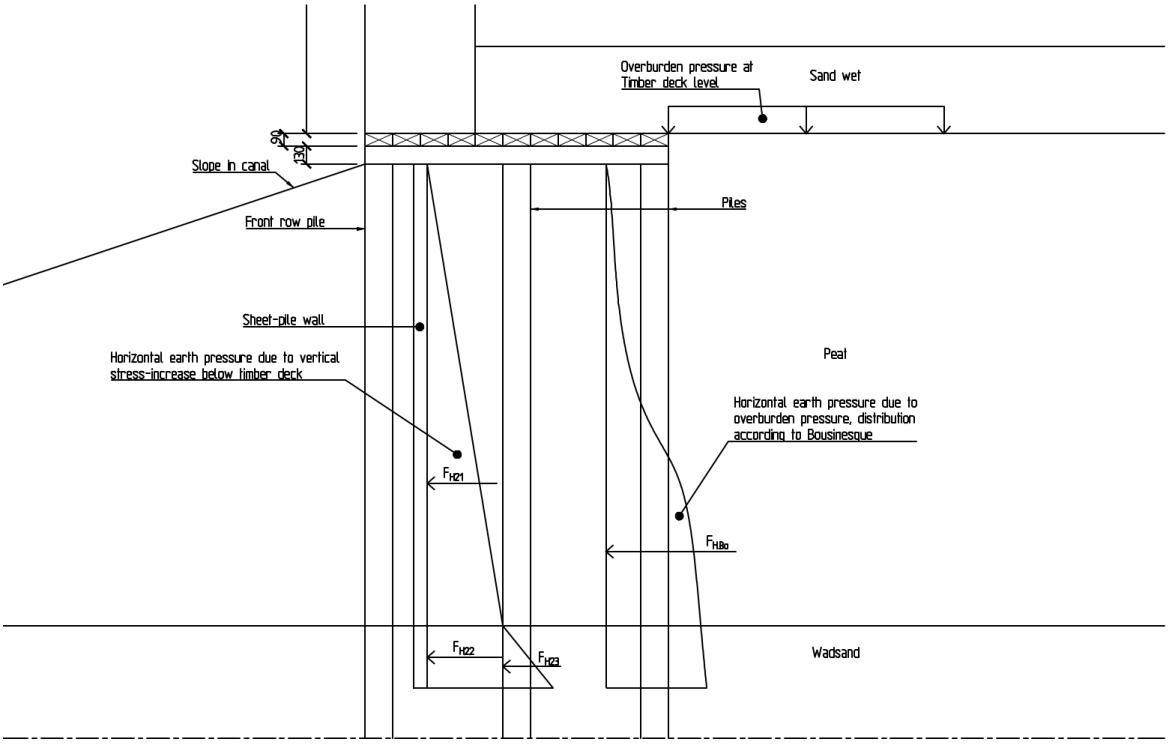


Figure 4.6: Interpretation of the Grimburgwal pile foundation with loads applied to the sheet-pile wall.

A Python-script was created, generating the geometry and applying loads to the structure in Abaqus, which provides the possibility to do a parametric study of the quay wall. The geometry of the structure can be seen in Figure 4.7 where the division of elasto-plastic springs (dz) are marked.

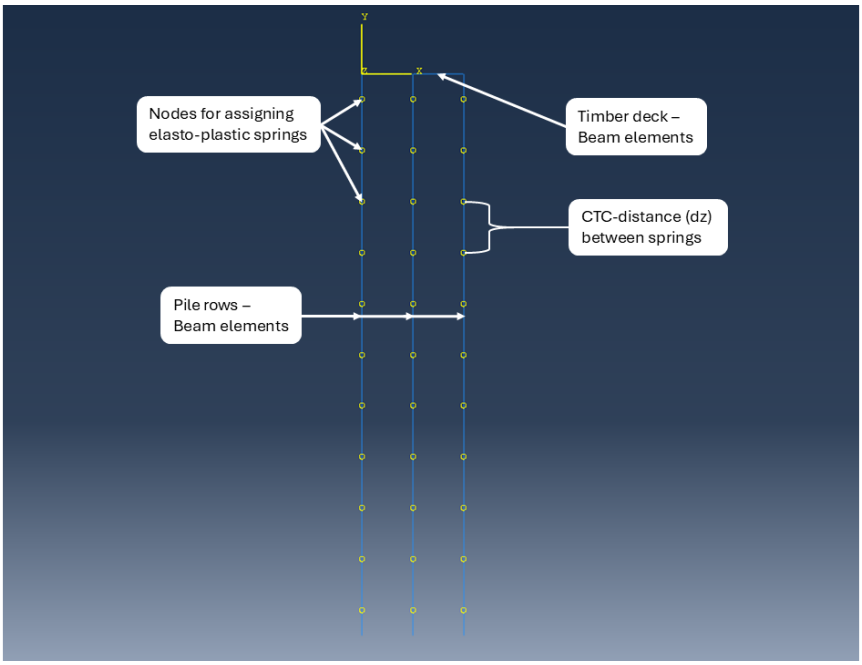


Figure 4.7: Structural model in Abaqus for benchmarking against the reference case.

4.1.1 Modelled geometry and material

The timber piles have, according to Hemel (2023), an elastic modulus of 7 GPa based on findings from bending tests of existing timber piles. The timber deck is modelled with the same elastic modulus but not as a separate deck supported on the headstock, as shown in Figure 4.1. Instead a single beam element, representing both the headstock and the deck, is modelled with the same stiffness as the deck and headstock. How this single beam element is formulated can be seen in Appendix F.

4.1.2 Modelled loads

The load application in the FE-model can be seen in Figure 4.8.

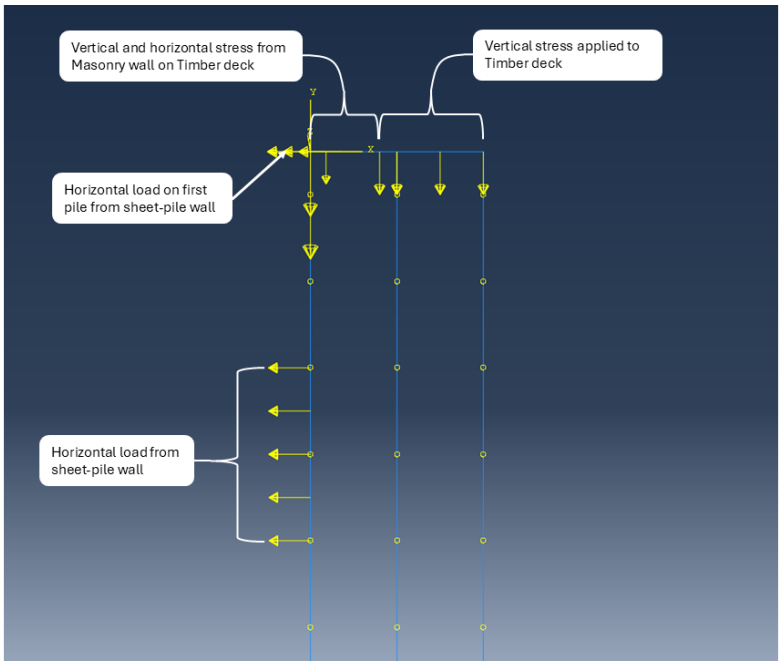


Figure 4.8: Load applications in Abaqus for the verification against the reference case.

The calculation of the load effects can be seen in Appendix F.

4.1.3 Modelling of spring properties

Non-linear springs are not possible to model directly in Abaqus with the spring to ground tool but can be implemented with an alternative method. The non-linear properties are written to the .inp file (the file from where the analysis is run) generated from the .cae-file (the file containing the Abaqus model). With the Python script, this is done automatically by primarily inserting "dummy-springs", one for each pile, in the .cae file and later replacing this data in the .inp file. The NL-springs are assigned to the correct nodal positions, at each modelled pile instance, with the corresponding plastic limit and elastic stiffness according to Brinch-Hansen and Menard, at the corresponding depths.

4.2 Stress comparison: Without soil wedge reduction

Initially, stresses in the piles are analysed without incorporating the reduced plastic limit of the springs with the soil wedge model. The soil wedge, at failure, is free to develop with

a shape according to Figure 4.9 when a fanning angle ,φ_m, for undrained soil conditions in clay and the assumption of loose sand is studied. This is modelled by setting φ_m to the internal friction angle divided by 5 for the clay layers and divided by 3 for the sand layers (Hemel et al., 2022).

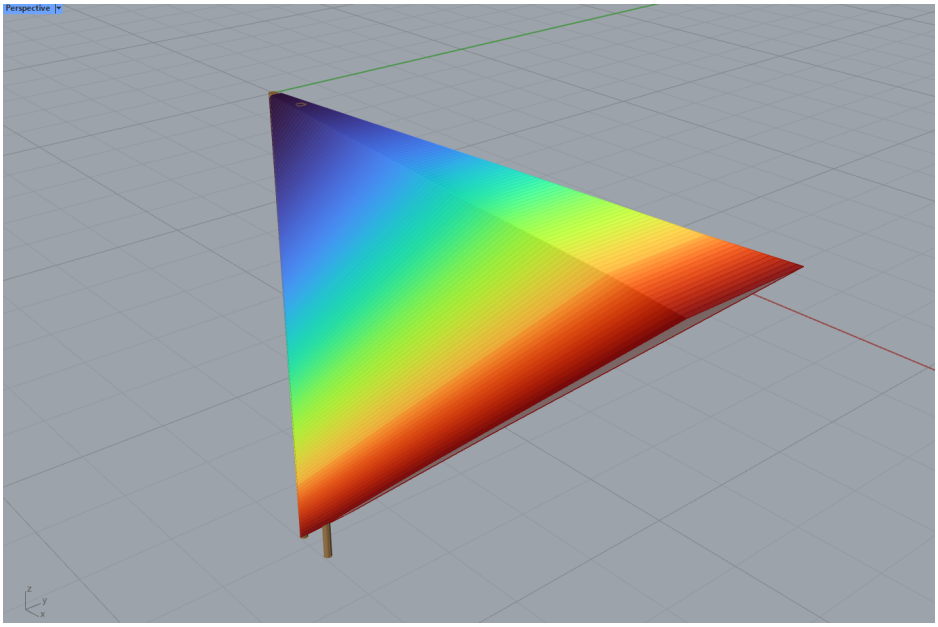


Figure 4.9: Contour wedges volumes for Pile 1, not including group effects or geometrical boundaries (unreduced). Each wedge representing a spring.

The soil vertical stress field is assumed to start at the ground level, compare Figure 4.4, and no non-linear geometry (NL-geometry) is applied. The lateral spring at the top of pile have a plastic stress limit determined at depth $2.6\,\mathrm{m}$ with this assumption.

Running the script produce primarily two set of results with pile diameter of $0.2\,\mathrm{m}$ and $0.25\,\mathrm{m}$. Each set have a varied dz from $1\,\mathrm{m}$ down to $0.1\,\mathrm{m}$ to evaluate the effect of spring spacing. 20 analyses are run in total for this comparison.

The resulting vonMises stress in the front-row pile for two pile rows, with diameter $0.2\,\mathrm{m}$ and $0.25\,\mathrm{m}$, can be seen in Figure 4.10 and 4.11 respectively.

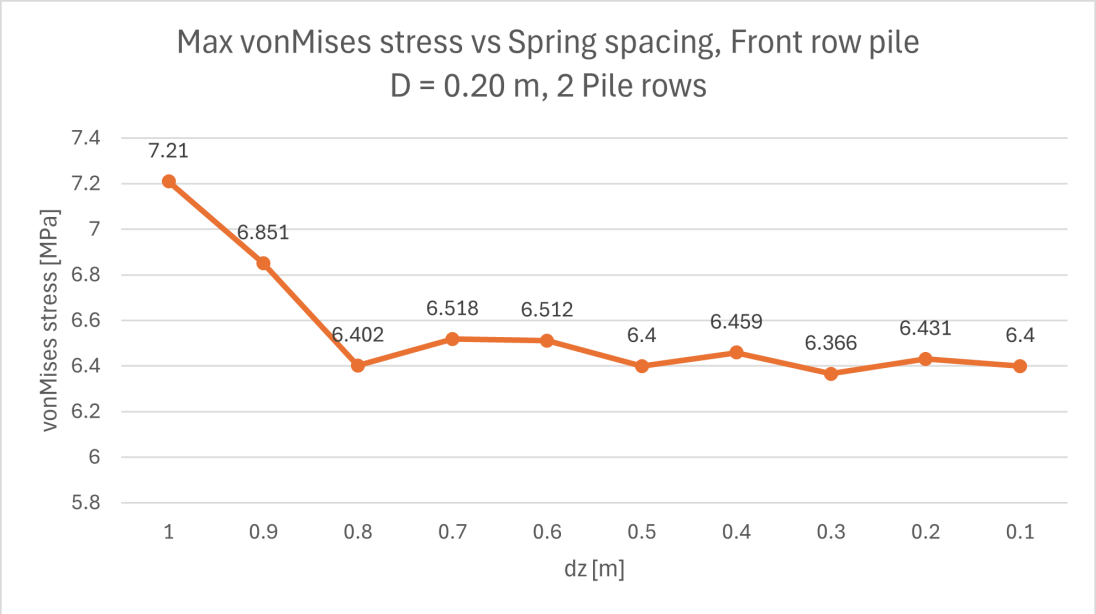


Figure 4.10: Maximum stress for D = $0.20\,\mathrm{m}$ with two pile rows, plotted against dz.

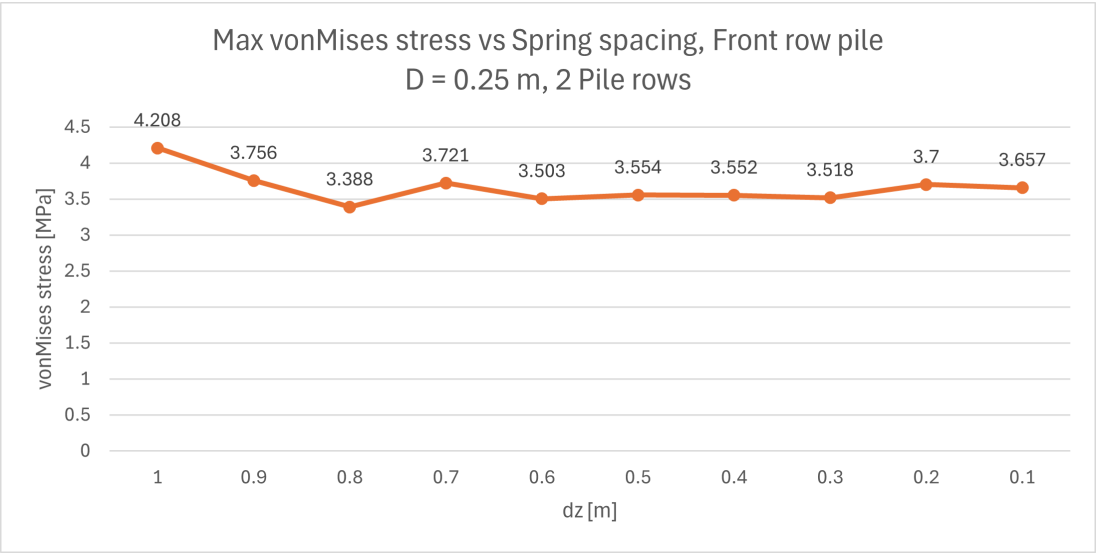


Figure 4.11: Maximum stress for D = $0.25\,\mathrm{m}$ with two pile rows, plotted against dz.

The maximum stress for D = $0.2\,\mathrm{m}$ lies between $7.2\,\mathrm{MPa}$ and $6.4\,\mathrm{MPa}$, and for D = $0.25\,\mathrm{m}$ between $4.2\,\mathrm{MPa}$ and $3.7\,\mathrm{MPa}$.

From the results we can conclude that the spring spacing have a large effect on the resulting stresses. Evaluating the results where D = $0.2\,\mathrm{m}$, from Figure 4.10, the stress drops from $7.2\,\mathrm{MPa}$ to $6.4\,\mathrm{MPa}$ when changing the spring spacing from $1\,\mathrm{m}$ down to $0.8\,\mathrm{m}$. Decreasing the spacing further to $0.1\,\mathrm{m}$ gives a scatter in obtained max stress between $6.4\,\mathrm{MPa}$ and $6.5\,\mathrm{MPa}$, concluding that the stress is converging.

An evaluation of stresses when D = $0.25\,\mathrm{m}$, from Figure 4.11, shows that a decrease of dz from $1\,\mathrm{m}$ to $0.8\,\mathrm{m}$ reduce the stress from $4.2\,\mathrm{MPa}$ to $3.4\,\mathrm{MPa}$. Further decreasing the

spacing leads to a scatter between 3.4 and $3.7\,\mathrm{MPa}$. The difference in maximum stress between the largest-, and lowest spacing is $24\,\%$, while only comparing a spacing between $0.7\,\mathrm{m}$ and $0.1\,\mathrm{m}$ gives a difference of $6\,\%$. From these comparisons we can conclude that a spring spacing of $0.7\,\mathrm{m}$, or less, leads to stress-convergence for the larger pile diameter. Following up, two new sets of analysis are run, both with a pile diameter of $0.2\,\mathrm{m}$ but with three and four pile rows respectively. As previously, each set was evaluated by varying dz from $1\,\mathrm{m}$ to $0.1\,\mathrm{m}$. The results can be seen in Figures 4.12 and 4.13 for three and four pile rows respectively.

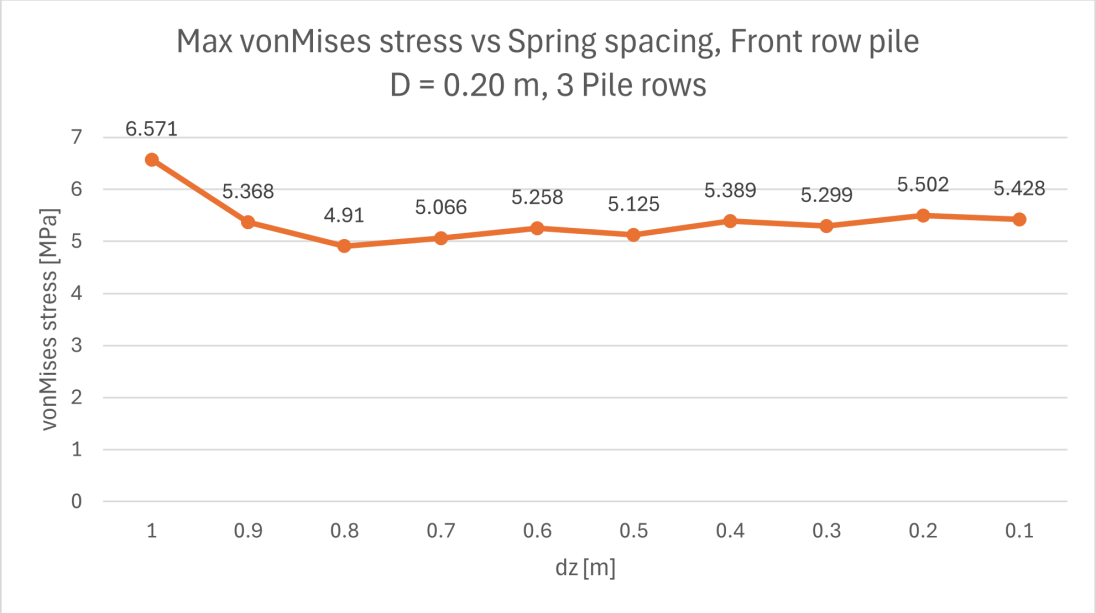


Figure 4.12: Maximum stress for D = $0.20\,\mathrm{m}$ with three pile rows, plotted against dz.

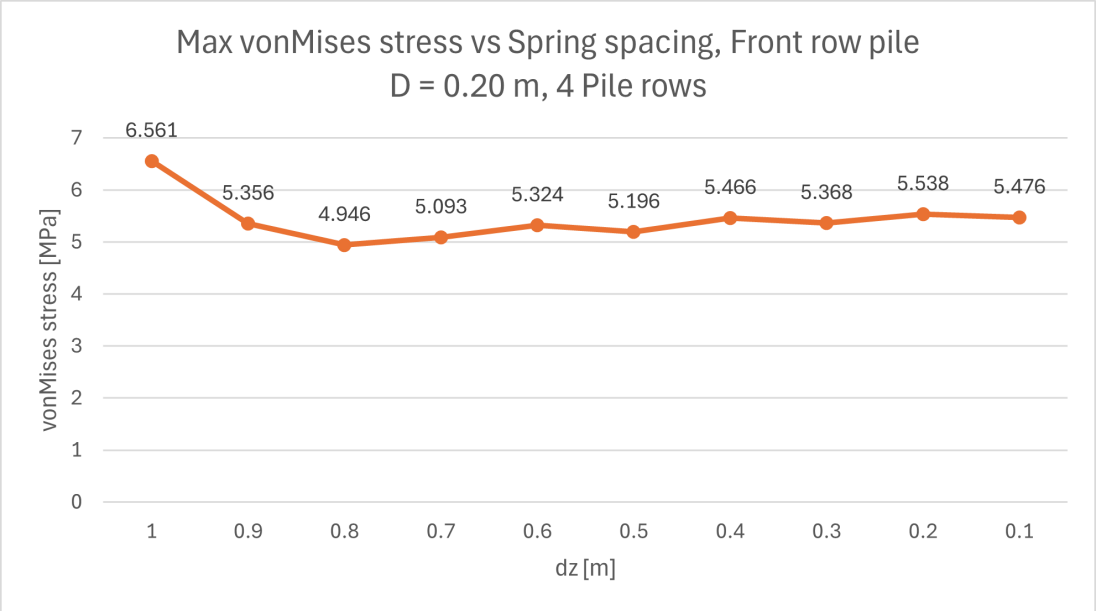


Figure 4.13: Maximum stress for D = $0.20\,\mathrm{m}$ with four pile rows, plotted against dz.

The results show the same dependency on the dz, for the resulting stress, as when analysing

two pile rows. Increasing the amount of piles from two to three have an effect on the stress in the piles, decreasing to around $6.6\,\mathrm{MPa}$ and $5.4\,\mathrm{MPa}$ with a dz of $1\,\mathrm{m}$ and $0.1\,\mathrm{m}$ respectively.

Comparing the obtained stress against the Grimburgwal reference case, where the stress was found to be $5.1\,\mathrm{MPa}$ and $3.0\,\mathrm{MPa}$ for three and four pile rows respectively, it is higher from the Abaqus-model. This is due to the horizontal stress against the sheet pile wall being overestimated, since the surcharge loading from the soil above the timber deck is assumed to start behind the second pile row.

This assumption is strengthened by evaluating the displacement curves for pile diameter $0.20\,\mathrm{m}$ with two up to four pile rows, seen in Figures 4.14 and 4.15, where the displacements at the location of the applied sheet-pile load is more distinctive for three and four pile rows compared to two pile rows.

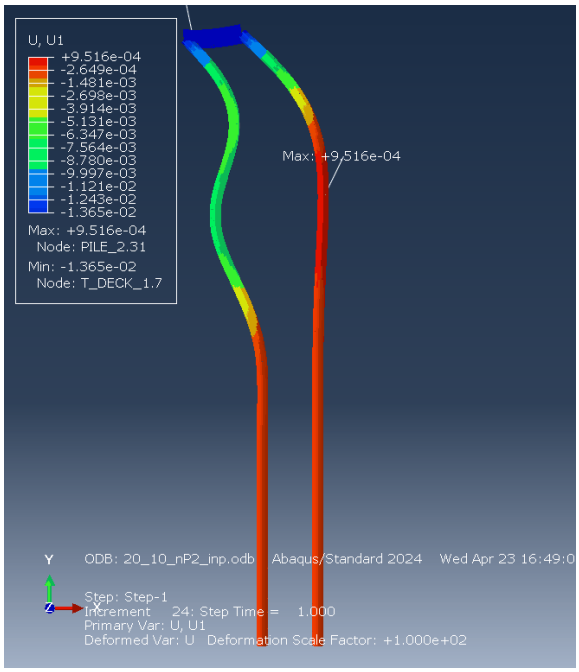


Figure 4.14: Displacement curves, Abaqus model, unreduced spring stiffness, D = $0.20\,\mathrm{m}$, two pile rows.

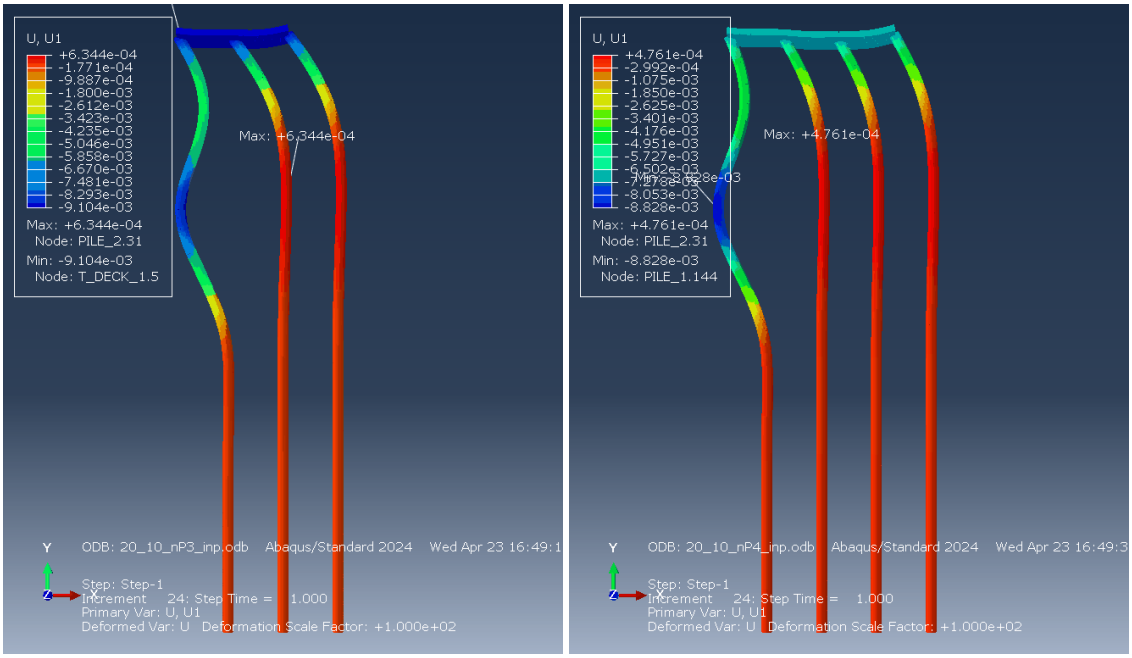


Figure 4.15: Displacement curves, Abaqus model, unreduced spring stiffness, D = $0.20\,\mathrm{m}$, three to four pile rows.

This problem is solved, in the following comparisons, by calculating the vertical stress field with the surcharge load applied behind the back pile-row. Concluding from these benchmarks that the computational time is fast, and that a smaller dz than $0.7\,\mathrm{m}$ shows convergence when analysing the stresses, a dz of $0.1\,\mathrm{m}$ is chosen for the following comparisons, that implement the soil wedge reduction.

4.3 Stress comparison with soil wedge reduction: Initial model

The effect of the reduced soil-wedges is studied in this section. Initially, a fanning angle, φ_m, for undrained soil conditions in clay and assuming loose sand is studied. φ_m is set to the internal friction angle divided by 5 for the clay layers and divided by 3 for the sand layers (Hemel et al., 2022).

These reduced soil wedges, produced with Grasshopper, can be seen in Figure 4.16 and Figure 4.17, for the front- and back-row pile respectively. It's clear that the wedges are smaller, compared to the idealized wedge in Figure 4.9, leading to a reduced plastic limit of the lateral soil springs. The different colours represent the different soil layers shown in Table 4.2.

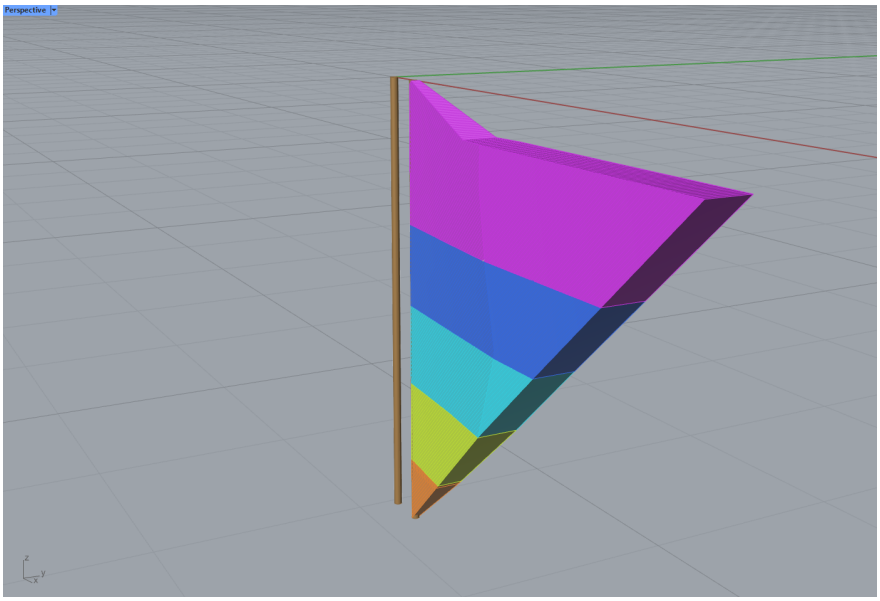


Figure 4.16: Reduced wedges for the front-row pile, colours indicate different soil layers.

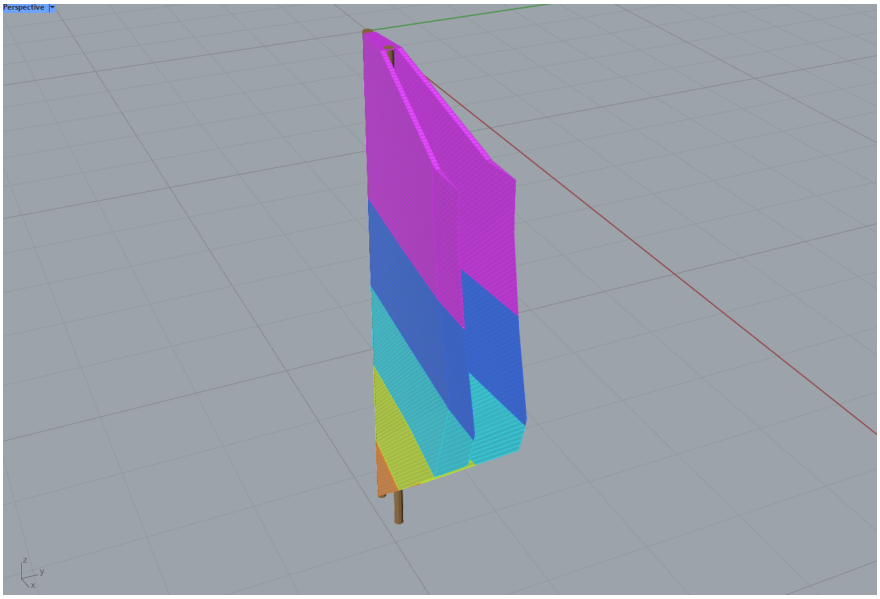


Figure 4.17: Reduced wedges for the back-row pile, colours indicate different soil layers.

The obtained reduction factors, for the initial soil wedge model, are shown in Table 4.3. Pile 2 is closest to the canal and Pile 1 is valid for the trailing pile rows.

Table 4.3: Obtained reduction factors, initial wedge model.

	Pile 1		**Pile 2**	
Depth [m]	Ψ_w	Ψ_c	Ψ_w	Ψ_c
0.1	1	1	0.885	0.973
1	0.854	0.940	0.685	0.689
2	0.491	0.519	0.641	0.642
3	0.351	0.617	0.617	0.618

The comparison was conducted with different combinations of parameters, studying the effect from applying non-linear geometry (NL-geometry), and overburden pressure (OP), when determining the plastic spring limit.

Including overburden pressure is made with the same assumption as the comparison with the unreduced wedges. Not including it is done by setting the depth to $0\,\mathrm{m}$, at the top of the piles, when calculating the vertical effective stress, affecting the plastic limit of the springs. This can be seen as an upper-, and lower-bound solution, when determining the spring-stiffness. The assumption that the effective stress field beneath the timber deck is affected by the soil layers above can be seen as the upper-bound solution.

The resulting maximum stress, for each combination of parameters, can be seen in Table 4.4 for the front-, and second pile row.

Table 4.4: Stress comparison with initial soil wedge reduction model.

Diameter [m]	**Pile rows** [2-4]	**OP** [ON/OFF]	**NL-geometry** [ON/OFF]	**Stress, Front pile** [MPa]	**Stress, Pile 2** [MPa]
0.20	2	ON	OFF	6.4	4.37
0.20	2	OFF	OFF	7.00	5.02
0.20	2	OFF	ON	7.25	5.18
0.25	2	ON	OFF	3.66	2.86
0.25	2	OFF	OFF	3.74	2.94
0.25	2	OFF	ON	3.80	2.97
0.20	3	ON	OFF	4.06	3.89
0.20	3	OFF	OFF	4.09	3.41
0.20	3	OFF	ON	4.21	3.52
0.20	4	ON	OFF	3.14	2.74
0.20	4	OFF	OFF	3.14	2.74
0.20	4	OFF	ON	3.23	2.82

The results show that changing the location of the surcharge load, when determining the vertical stress field for three and four pile rows, leads to a stress decrease in the front-row pile. The maximum stress decrease from $5.4\,\mathrm{MPa}$ and $5.5\,\mathrm{MPa}$ to $4.1\,\mathrm{MPa}$ and $3.1\,\mathrm{MPa}$ for three and four piles respectively. See Figure 4.18.

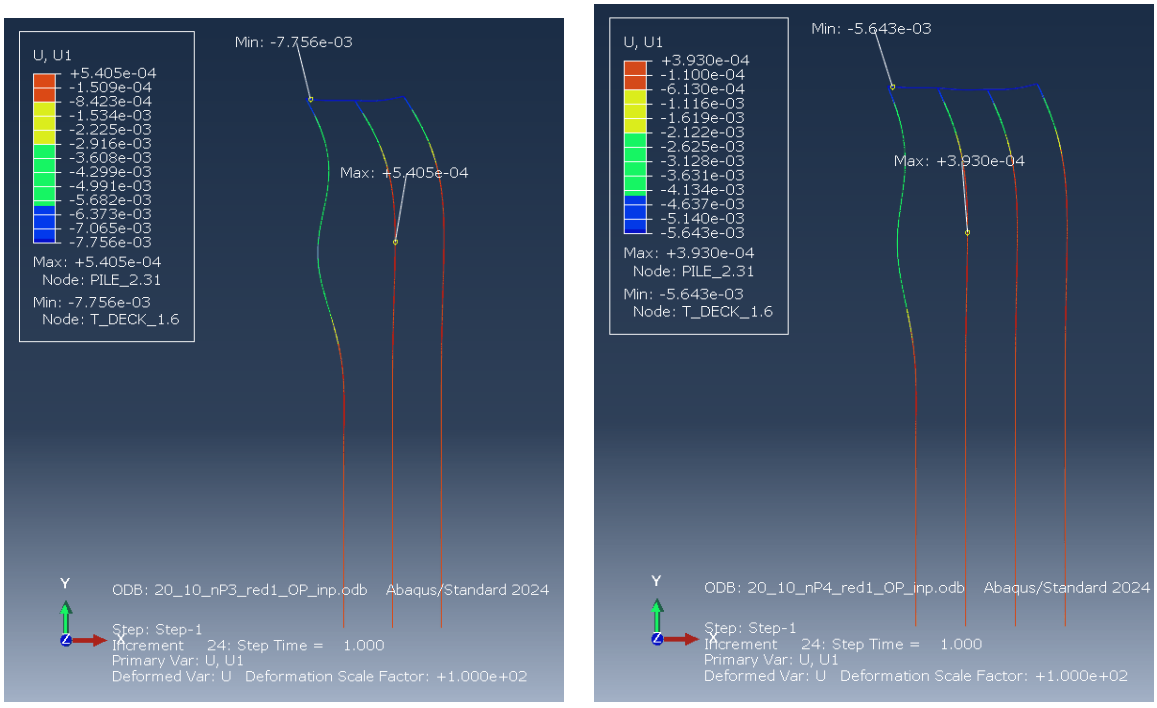


Figure 4.18: Stress in piles from Abaqus model with initial reduction model, D = $0.20\,\mathrm{m}$, including overburden pressure. Left: Three piles, Right: Four piles.

A contraindicative observation is that the maximum stress, for two pile rows, did not change even though the plastic limit was reduced with the soil wedge model. Including, or not including, overburden pressure affects the maximum stress in the piles. The same conclusion

is drawn regarding the effect of NL-geometry. Both conditions have a larger impact when fewer, and smaller piles are used.

The resulting stress distribution, using two pile rows with D = $0.20\,\mathrm{m}$, can be seen in Figure 4.19 for the three different combinations of parameters.

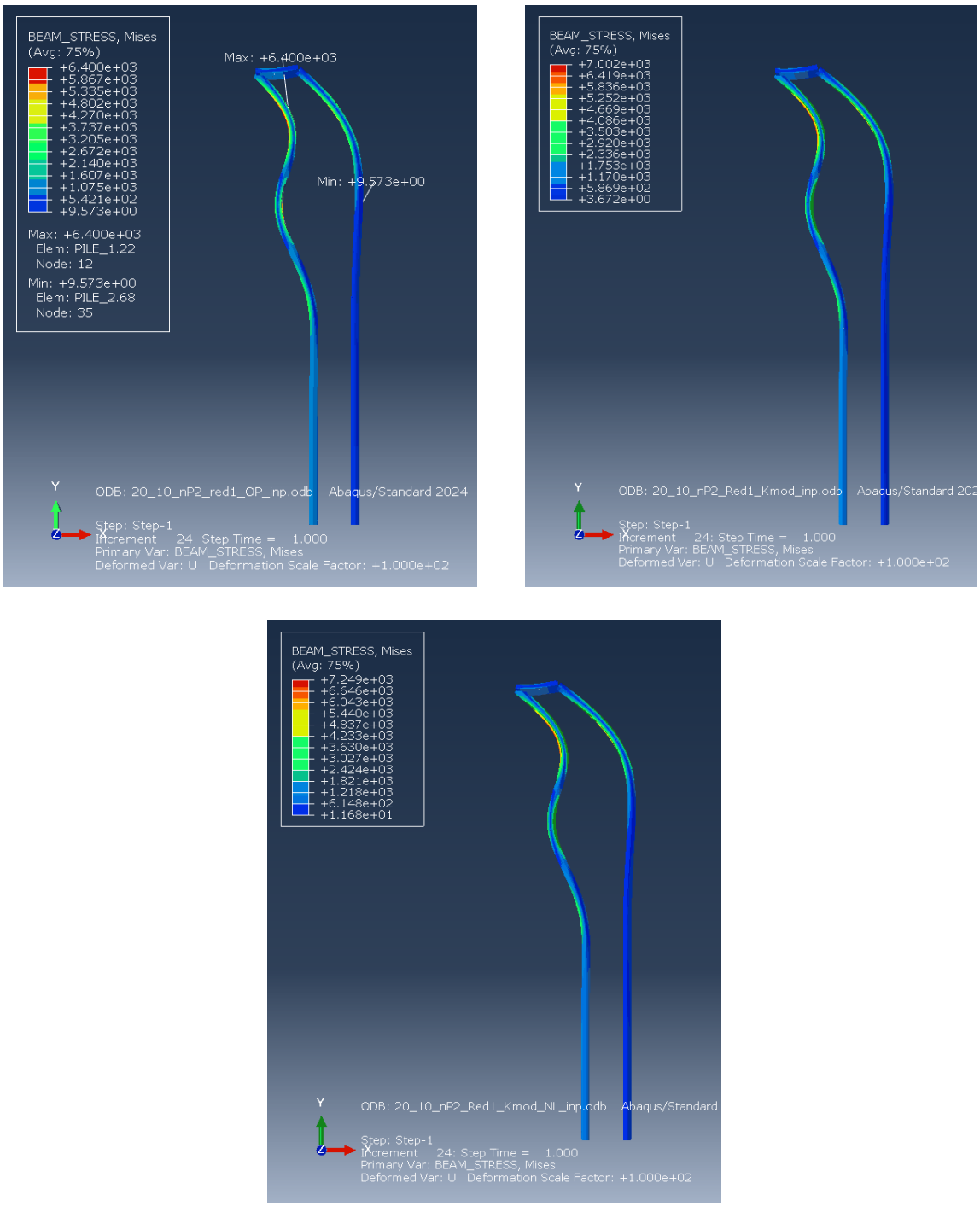


Figure 4.19: Stress in piles from Abaqus model with initial reduction model, D = $0.20\,\mathrm{m}$, two pile rows. From top left and clockwise, the setup from row one to row three in Table 4.4 is shown.

4.4 Stress comparison with soil wedge reduction: Refined model

Since implementing the initial reduction model, assuming undrained condition for the clay and loose sand for the granular layers, had a small effect on the resulting stresses, a refined soil wedge model is analysed. By investigating the Matlab-model created by Hemel (2023) and reading the paper more thoroughly, a fanning angle equal to the internal friction angle was chosen for both the clay and granular layers. This leads to a bigger wedge, producing larger reduction factors, that lowers the plastic limit of the springs.

The unreduced and reduced wedge for the refined reduction model can be seen in Figure 4.20, showing that the reduction is larger compared to the initial reduction model.

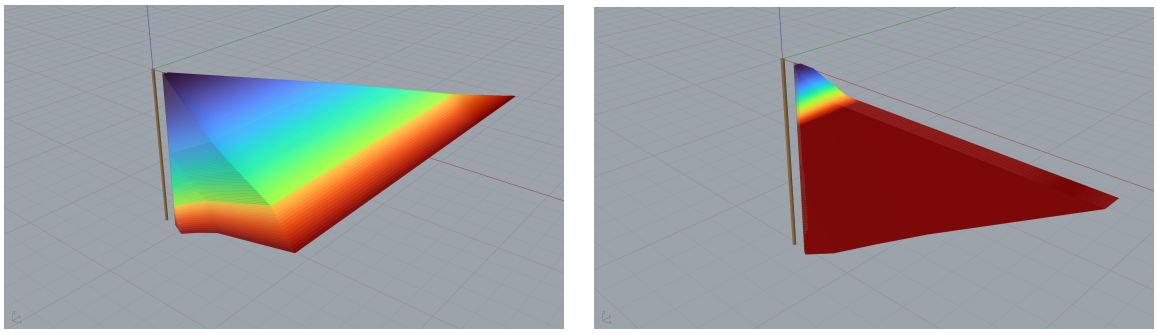


Figure 4.20: Volume comparison of soil wedges contours of Pile 2, Left: Unreduced, Right: Reduced.

Another comparison is shown in Figure 4.21, illustrating how an isolated wedge from a trailing pile intersects with the soil from an obstructing pile.

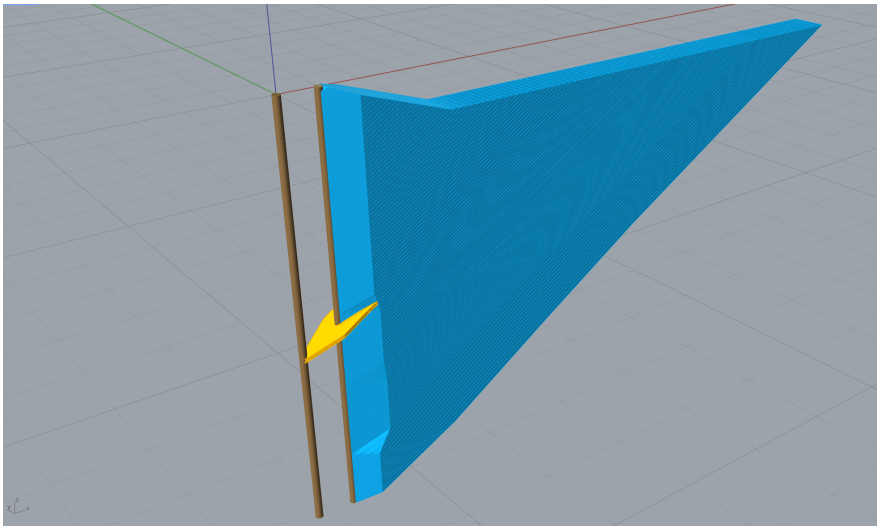


Figure 4.21: Reduced soil wedge from front-pile, interacting with the wedge from the trailing pile.

For the refined wedge model, obtained reduction factors are shown in Table 4.4, where Pile 2 is closest to the canal and Pile 1 is valid for the trailing pile rows.

Table 4.5: Obtained reduction factors, refined wedge model.

	Pile 1		**Pile 2**	
Depth [m]	Ψ_w	Ψ_c	Ψ_w	Ψ_c
0.1	1	1	0.707	0.813
1	0.461	0.497	0.421	0.433
2	0.135	0.141	0.269	0.274
3	0.0623	0.065	0.203	0.204

Analysis have been conducted for two up to four pile rows, including or not including overburden pressure when determining the plastic spring limit, same as for the stress comparison with the initial reduction model. NL-geometry was applied for all analysis. The resulting stresses, for each combination of parameters, can be seen in Table 4.6.

Table 4.6: Stress comparison with refined soil wedge reduction model.

Diameter [m]	**Pile rows** 2-4	**OP** [ON/OFF]	**Stress, Front pile** [MPa]	**Stress, Pile 2** [MPa]
0.20	2	ON	6.57	4.46
0.20	2	OFF	7.00	6.00
0.25	2	ON	3.71	2.89
0.25	2	OFF	3.86	3.05
0.20	3	ON	4.18	3.49
0.20	3	OFF	4.26	3.53
0.20	4	ON	3.23	2.82
0.20	4	OFF	3.23	2.82

When considering NL-geometry and the impact of overburden pressure, the stress in both the front-row and back-row piles (with a diameter of $0.2\,\mathrm{m}$ and two pile rows) is similar to the initial reduction model but is slightly increased due to NL-effects. Turning of the effect of overburden pressure, on the other hand, leads to a decrease of stress in the front-row pile but increasing it for the trailing pile row. The displacements in x-direction, at the top of the timber deck, increase which can be seen in Figure 4.22.

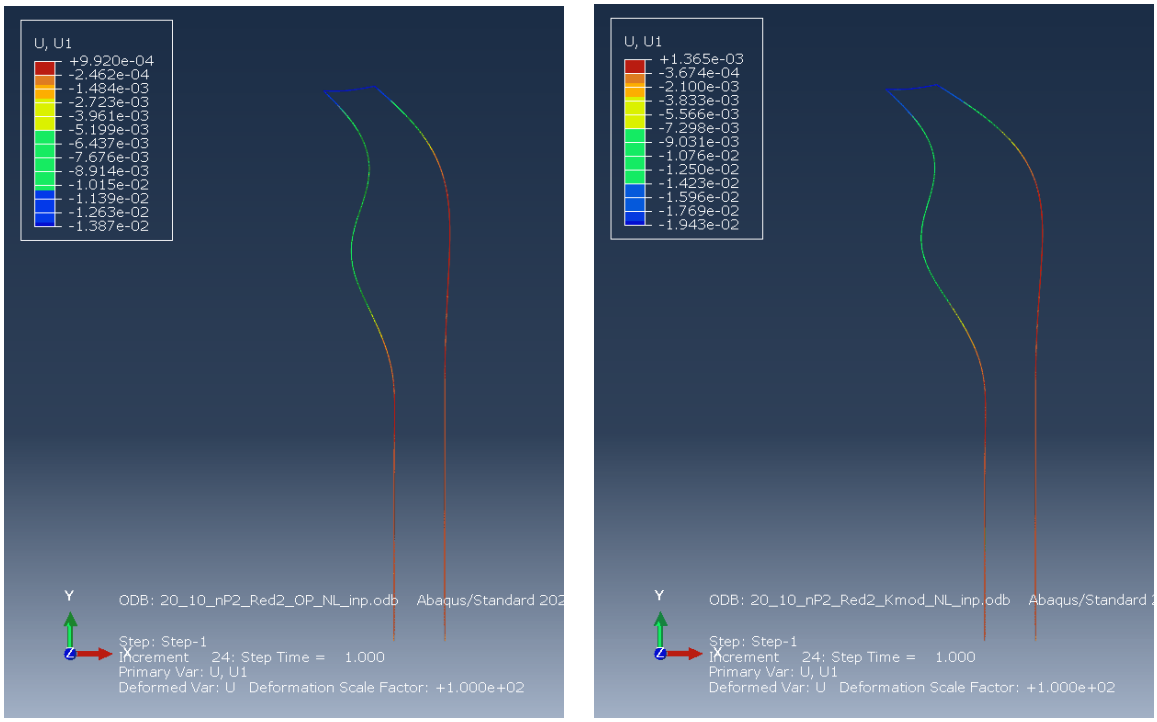


Figure 4.22: Displacement curves, Abaqus model, refined reduction model, D = $0.20\,\mathrm{m}$, including, or not including, overburden pressure.

Evaluating the results for three and four pile rows, with the same diameter, and two pile rows with a diameter of $0.25\,\mathrm{m}$, show no change of maximum stress when comparing the initial and refined reduction model. The small difference in displacements, including, or not including overburden pressure, for two pile rows with D = $0.25\,\mathrm{m}$, can be seen in Figure 4.23.

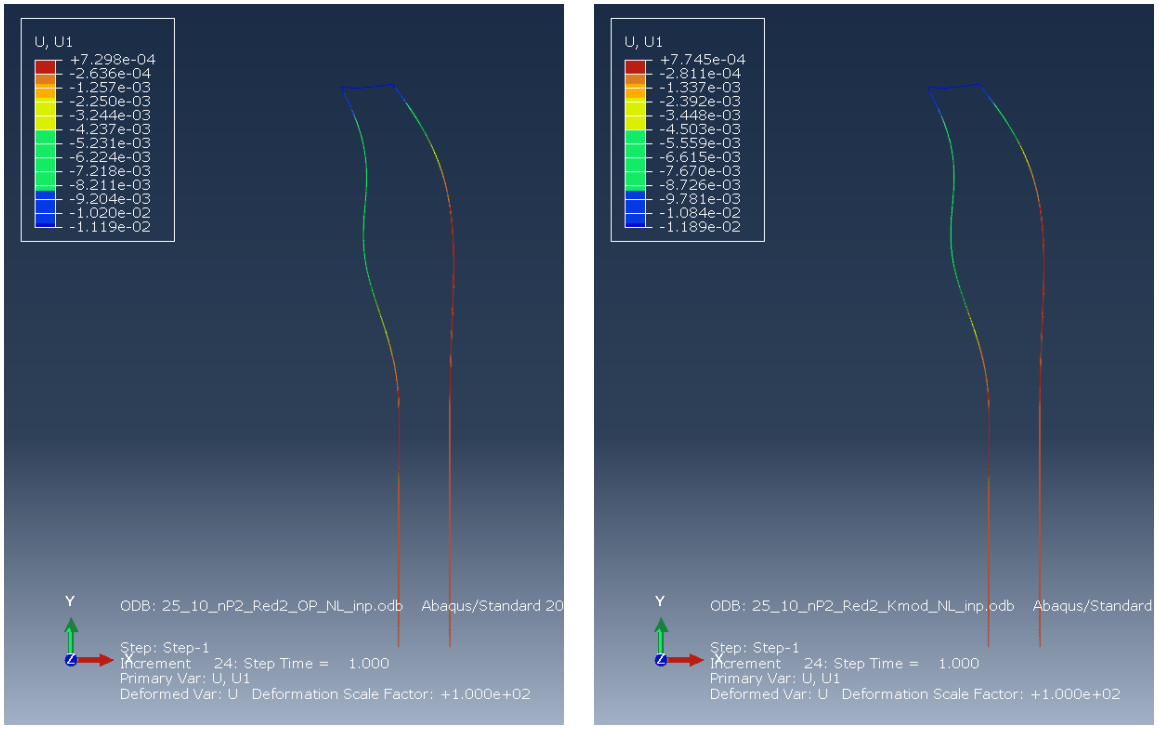


Figure 4.23: Displacement curves from Abaqus model with refined reduction model, D = $0.25\,\mathrm{m}$, including / not including overburden pressure.

From evaluating the figures we note that a decrease of lateral soil stiffness, in the FE-model, redistribute the stress to the trailing pile rows. When fewer and less stiff piles are used there is a clear change of stress distribution between the piles. When more, or stiffer piles, are used a decrease of soil stiffness is of less importance.

For all pile configurations, no result comply with the results obtained in the reference case of the Grimburgwal. The stress is, with the FE-model, lower for all compared cases except when four pile rows are analysed with a diameter of $0.2\,\mathrm{m}$.

4.5 Comparison: Abaqus - Grimburgwal

To summarize, the benchmarking of the FE-model against the Grimburgwal reference case, the stresses are compared in Table 4.7. Note that the results from the models with three and four pile rows that didn't implement the soil wedge reduction are not included, since these models overestimated the horizontal stress from the sheet-pile wall.

Table 4.7: Stress comparison, Reference case vs Abaqus model.

Diameter [m]	**Pile rows** [2-4]	**Grimburgwal** [MPa]	**Unreduced** [MPa]	**Initial** [MPa]	**Refined** [MPa]	**Difference** [%]
0.20	2	12.0	6.4	7.25	7.0	71.4
0.25	2	4.9	3.66	3.8	3.86	26.9
0.20	3	5.1	-	4.21	4.26	19.7
0.20	4	3.0	-	3.23	3.23	-7.1

4.6 Stress comparison: Hand-calculations

As a comparison, to the result from the FE-model, hand-calculations where conducted against the setup with two pile rows. The calculations are limited to static linear elastic analysis for the piles while the behaviour of the soil is studied both with and without assuming plastic soil behaviour. The input data is the same for all analysis and can be viewed in Table 4.8.

Table 4.8: Input values for comparison with hand-calculation.

Pile diameter	D	0.2 m / 0.25 m
Elasticity modulus	E	7 GPa
Compressive capacity	f_{ck}	11.2 MPa
Lateral stiffness	k	4632 kPa
Vertical force	V	29.1 kN
Horizontal force	H	8.6 kN

The lateral stiffness of the soil is assumed to be that of the peat layer, based on the findings by Hemel (2023), utilizing Equation 2.3. Young's modulus and compressive capacity of the pile is chosen according to the same author where it's also mentioned that the bending capacity of timber is 90 % of the compressive capacity.

The load distribution on the timber deck is shown in Figure 4.24 where the distribution of forces to the piles is calculated based on a linear-elastic distribution. This calculation can be seen in Appendix F.

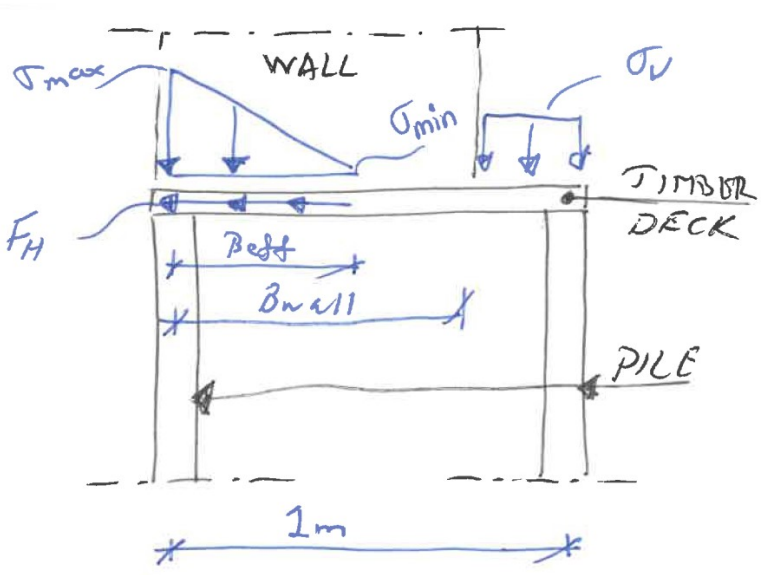


Figure 4.24: Load distribution to piles.

The stress that timber can carry before plastic behaviour occurs is different for bending and axial stress, stated by Hemel (2023) and in Eurocode 5 ("EN 1995-1-1", 2009). The utilisation rate, with this combined loading, is determined with Equation 4.1. The combined

stress, without considering whether it's originating from axial-, or bending stress, is evaluated with Equation 4.2. This is determined as an appropriate comparison against the finding by Hemel (2023) and from the FE-analysis conducted in this work.

$$u = \frac{\sigma_c}{k_c \times f_c} + \frac{\sigma_m}{f_m} \tag{4.1}$$

$$\sigma = \frac{M}{W} + \frac{V}{k_c \times A} \tag{4.2}$$

Instability effects are included, based on Eurocode 5, by reducing the axial capacity of the pile with the factor k_h, calculated according to Equation 4.3, 4.4, and 4.5 ("EN 1995-1-1", 2009).

$$k_c = \frac{1}{k + \sqrt{k^2 - \lambda_{rel}^2}} \tag{4.3}$$

$$k = 0.5 \times (1 + \beta_1 \times (\lambda_{rel} - 0.3) + \lambda_{rel}^2) \tag{4.4}$$

$$\lambda_{rel} = \frac{\lambda}{\pi} \times \sqrt{\frac{f_{ck}}{E}} \tag{4.5}$$

The buckling length for a pile considered rigid at a certain depth L is determined by the expression $L_c = \beta L$, where β depends on the buckling mode. For a column or beam considered rigid in one end and free in the other $\beta = 2.25$ ("Design of timber structures - Volume 1", 2016).

4.6.1 Swedish handbook *Pålgrundläggning*

Initially, the theories in the Swedish handbook *Pålgrundläggning* are studied. A pile standing free in water can at a certain embedded depth in the soil be considered rigid due to the lateral soil restraint, visualized in Figure 4.25.

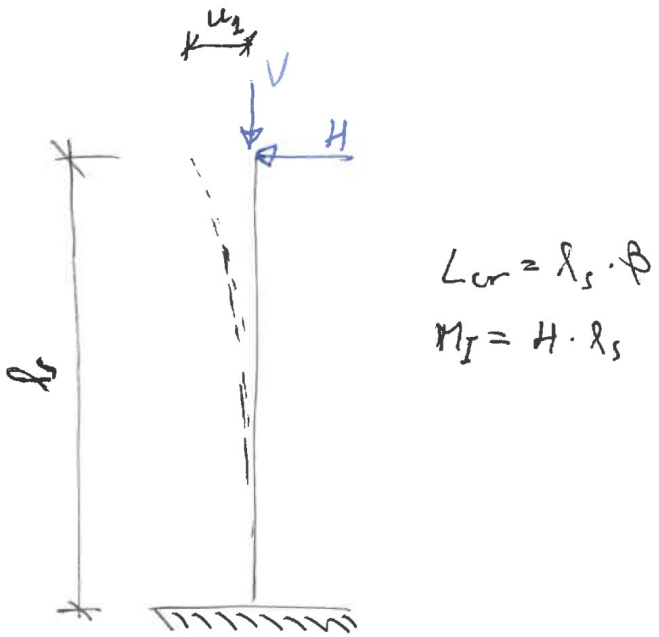


Figure 4.25: Static model of pile embedded in soil, considered rigid at base.

The length until the pile is considered rigid at the base can be determined according to Equation 4.6, considering the stiffness and diameter of the pile (EI and D) and the lateral spring stiffness off the soil (k) (Olsson & Holm, 1993).

$$l_s = 1.4 \times^4 \sqrt{\frac{E \times I}{k \times D}} \tag{4.6}$$

This results in a free length, l_s, of $0.822\,\mathrm{m}$ and from this the stress in the pile, due to the horizontal force, can be calculated as well as the reduction factor due to buckling, k_c. The bending moment, determined as $H \times l_s$, becomes $7.06\,\mathrm{kNm}$ and the reduction factor 0.958. Determining the utilisation rate according to Equation 4.1, using characteristic loads and timber capacity, gives us a utilisation of $93\,\%$ and $60\,\%$ for a pile diameter of $0.2\,\mathrm{m}$ and $0.25\,\mathrm{m}$ respectively. The combined stress, obtained according to Equation 4.2, becomes $9.37\,\mathrm{MPa}$ and $6.13\,\mathrm{MPa}$ for the respective diameters.

4.6.2 Broms' extension method

Utilizing the methods presented by Cecconi et al. (2019), the maximum horizontal load that the pile can carry is investigated, along with the bending moment obtained in the pile under actual loading conditions. The required length to obtain horizontal equilibrium, L_{eq}, is solved for.

The maximum horizontal force that the pile can resist is determined using Equation 2.13. This is primarily achieved by solving Equation 2.14, where the maximum bending moment is set equal to the bending capacity of the pile. With this assumption, no additional vertical load can be carried by the pile.

If the horizontal force is arbitrary, the maximum bending moment in the pile can be solved by combining the previous Equations. The reduction factor due to buckling can then be calculated based on L_{eq} to determine the combined axial-, and bending stress according to Equation 4.2. With the characteristic horizontal load of $8.6\,\mathrm{kN}$, the maximum bending moment was determined as $2.51\,\mathrm{kNm}$ and L_{eq} as $0.58\,\mathrm{m}$. The obtained reduction factor is negligible for this pile length and the combined stress, according to Equation 4.2, becomes $4.12\,\mathrm{MPa}$.

4.6.3 Stress comparison: Including Hand-calculations

To include the results from the hand-calculations they are compared against the FE-model and reference case in Table 4.9.

Table 4.9: Stress comparison, Including hand-calculation

Grimburgwal MPa	**Unreduced** MPa	**Initial** MPa	**Refined** MPa	*Pålgrundläggning* MPa	**Broms' extension** MPa
12.0	6.4	7.25	7.0	9.37	4.12

5 Case study 2 - Gothenburg quay walls

With the model from Case study 1, a parametric study is conducted on the Gothenburg quay walls. A sensitivity study is made on the results from the parametric model to determine how different parameters affect resulting stresses in the pile foundation.

The structural layout of the historical quay walls in Gothenburg is, as previously described, varying. The layout used for Case study 2 is shown in Figure 5.1.

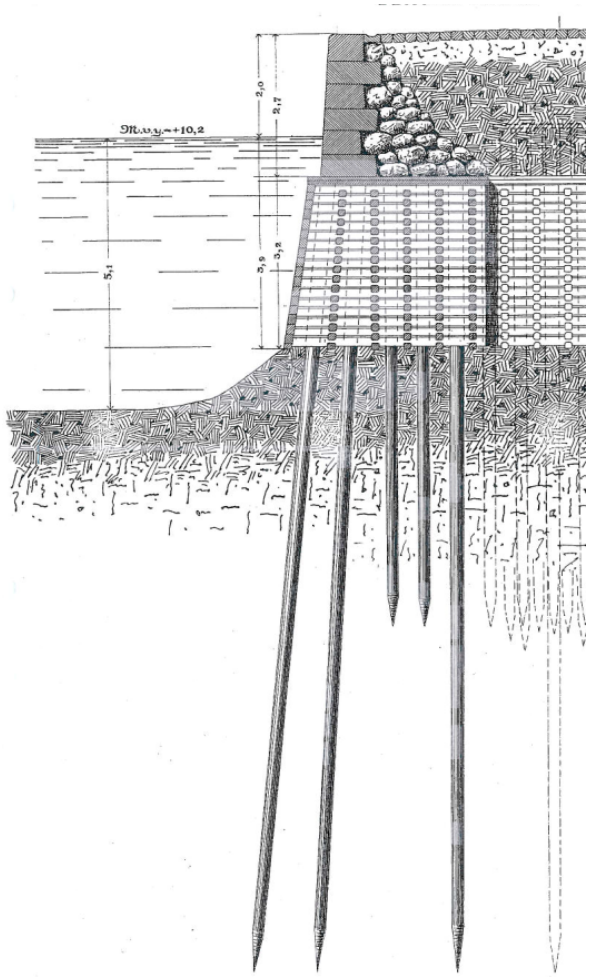


Figure 5.1: Drawing of an old quay wall built in Gothenburg. Source: City of Gothenburg, provided by Sweco AB, April 04, 2025.

The quay walls consist, similar to the reference case of the Grimburgwal, of a masonry structure supported on a deck and piles in timber. The piles are encased in a rock cofferdam, one for each pile creating a timber-frame structure. Against the canal, a timber panel is protecting the filling material inside the cofferdam from erosion. The panel is assumed to have been attached to the timber-frame when the wall was built.

Behind the timber deck, supporting the quay wall, there is assumed to be an embankment piling intended to ensure geotechnical stability and avoid excessive horizontal stress to affect the quay wall foundation.

5.1 Assumptions for the model

The structural representation of the assumed Gothenburg quay wall implemented in the models is shown in Figure 5.2. The geometry is based on old drawings found on existing historical quay walls around the Gothenburg canals.

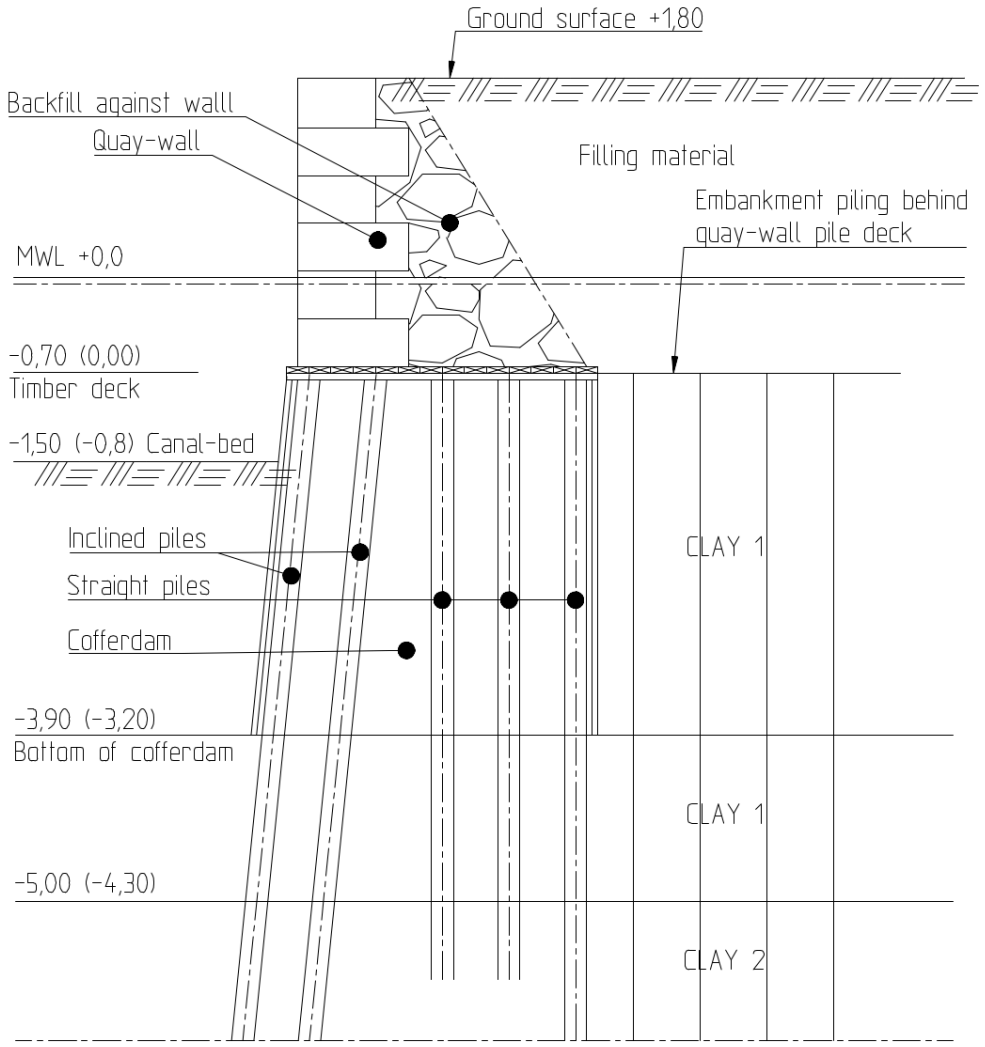


Figure 5.2: General section of the studied Gothenburg wall.

The assumed level of the ground surface, mean water level (MWL) etc., is shown as well as the corresponding depths when assuming the timber deck as the origin (noted in parenthesis) which will be the case when creating the soil-wedge-, and FE-model. The MWL is assumed to be located at +0,00. Because long term effects are studied, the ground water level behind the wall is assumed at the same depth. Hydrostatic water pressure is assumed to have been developed leading to no water pressure acting as an overturning load effect.

A more detailed representation of the quay wall structure can be seen in Figures 5.3 and 5.4, showing the superstructure and foundation respectively.

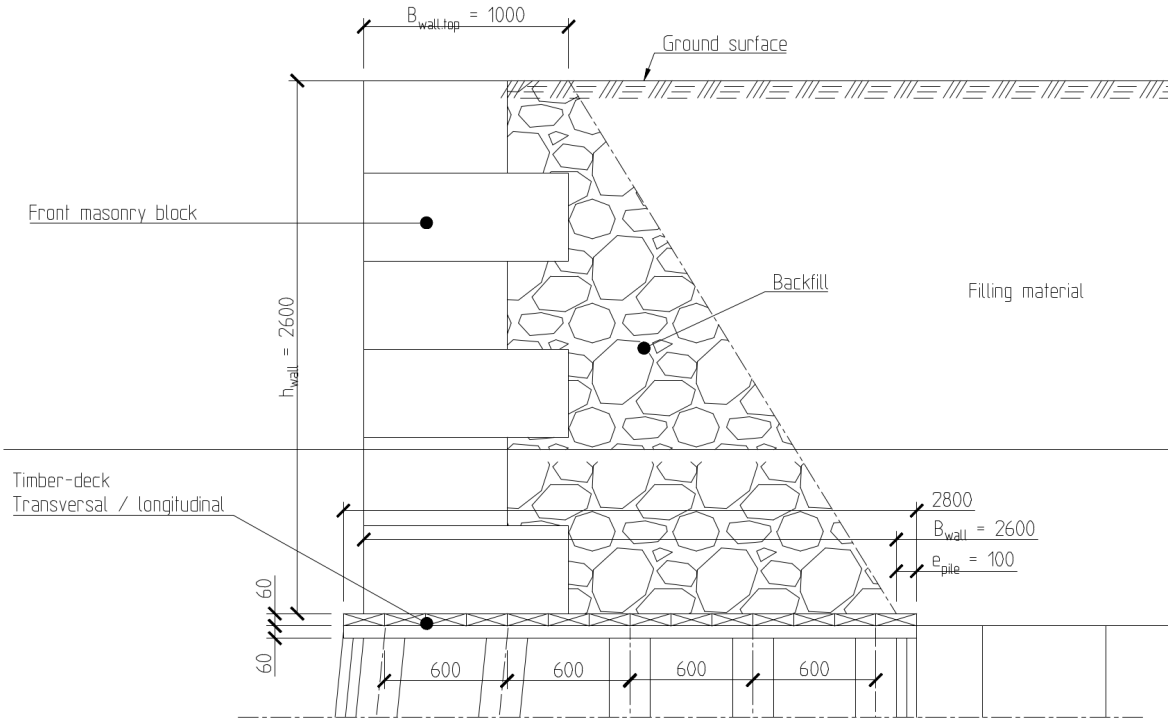


Figure 5.3: Superstructure of the Gothenburg wall.

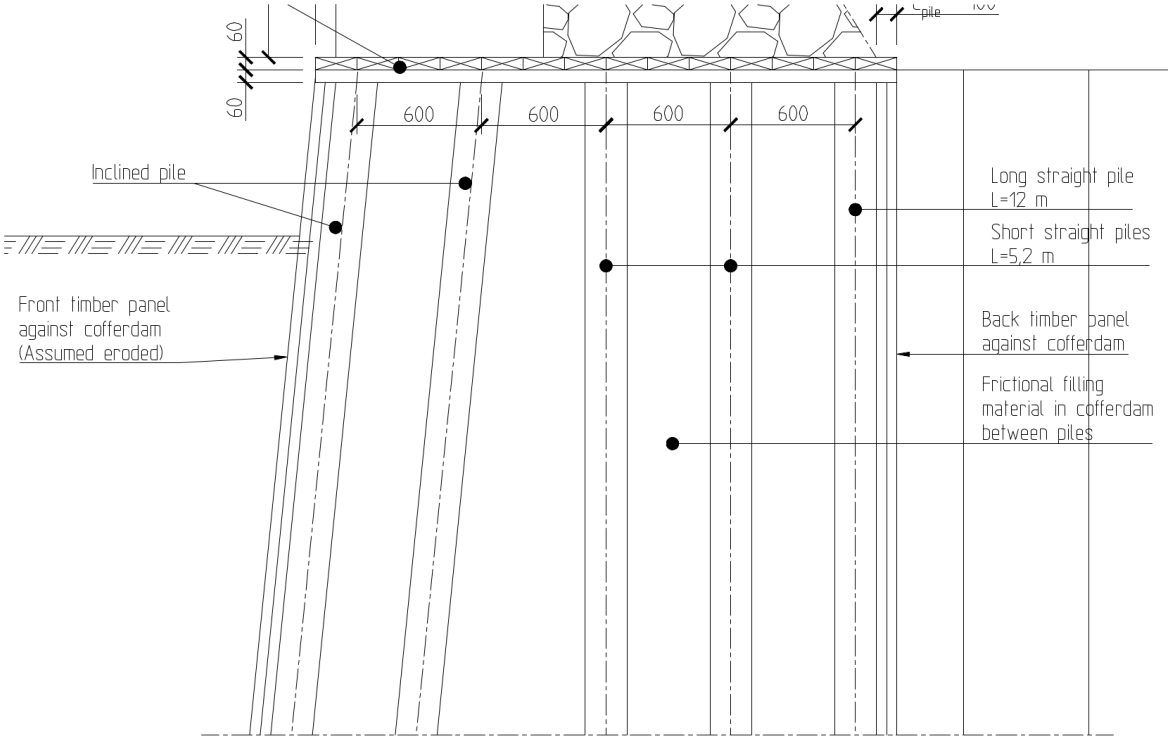


Figure 5.4: Layout of the pile foundation and canal bed of the Gothenburg wall.

The exterior front surface of the wall is assumed to be made out of stone masonry supported with a backfill. The wall is $2.6\,\mathrm{m}$ wide at the base and the timber deck protrudes $0.1\,\mathrm{m}$ on

each side. The timber deck is assumed to be made out of two layers of timber beams, arranged in the longitudinal and transversal direction. The spacing of the piles is determined to be around $0.6\,\mathrm{m}$ and five piles are assumed to support the quay wall. Out of these, the two closest piles to the canal are assumed to be inclined while the rest are straight piles with varying length. The exact inclination of the piles is unknown but an effort to measure it in old drawings have been made.

Other general assumptions made when creating the Gothenburg model are accounted for below:

- The timber panel is assumed to be eroded due to decay rendering the front pile row unprotected.
- The timber frame in the cofferdam is assumed to not contribute to the lateral bearing capacity of the piles.
- Lateral soil springs in the cofferdam layer is assumed to have properties according to the filling material properties.
- The filling material inside the cofferdam is assumed to be of loose frictional material.
- Undrained conditions are assumed for the clay.
- Land- and canal-side soils where considered the same with it's respective layering.
- Long term behaviour of the soil is implemented in the models.

Important to note about the assumption that the timber panel and frame in the cofferdam are eroded, is that it implies a risk of deterioration of the timber piles closest to the canal. Additionally, some of the filling material between the piles may have eroded into the canal. Not considering the front-row pile and no filling, visualized in Figure 5.5, represent a worst-case scenario that provides low lateral-soil resistance.

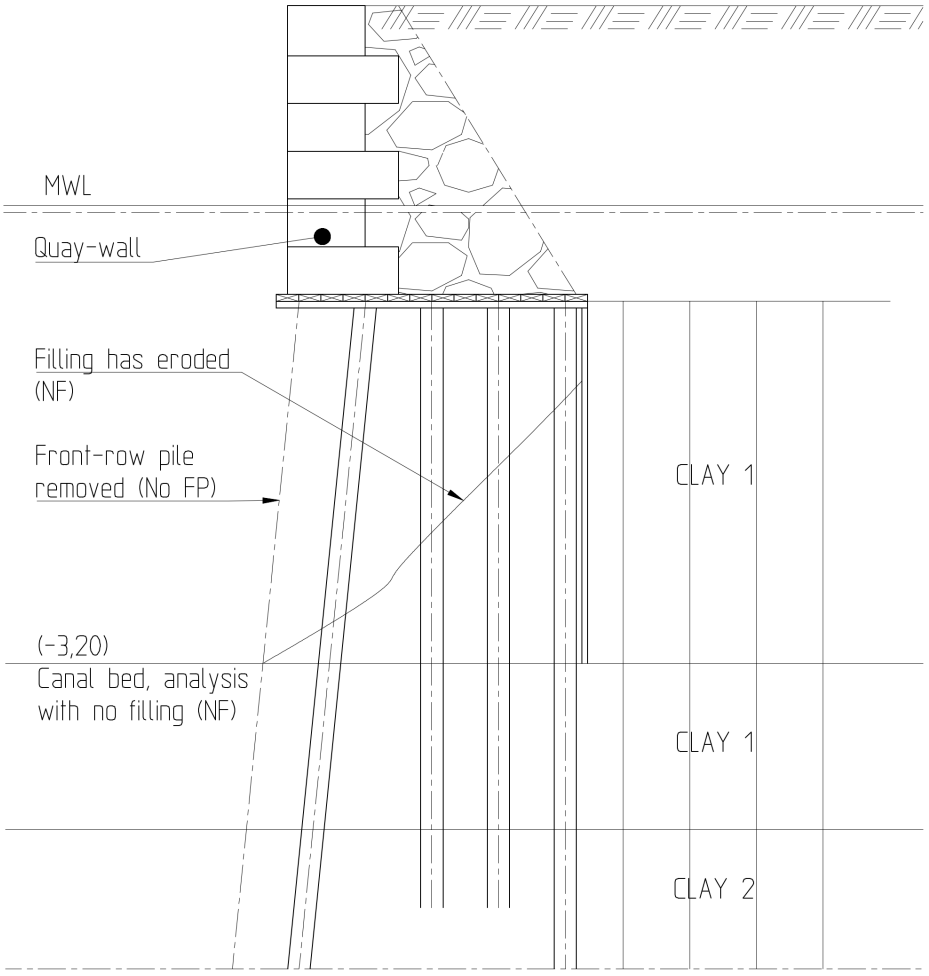


Figure 5.5: Structural layout when the front-row pile is removed and no filling is assumed.

5.1.1 Geotechnical properties

The assumed soil properties considered for the Gothenburg model are shown in Table 5.1. The properties are chosen as representative values based on discussions with the geotechnical department at Sweco in Gothenburg and their experience from projects around the city.

Table 5.1: Soil properties for the Gothenburg case.

Material	γ'_k [kN/m^3]	c_{uk} [kN/m^2]	**Depth start** [m]	**Depth end** [m]	ϕ [deg]
Filling Material	10	0	0	3.2	33
Clay 1	6	23	3.2	4.3	30
Clay 2	6	23 + 0.7 $\cdot z$	4.3	Pile Length	30

The indicated depths in Table 5.1 are relative to the timber deck, which is considered as the origin. For the soil-wedge model, the shear strength was based on soil tests on land which are the parameters shown in the table.

When creating the FE model, the ultimate lateral capacity of the clay and Young's modulus of the soil are determined using the properties found in the canal. This corresponds to a shear strength of $13\,\mathrm{kPa}$ in the first clay layer and a strength increase of $1.7\,\mathrm{kPa/m}$ in the second clay layer. This assumption is on the safe side when determining the strength of the soil and should produce higher stresses from the FE-analysis.

No available data of the cone resistance has been found for the soil whereas other means are necessary to determine the lateral elastic stiffness, E_m, utilised in determining the modulus of subgrade reaction according to Menard's stiffness (Hemel, 2023). Based on the internal friction angle of the filling material, empirical data from Swedish handbooks was used to assume a cone resistance, q_c, of $2500\,\mathrm{kPa}$ (Olsson & Holm, 1993). E_m is determined by multiplying q_c with the empirical factor α which is 0.6 for gravel according to Hemel (2023). For clay, the undrained lateral stiffness can be assumed as $50 \times c_u$, considering long term effects from creep (Svahn & Alén, 2006).

5.2 Wedge model

To create the wedge model Grasshopper was utilised, in combination with a Python code, to obtain the respective angles (φ and β) based on the assumed geotechnical properties from Table 5.1. To capture the increase of cohesive strength due to depth, in the second clay layer, the depth of the centroid of the shear-plane for the respective wedge was used when determining the shear strength. This is shown in Figure 5.6 where the red dot represents the centroid of the wedge.

The analysis was conducted with two models created in Grasshopper; one with straight piles (Gothenburg-straight), and one with inclined and straight piles.

5.2.1 Straight model

As in the Grimburgwal case, the first modelled elements are the piles, as shown in Figure 5.7, which helps as a verification of the wedges locations.

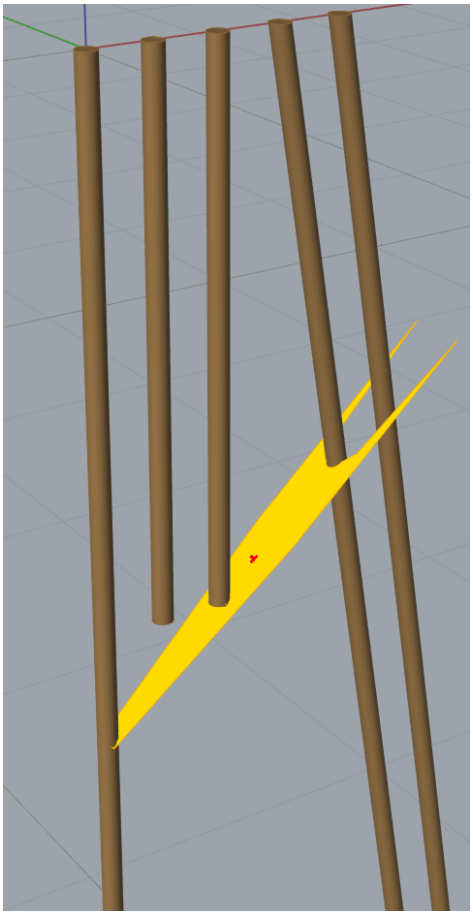


Figure 5.6: Representation of a random reduced wedge and it's centroid.

The process that follows is similar to the Grimburgwal case but with the geometry of the Gothenburg quay walls implemented in the Grasshopper model. The comparison of the reduced and unreduced wedges for each pile is shown in Appendix H.

The comparison between the unreduced and reduced soil wedges, for the whole pile group, is seen by comparing Figures 5.8 and 5.9, showing the unreduced and reduced wedges respectively.

Resulting reduction factors for soil weight and cohesion, Ψ_w and Ψ_c, for the different piles are presented in Table 5.2.

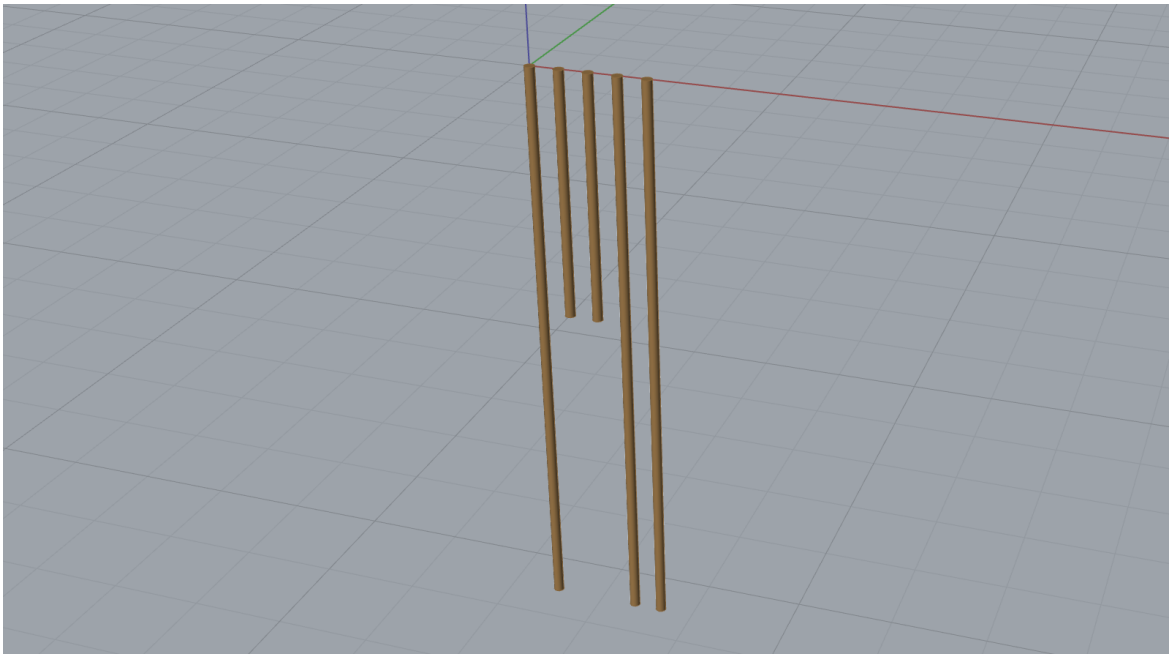


Figure 5.7: Pile group, straight model.

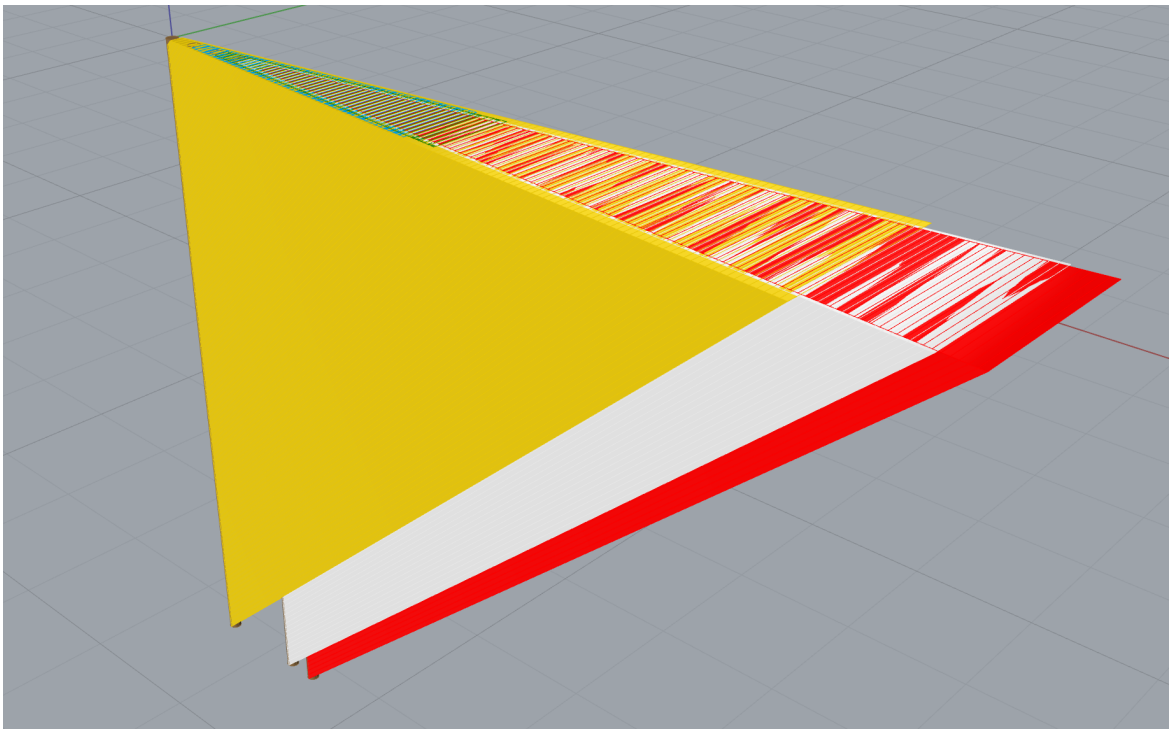


Figure 5.8: Unreduced volumes for the soil wedges, for all piles in the pile group.

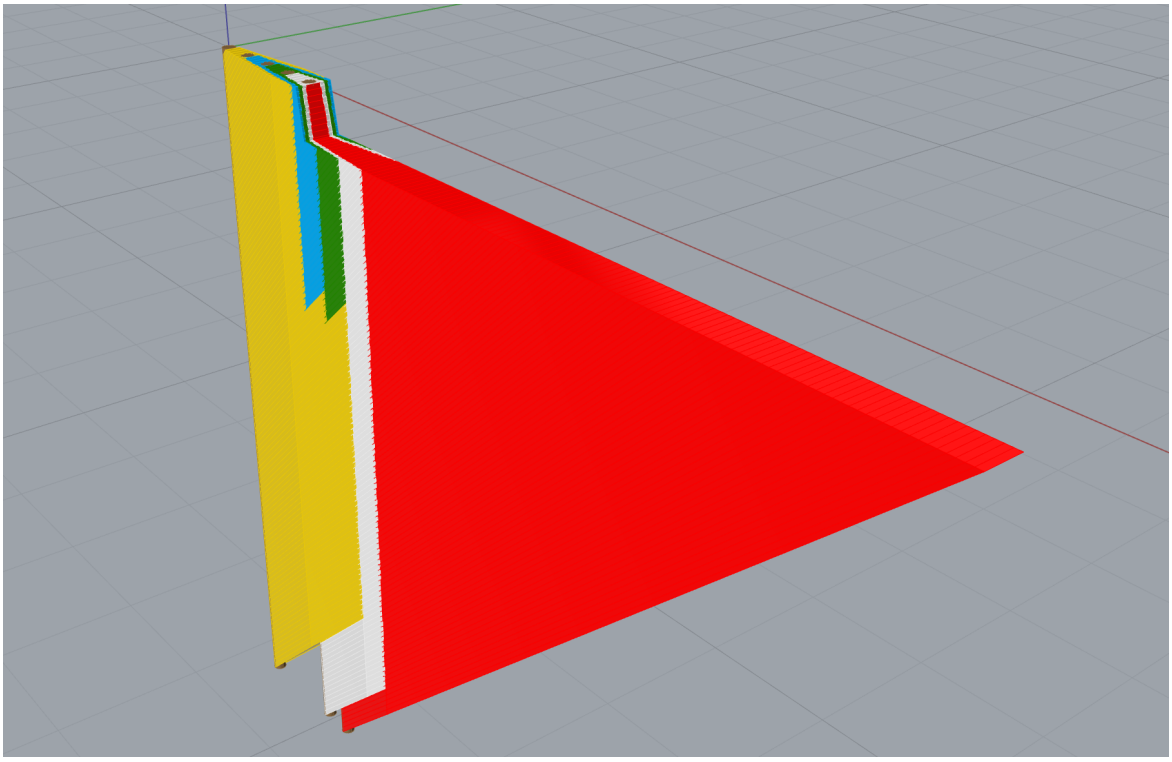


Figure 5.9: Reduced volumes for the soil wedges, for all piles in the pile group.

Table 5.2: Obtained reduction factors for each pile at different depths.

Ψ_w and Ψ_c results — Piles 1 to 3						
Depth [m]	**Pile 1**		**Pile 2**		**Pile 3**	
	Ψ_w	Ψ_c	Ψ_w	Ψ_c	Ψ_w	Ψ_c
0.1	1	0	1	0	1	0
1	0.5562	0	0.5562	0	0.5562	0
2	0.3242	0	0.3062	0	0.2469	0
3	0.1747	0	0.1747	0	0.1747	0
8	0.0710	0.3034	–	–	–	–

Ψ_w and Ψ_c results — Piles 4 and 5				
Depth [m]	**Pile 4**		**Pile 5**	
	Ψ_w	Ψ_c	Ψ_w	Ψ_c
0.1	1	0	0.6929	0
1	0.4332	0	0.1792	0
2	0.2397	0	0.4876	0
3	0.1747	0	0.5934	0
8	0.0228	0.1014	0.3870	0.7308

As expected, the top wedge have no reduction regarding the soil weight, Ψ_w, since there are yet no geometries prohibiting the wedge propagation to the surface. Evaluating the reduction further down we notice a larger reduction. This is reasonable since the wedges from

the back-row piles are cut against the wedges from the piles in front and restricted due to longitudinal pile spacing. On the other hand, the wedges from the front-row pile (Pile 5) are immediately reduced due to the timber panel in front. For Pile 2 and 3, no reduction factors exist below $5.2\,\mathrm{m}$ since these piles are shorter than the rest.

Evaluating the reduction of the shear areas, Ψ_c, full reduction is applied above $3.2\,\mathrm{m}$ since no cohesion is assumed in the filling material around the piles.

5.2.2 Inclined Model

In this model, to better represent the real structural layout, an inclination of 6:1 for the two front-row piles is included. This new configuration is shown in Figure 5.10. The Python codes utilised for this, are located in Appendix I.

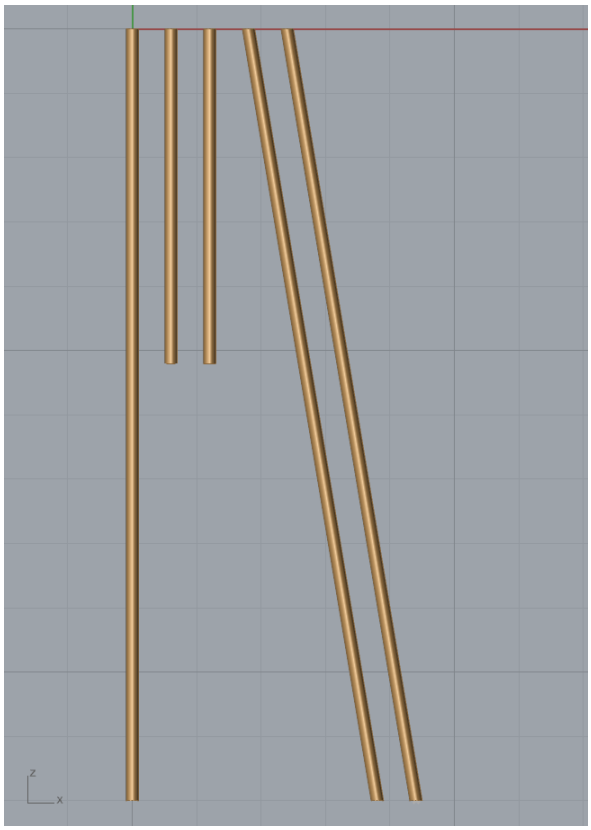


Figure 5.10: Pile group, inclined model.

The reduced wedge model for this case is shown in Figure 5.11 where the colours of the wedges, representing each pile, are the same as for the straight model.

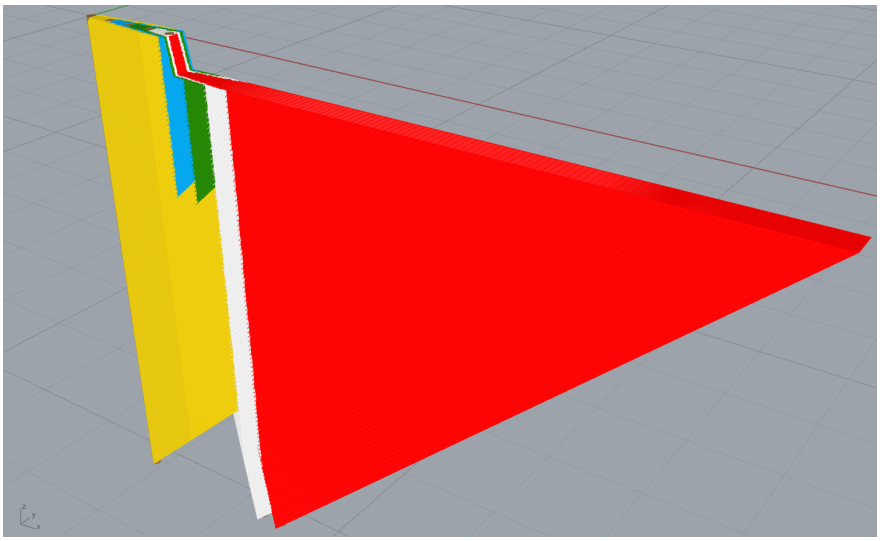


Figure 5.11: Reduced wedge model for inclined piles

The comparison between the inclined and straight pile-model is shown in Figure 5.12.

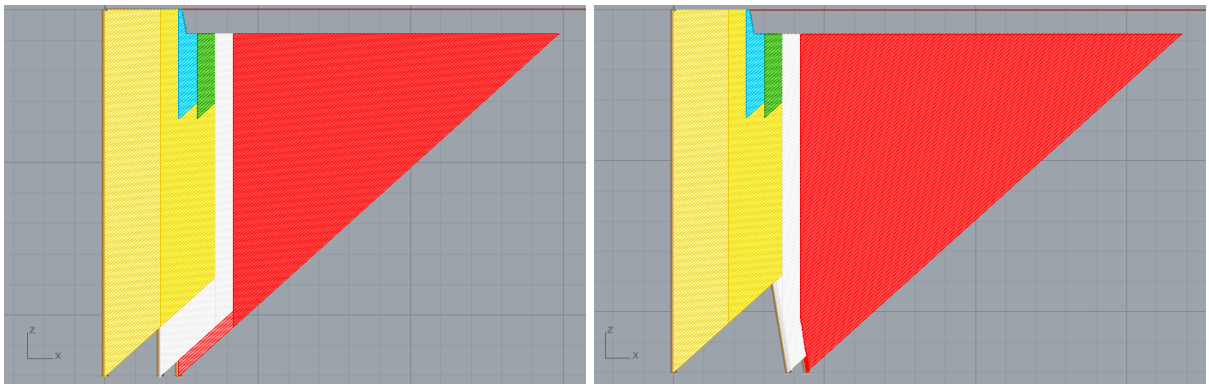


Figure 5.12: Comparison of the reduced volumes of the soil wedges. Left: Straight model, Right: Inclined model

The major difference between the models is that the volume belonging to the soil wedge of Pile 4 (white) is smaller in the inclined model, resulting in a larger volume corresponding to Pile 1 (yellow).

Due to the differences of the models, utilizing both in the analysis would lead to different results. Since the inclined model better represents the real structural layout this is used when running the sensitivity study.

The obtained reduction factors, at some specified depths along the pile, are shown in Table 5.3.

Table 5.3: Obtained reduction factors for each pile at different depths, inclined model.

Ψ_w **and** Ψ_c **results — Piles 1 to 3**						
Depth [m]	**Pile 1**		**Pile 2**		**Pile 3**	
	Ψ_w	Ψ_c	Ψ_w	Ψ_c	Ψ_w	Ψ_c
0.1	1	0	1	0	1	0
1	0.5644	0	0.5644	0	0.6135	0
2	0.3273	0	0.3093	0	0.3049	0
3	0.2970	0	0.1764	0	0.2275	0
8	0.0948	0.3366	–	–	–	–

Ψ_w **and** Ψ_c **results — Piles 4 and 5**				
Depth [m]	**Pile 4**		**Pile 5**	
	Ψ_w	Ψ_c	Ψ_w	Ψ_c
0.1	1	0	0.6156	0
1	0.3651	0	0.1714	0
2	0.2270	0	0.5004	0
3	0.1386	0	0.5714	0
8	0.0172	0.0581	0.3330	0.5654

5.3 FE-model

The parametric study is conducted utilising the FE-software Abaqus. The models are created in a similar manner as Case study 1 with some differences:

- A surcharge load of $5\,\mathrm{kPa}$ is included to consider loading from pedestrian traffic (Trafikverket, 2023). The surcharge is considered to be uniformly distributed on the ground surface.
- Since undrained soil condition is considered for clay the method presented by Christensen and Hansen (1961) for drained condition is not applicable. Instead the extended Broms' theory is utilised, according to Cecconi et al. (2019), to determine the ultimate lateral resistance of the soil.
- Inclined piles are present in the pile foundation.
- The effect of deterioration of the front-row pile will be investigated.

The geometry in the FE-model can be seen in Figure 5.13, showing the inclined piles and the pile division for insertion of lateral soil springs. The timber deck is acting as a cantilever, protruding out over the second pile row, when the front-pile is removed. Further details can be found in Appendix J.

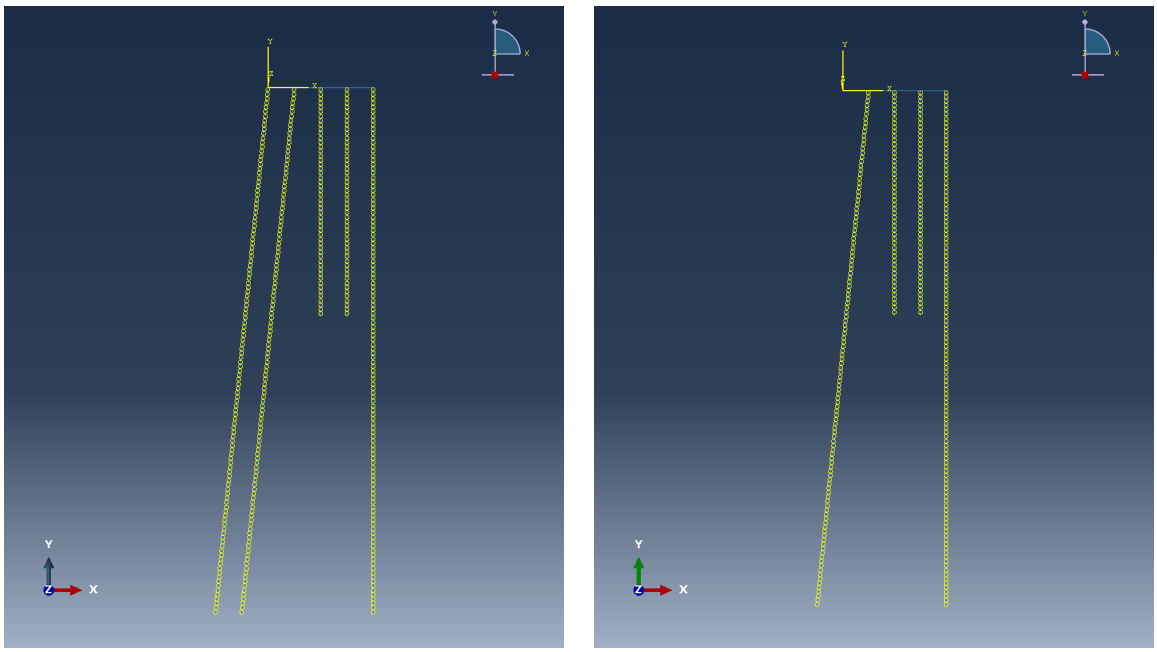


Figure 5.13: The FE-model, assembly view, Left: With front pile, Right: Without front pile.

5.3.1 Conducted iterations

The sensitivity study is conducted on the following parameters which are divided into "cases and conditions":

1. **Conditions**
   (a) Including or not including the reduction factors Ψ_w and Ψ_c in the model.
   (b) Including or not including the filling material around the piles in the cofferdam to consider erosion.
2. **Cases**
   (a) Including or not including the front-row pile in the analysis to consider deterioration of the timber piles.
   (b) Varying the inclination of the piles from 10:1 to 6:1.
   (c) Varying the pile diameter from $0.18\,\mathrm{m}$ to $0.22\,\mathrm{m}$.
   (d) Varying the horizontal earth pressure coefficient between active- and at-rest conditions.

In total 96 analysis are conducted where the maximum stress in each pile is obtained. For all iterations non-linear effects are considered by including non-linear geometry in the FE-model.

5.3.2 Implemented loads

The loads from the superstructure are assumed to be distributed down to the timber deck with the same assumptions made Case study 1, see Figure 4.5. Since the back of the wall is inclined, the methods presented for the calculation of forces acting on gravitational retaining walls were used, compare Figure 3.13 (Avilés L., 2015), (Algers et al., 1961b). The effect of wall friction is beneficial when determining the horizontal force component on an inclined wall, therefore δ was set to 0 in the equation.

The vertical forces from the self weight of the wall, and the overturning moment from horizontal earth pressure and surcharge loading, are combined to find the location of the vertical resultant force according to Equation 3.1. The stress distribution against the timber deck is then determined according to Equation 3.2 or 3.3. See Figure 5.14.

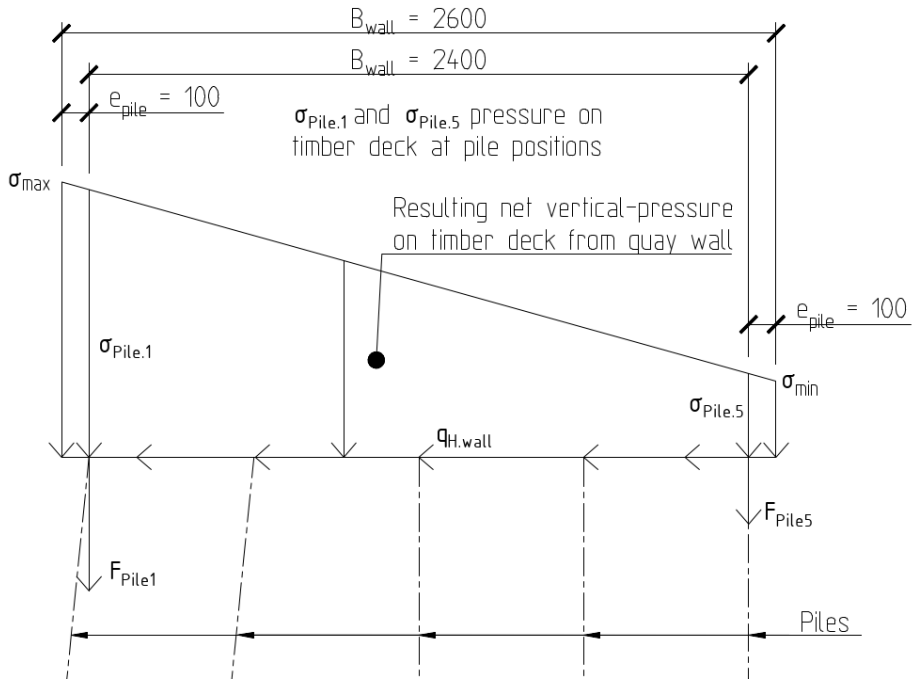


Figure 5.14: Assumed load distribution from wall to the foundation.

Since the wall is wider than the distance between the front-, and end pile, the stress distribution along the timber deck is only described between these piles. The max-, and min stress is obtained at the respective pile positions. The stress field outside the timber deck is implemented in the model by applying a concentrated vertical force on the edge of the timber deck, see F_{pile1} and F_{pile2} in Figure 5.14. Due to the wall-, and timber deck geometry no vertical stress is applied to the timber deck.

Since the embankment piling exists behind the quay wall, resisting the vertical forces behind the pile foundation of the quay wall, no horizontal loads on the piles are assumed to exist. The application of the loads in the FE-model can be seen in Figure 5.15 where the concentrated load applied to the front of the timber deck is marked.

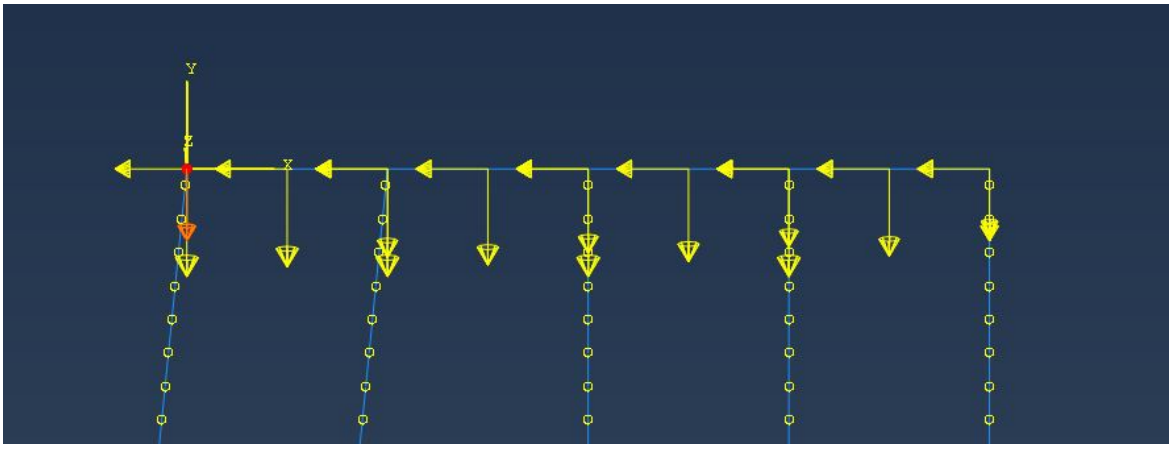


Figure 5.15: Load distribution from the wall to the timber deck in Abaqus.

5.4 Results

The study considers different cases of structural layout, and conditions of wedge reduction and inclusion of the lateral springs in the filling material. These results where sorted with a Python code and tables are presented to summarize the results. The sorting is ordered based on the previously explained cases by:

- Diameter:
  - $180\,\mathrm{mm}$
  - $200\,\mathrm{mm}$
  - $220\,\mathrm{mm}$
- Horizontal earth pressure coefficient as K0 or KA.
- Front pile included or not (FP or No FP).
- Varied pile inclination (I = 10:1 or 6:1).

In total 24 cases of structural layout are created which are then sorted in 4 different combinations of conditions; with or without soil wedge reduction or lateral springs in the filling material. These conditions are sorted as:

- RNF $\rightarrow$ Reduction and No Filling Included.
- NRNF $\rightarrow$ No Reduction and No Filling included.
- RF $\rightarrow$ Reduction and Filling Included.
- NRF $\rightarrow$ No reduction and Filling included.

The combinations are exemplified in Table 5.4 for one evaluated pile diameter and condition.

Table 5.4: Example of iterations conducted for condition NRNF and pile diameter $180\,\mathrm{mm}$.

Case ID	K-coefficient	Front Pile	Pile Inclination	Condition
1	K0	FP	10:1	NRNF
2	K0	FP	6:1	NRNF
3	K0	No FP	10:1	NRNF
4	K0	No FP	6:1	NRNF
5	KA	FP	10:1	NRNF
6	KA	FP	6:1	NRNF
7	KA	No FP	10:1	NRNF
8	KA	No FP	6:1	NRNF

These combinations sum up to the 96 different setups that are obtained from the parametric study. The condition with no wedge reduction and including the filling (NRF) can be viewed as the most beneficial, since the response of the lateral soil springs is the most stiff. A condition with the wedge reduction and no filling included (RNF) should produce the highest stress, since the lateral spring stiffness is reduced.

The obtained maximum pile stress is compared against the strength of the timber piles. The range presented by Hemel (2023) is utilised to check the risk of failure where the mean MOR of $10.6\,\mathrm{MPa}$ is used to determine the $90\,\%$ confidence interval, between $5\,\mathrm{MPa}$ and $18.2\,\mathrm{MPa}$. The obtained stress from the analysis is compared to these stress ranges.

5.4.1 Results - General

For all studied combinations the maximum stress in each pile have been obtained. This is summarized in Table 5.4.1 where the stress is sorted based on the evaluation of the MOR.

Table 5.5: Sorting of highest stress in each pile.

ID	$> 18.2\,\mathrm{MPa}$	$18.2\,\mathrm{MPa} \geq$ **Max Stress** $\geq 5\,\mathrm{MPa}$	$< 5\,\mathrm{MPa}$
Amount of Piles	30	179	223
Compliance $\%$	6.94	41.44	51.62

Evaluating Table 5.4.1 it's concluded that about $7\,\%$ of all the piles are in the critical stress range, above $18.2\,\mathrm{MPa}$, which indicates failure. More than half of the piles are in the lower range, below $5\,\mathrm{MPa}$, indicating low risk of yielding.

If one pile in the pile group for each combination is in the upper range there is a significant risk of failure for the whole pile group. Therefore, the maximum stress for each pile group is evaluated in Table 5.4.1 for each case, considering only the worst condition and sorting it in the corresponding stress range.

Table 5.6: Sorting of highest stress for each case considering all conditions.

ID	$> 18.2\,\mathrm{MPa}$	$18.2\,\mathrm{MPa} \geq$ **Max Stress** $\geq 5\,\mathrm{MPa}$	$< 5\,\mathrm{MPa}$
Amount of Cases	8	15	1
Compliance $\%$	33.33	62.50	4.17

By evaluating the maximum stress for each case it's concluded that one third of all cases are in the upper stress range. These cases risk collapse of the whole pile group. The maximum stress occurs with a small pile diameter, no front-row pile, straighter piles and without lateral springs in the filling. The stress distribution is shown in Figure 5.16, comparing the conditions with the lowest (NRF) and highest stress (RNF).

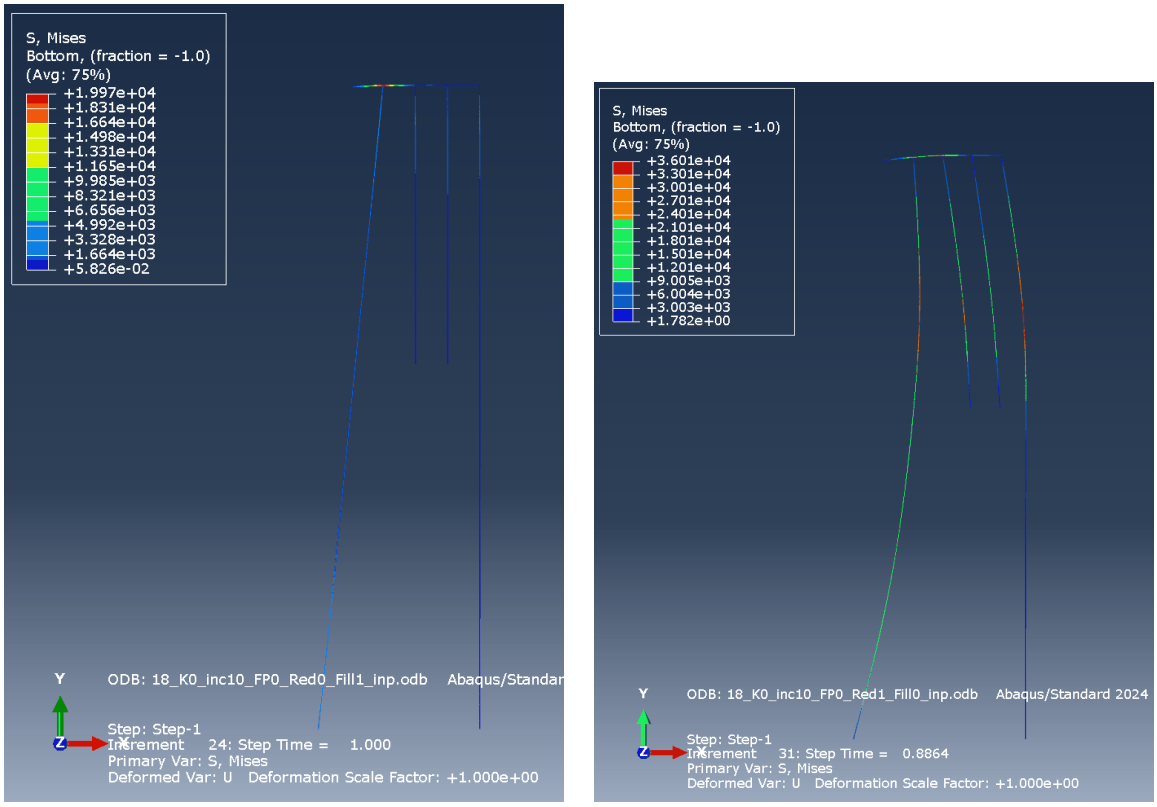


Figure 5.16: vonMises stress, D = 180 mm, No FP, K0, I = 10:1. Left: NRF, Right: RNF.

For the same case, the assumption of including wedge reduction (R) and no filling (NF) produce much larger stresses than when they are not included (NRF). For the most beneficial condition, the maximum stress occurs in the timber deck and moves to the piles when the worst condition is applied. It is notable that the whole load increment is not applied for the worst condition but stops at an increment of 0.8864.

A comparison, evaluating the stress distribution for a diameter of 200 mm for the same conditions, can be seen in Figure 5.17 where K0 is considered and the front-row pile is removed.

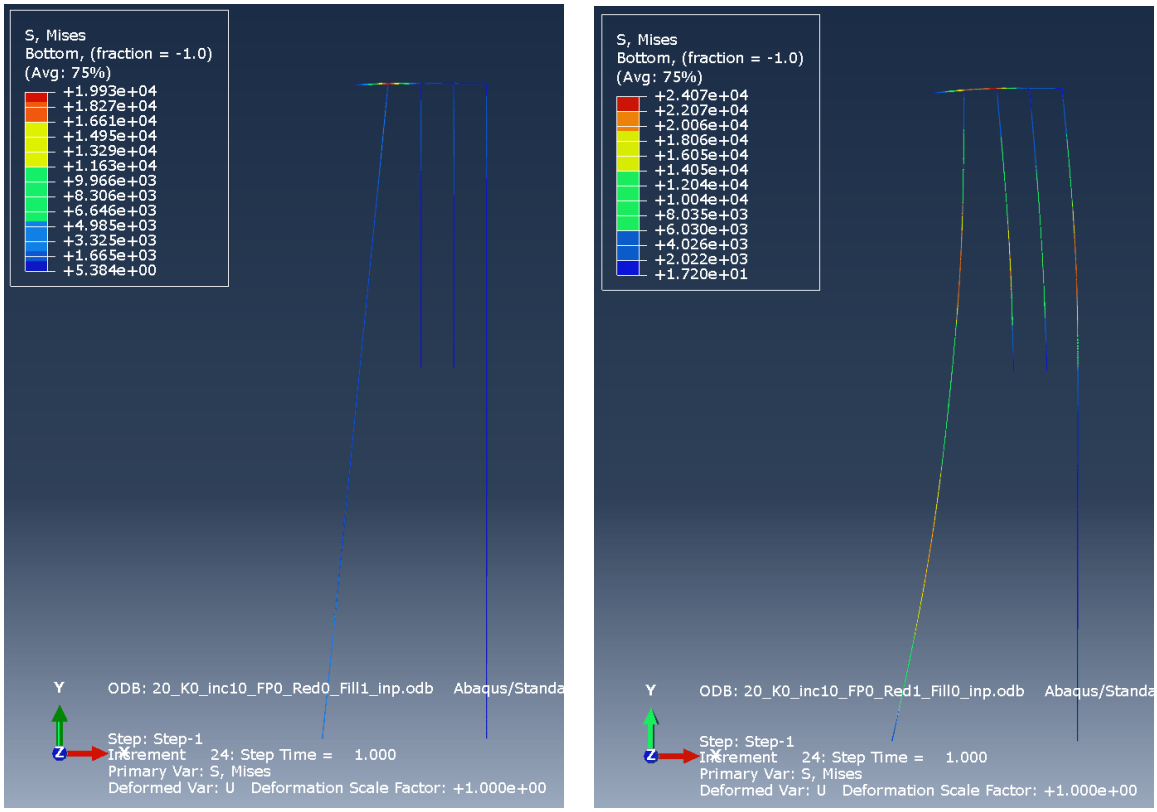


Figure 5.17: vonMises stress, D = 200 mm, No FP, K0, I = 10:1. Left: NRF, Right: RNF.

The maximum stress can, for both conditions, be found in the timber deck but the stress increase in the pile is clearly more visible in the worst condition RNF. Compared to Figure 5.16 when the condition RNF is studied, the stress have decreased by at least 10 MPa. For the condition NRF, the stress in the timber deck is much larger than in the piles.

To study in detail which parameter is the most critical, leading to higher stresses in the piles, the result is divided further.

5.4.2 Results - Detailed

The obtained maximum stress in the pile group for each case is elaborated on in the following chapter. The maximum stress is divided per condition (mentioned earlier) listed below with it's corresponding result table:

- RNF → Reduction and No Filling (Tables 5.7 and 5.8).
- NRNF → No Reduction and No Filling (Tables 5.9 and 5.10).
- RF → Reduction and Filling (Tables 5.11 and 5.12).
- NRF → No reduction and Filling (Tables 5.13 and 5.14).

Tables 5.9 to 5.12 show in which stress range the maximum stress for each case is located, sorted per condition. The first table for each condition shows the stress range only considering the piles while the second includes the maximum stress in the timber deck. The compliance represent the percentage of cases for each condition in proportion to all 96 combinations or in proportion to the studied condition. The complete results for these conditions are located in Appendix L.

Condition RNF

The division in stress range for the condition with reduction factor and no filling (RNF), and not including the timber deck in the evaluation, is shown in Table 5.7.

Table 5.7: Condition RNF - Stress evaluation, cases.

ID	$> 18.2\,\mathrm{MPa}$	$18.2\,\mathrm{MPa} \geq$ **Max Stress** $\geq 5\,\mathrm{MPa}$	$< 5\,\mathrm{MPa}$
Amount of Cases	8	15	1
Compliance Global %	8.33	15.63	1.04
Compliance Condition %	33.33	62.50	4.17

In this condition (RNF) the highest stress in the pile group should be obtained and it is found that 8 out of 24 combinations are in the critical stress range. From these combinations the following is observed:

- $75\,\%$ (6 cases) of the combinations are with **no front pile**
  - 4 of these ($66.67\,\%$) are with **K0**.
    - All of these are in the critical range **both with 10:1 and 6:1** inclination.
- $25\,\%$ (2 cases) are **with front pile**.
  - Both are with **K0** and **D**=$180\,\mathrm{mm}$.
    - **Pile inclination does not matter**.

The stress distribution in the piles for the worst combination is shown in Figure 5.18, represented with the red column. This occurs at D = $180\,\mathrm{mm}$, with no front-row pile, K0 and an inclination of 10:1.

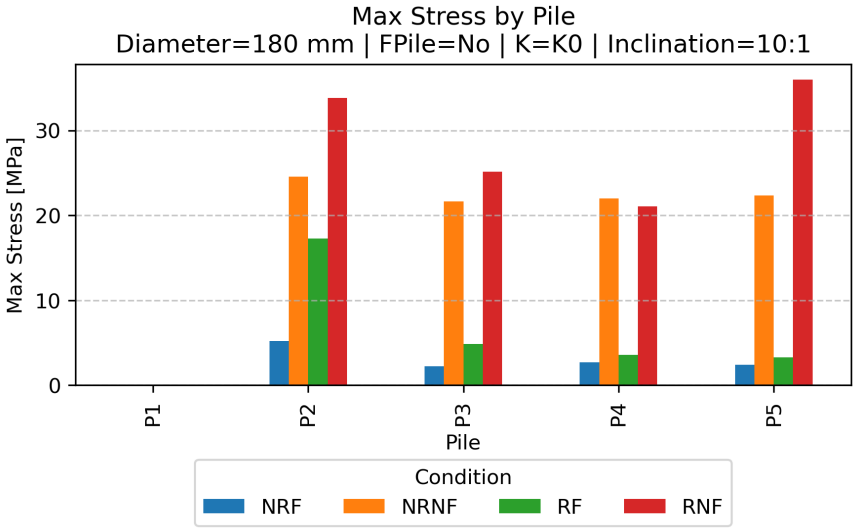


Figure 5.18: Max stress by pile when D = 180 mm, No FP , K0 and I = 10:1.

As a comparison, the stress distribution when changing certain parameters is shown; including the front-row pile in Figure 5.19, and changing to D = 200 mm in Figure 5.20.

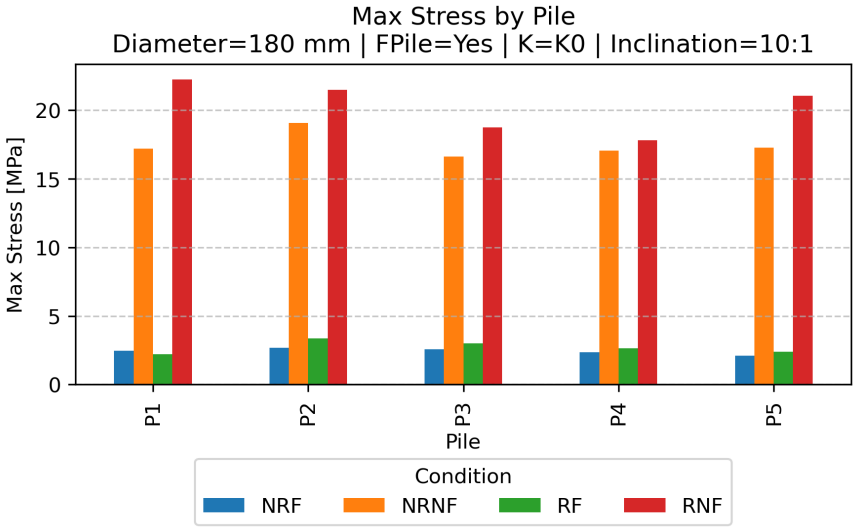


Figure 5.19: Max stress by pile when D = 180 mm, FP, K0 and I = 10:1.

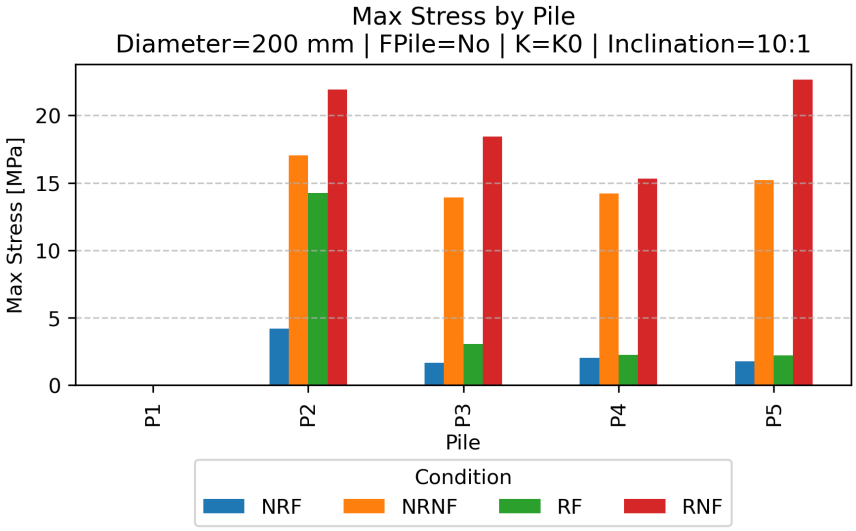


Figure 5.20: Max stress by pile when D = 200 mm, No FP, K0 and I = 10:1.

The stress decrease, when including the front-row pile or increasing the pile diameter, is clear when comparing Figure 5.18 with Figures 5.19 and 5.20.

The division in stress range when considering the stress in the timber deck in the evaluation is shown in Table 5.8.

Table 5.8: Condition RNF - Stress evaluation considering the timber deck.

ID	$> 18.2\,\mathrm{MPa}$	$18.2\,\mathrm{MPa} \geq$ **Max Stress** $\geq 5\,\mathrm{MPa}$	$< 5\,\mathrm{MPa}$
Amount of Cases	14	9	1
Compliance Global %	14.58	9.38	1.04
Compliance Condition %	58	38	4

Considering the stress in the timber deck, the amount of combinations that are sorted into the critical stress range increase from 8 to 14 cases. Out of these, 12 are when the front-row pile is not included.

Condition NRNF

The division in stress range for the condition with no reduction factor and no filling (NRNF) is shown in Table 5.9, not considering the stress in the timber deck.

Table 5.9: Condition NRNF - Stress evaluation, cases.

ID	$> 18.2\,\mathrm{MPa}$	$18.2\,\mathrm{MPa} \geq$ **Max Stress** $\geq 5\,\mathrm{MPa}$	$< 5\,\mathrm{MPa}$
Amount of Cases	2	20	2
Compliance Global %	2.08	20.83	2.08
Compliance Condition %	8.33	83.33	8.33

For this condition, two cases where D = $180\,\mathrm{mm}$, K0 is applied and the pile inclination is 10:1 lead to a maximum stress in the pile group in the critical range. The pile stresses for these cases are shown in Figure 5.21, represented by the orange columns.

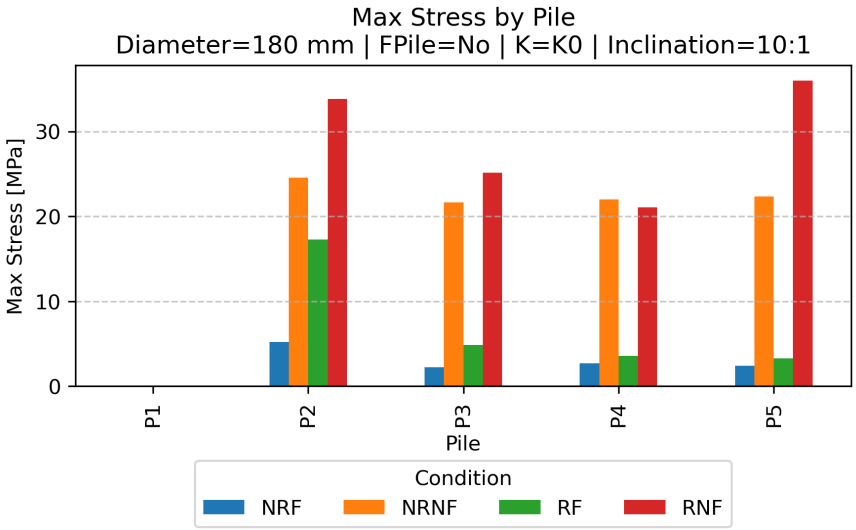


Figure 5.21: Max stress by pile when D = 180 mm, No FP , K0 and I = 10:1.

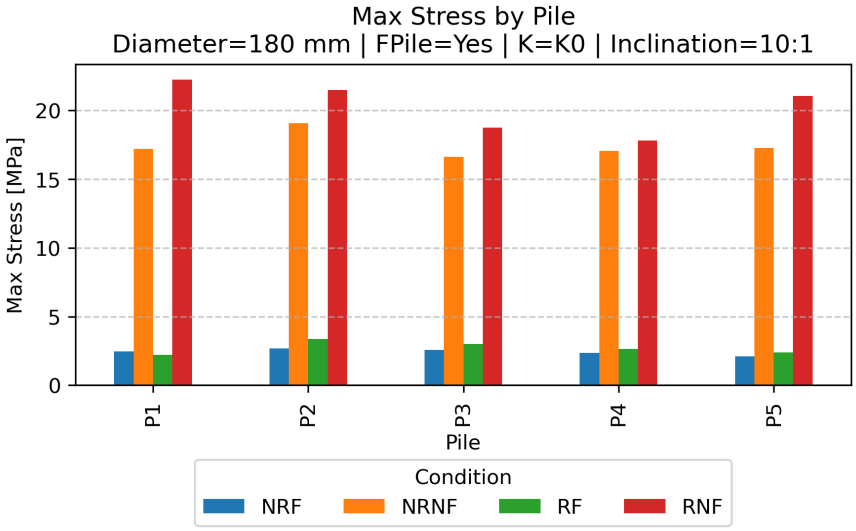


Figure 5.22: Max stress by pile when D = 180 mm, FP , K0 and I = 10:1.

It doesn't matter if the front pile is included or not, the maximum stress for this structural layout is still in the critical range.

When including the timber deck stress in the evaluation, five other cases are in the critical range, see Table 5.10.

Table 5.10: Condition NRNF - Stress evaluation considering the timber deck.

ID	$> 18.2\,\mathrm{MPa}$	$18.2\,\mathrm{MPa} \geq$ **Max Stress** $\geq 5\,\mathrm{MPa}$	$< 5\,\mathrm{MPa}$
Amount of Cases	7	16	1
Compliance Global %	7.29	16.67	1.04
Compliance Condition %	29	67	4

For the timber deck stress to be in the critical range, for this condition, it is required that the front-row pile is removed and that at-rest earth pressure is applied. A larger pile diameter does not affect the result.

Condition RF

The condition including the reduction factor and the filling (RF) is shown in Table 5.11, not considering the timber deck.

Table 5.11: Condition RF - Stress evaluation, cases.

ID	$>18.2\,\mathrm{MPa}$	$18.2\,\mathrm{MPa} \geq$ **Max Stress** $\geq 5\,\mathrm{MPa}$	$<5\,\mathrm{MPa}$
Amount of Cases	0	12	12
Compliance Global %	0	12.5	12.5
Compliance Condition %	0	50	50

Including only the reduction factor resulted in no case with a maximum stress in the critical range. The cases are equally divided in the intermediate and lower stress range. The highest stress, shown in Figure 5.23 and represented by the green columns, is observed in the same case as for condition RNF and NRNF.

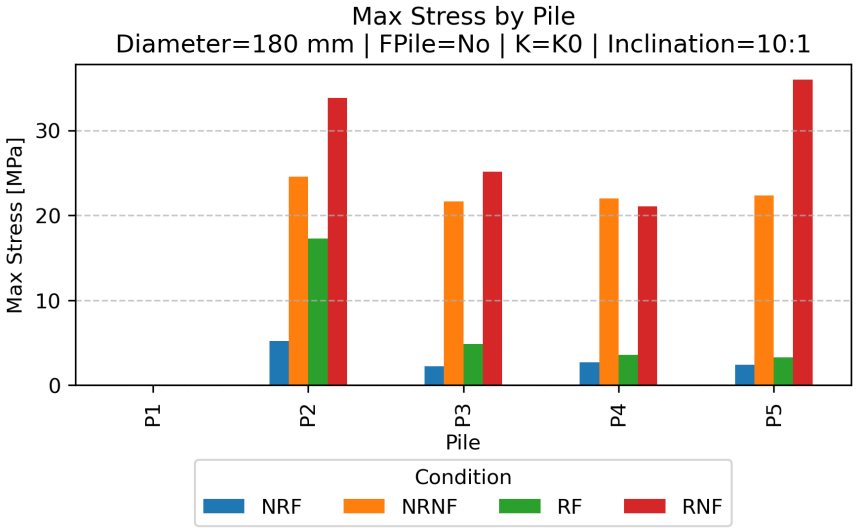


Figure 5.23: Max stress by pile when D = 180 mm, No FP, K0 and I = 10:1.

The critical parameter is the inclusion of the front-row pile where all cases with a removed front-row pile are sorted in the intermediate stress range. The other cases are in the lower stress range.

Table 5.12 show the stress division when the stress in the timber deck is included, sorting $50\,\%$ of the cases in the critical stress range.

Table 5.12: Condition RF - Stress evaluation considering the timber deck.

ID	$> 18.2\,\mathrm{MPa}$	$18.2\,\mathrm{MPa} \geq$ **Max Stress** $\geq 5\,\mathrm{MPa}$	$< 5\,\mathrm{MPa}$
Amount of Cases	12	3	9
Compliance Global %	12.5	3.125	9.375
Compliance Condition %	50	13	38

Similar to when the pile group stress is sorted the critical parameter is when the front-row pile is removed.

Condition NRF

The condition with no reduction factor and filling included (NRF) is shown in Table 5.13, not including the stress in the timber deck.

Table 5.13: Condition NRF - Stress evaluation, cases.

ID	$> 18.2\,\mathrm{MPa}$	$18.2\,\mathrm{MPa} \geq$ **Max Stress** $\geq 5\,\mathrm{MPa}$	$< 5\,\mathrm{MPa}$
Amount of Cases	0	5	19
Compliance Global %	0	5.21	19.79
Compliance Condition %	0	20.833	79.17

In the condition NRF, no cases exhibit maximum pile group stress in the critical range, which was initially assumed. The highest stress is shown in Figure 5.24, displayed with blue bars for the most critical case; D = $180\,\mathrm{mm}$, K0 is applied, 6:1 pile inclination, and not including the front-row pile.

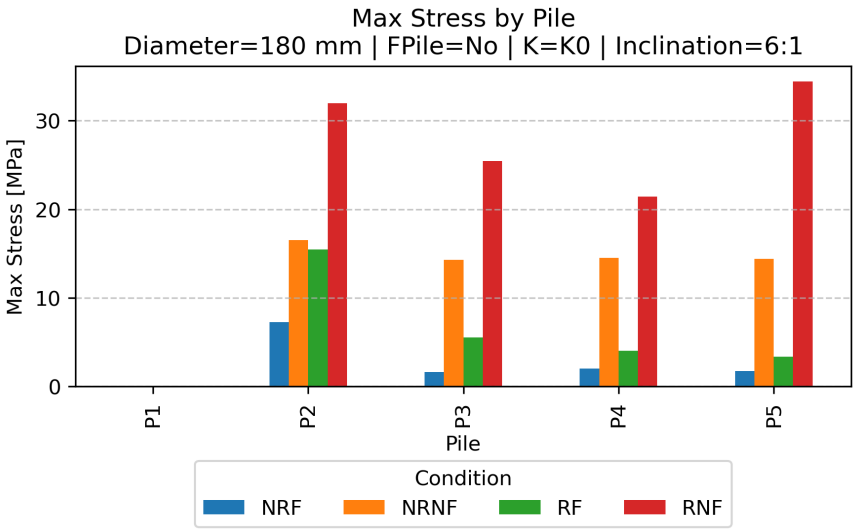


Figure 5.24: Max stress by pile when D = 180 mm, No FP , K0 and I = 6:1.

The critical parameter is the inclusion or not of the front-row pile. As can be seen in Figure 5.25, increasing the pile diameter to $200\,\mathrm{mm}$ and applying active earth pressure only leads to a small stress decrease, from around $7\,\mathrm{MPa}$ down to $5\,\mathrm{MPa}$.

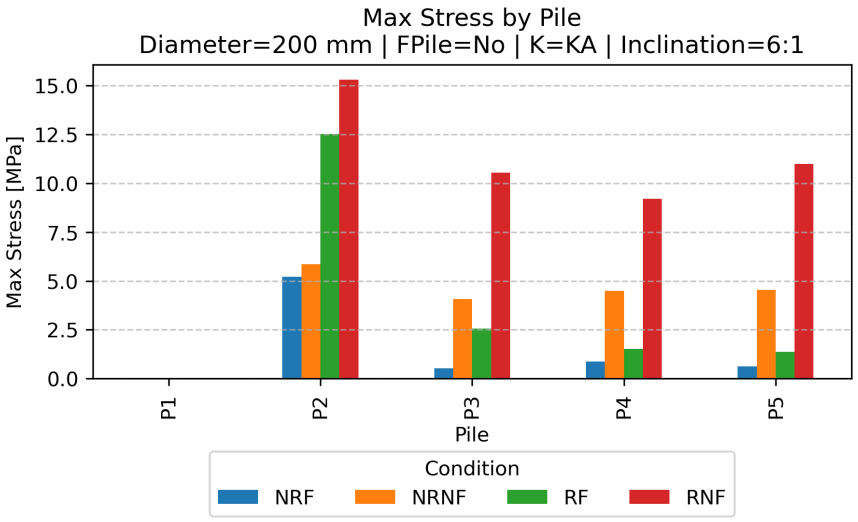


Figure 5.25: Max stress by pile when D = 200 mm, No FP , KA and I = 6:1.

A stress for the whole pile group in the critical range only occurs due to large stresses in the timber deck, see Table 5.14. For this to occur, the front-row pile needs to be removed.

Table 5.14: Condition NRF - Stress evaluation considering the timber deck.

ID	$> 18.2\,\mathrm{MPa}$	$18.2\,\mathrm{MPa} \geq$ **Max Stress** $\geq 5\,\mathrm{MPa}$	$< 5\,\mathrm{MPa}$
Amount of Cases	6	6	12
Compliance Global %	6.25	6.25	12.5
Compliance Condition %	25	25	50

Summary of critical conditions

In the following section the overall rate of simulations, within the different stress ranges, are shown. This corresponds to the global compliance shown in the previous result tables.

Figure 5.26 shows the overall rate of simulations that are in the critical stress range, above $18.2\,\mathrm{MPa}$, for each separate condition.

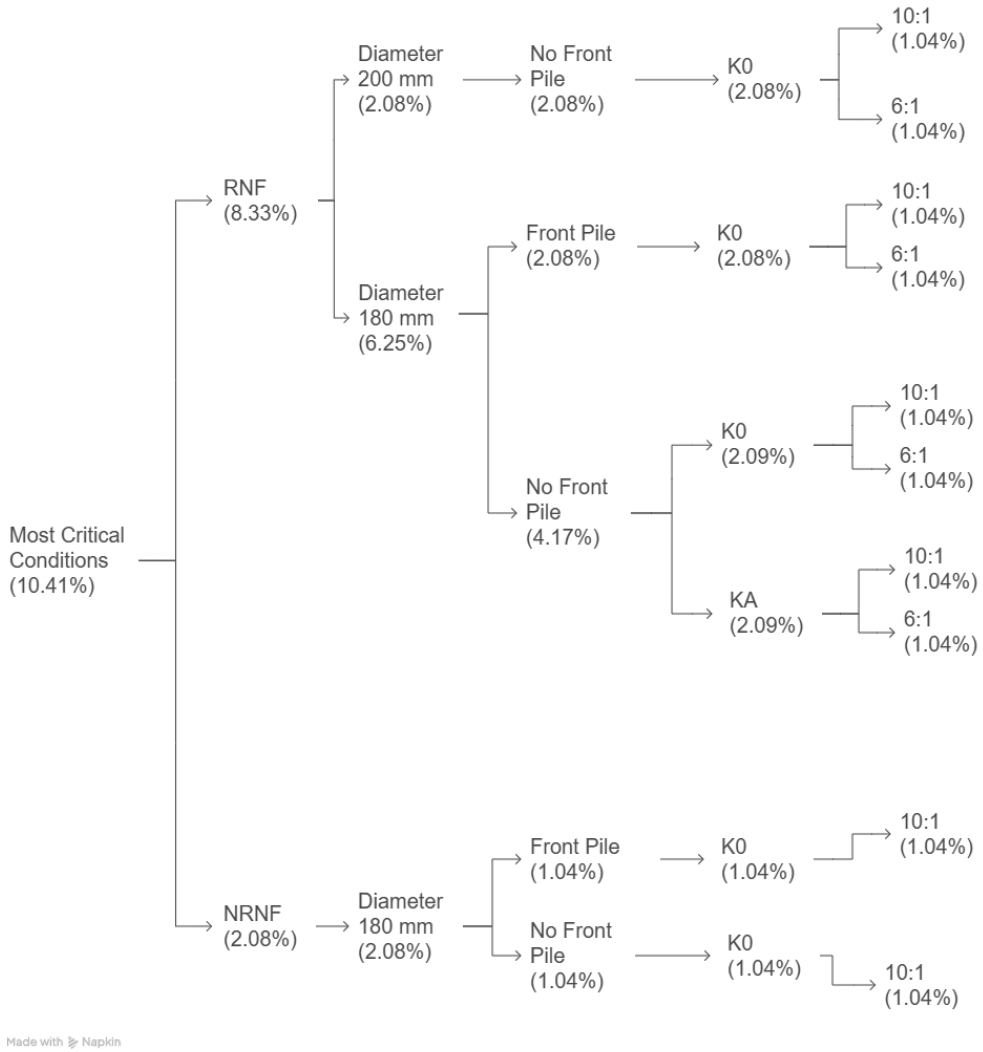


Figure 5.26: Rate from Critical Conditions.

In the same manner, Figures 5.27 and 5.28 show the rates for each condition within the intermediate stress range, between 5 MPa and 18.2 MPa.

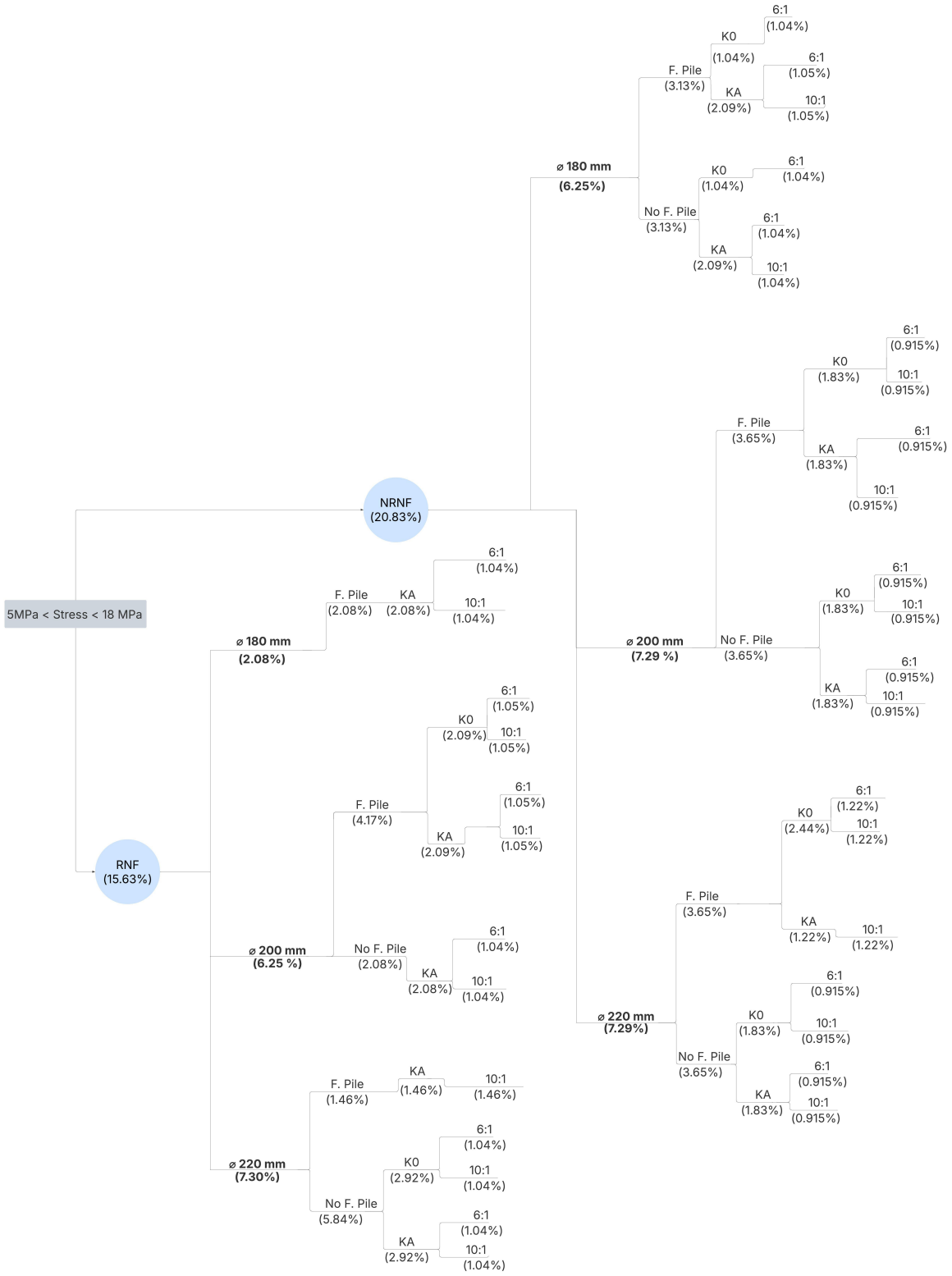


Figure 5.27: Rates for NRNF and RNF conditions between 5 MPa and 18 MPa.

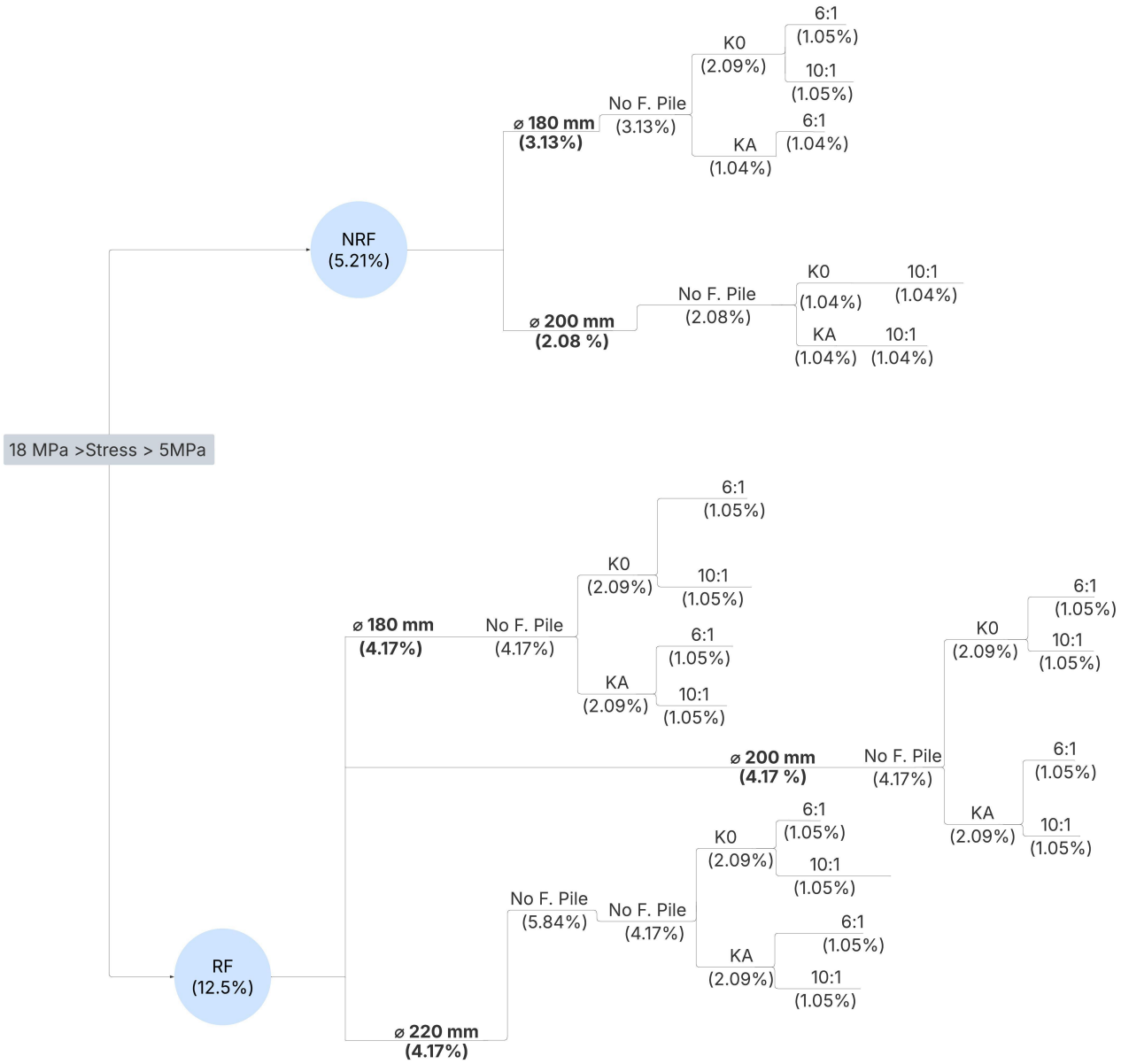


Figure 5.28: Rates for NRF and RF conditions between 5 MPa and 18 MPa.

Finally, Figure 5.29 shows the rates for each condition where the maximum stress in the pile group is lower than 5 MPa.

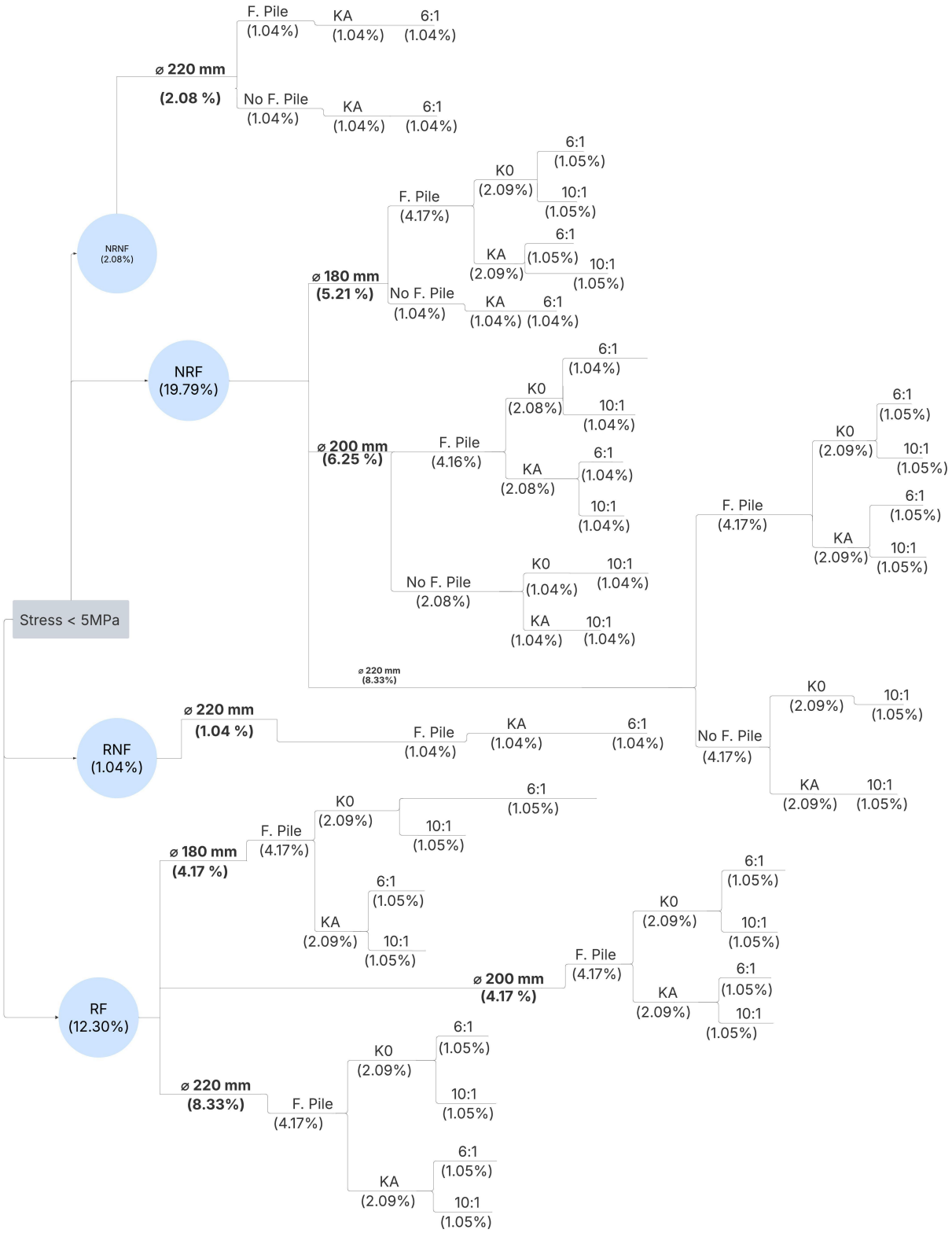


Figure 5.29: Rates for each condition with maximum stress lower than 5 MPa.

6 Discussion

6.1 Observations from Case study 1

The benchmarking of the finite element model in case study 1 did not result in the same stresses as the Grimburgwal reference case, obtained by Korff et al. (2022). This could be due to various reasons; different reduction factors from the soil wedge reduction, load assumptions, or different load distribution in the FE-model between the piles through the timber deck, as shown in Tables 4.4 and 4.6.

The comparison with the hand-calculation indicate that the method according to Olsson and Holm (1993) gives a similar stress distribution as the FE-model when comparing the stresses from the unreduced soil spring model. Utilising the Broms' extension method, presented by Cecconi et al. (2019), on the other hand, produces a much lower stress in the pile compared to the FE-model and reference case of the Grimburgwal. A reason for this larger capacity may be the drained shear strength of the peat layer, used to determine ultimate capacity of the soil, which may overestimate the strength.

6.2 Observations from Case study 2

The results from Case study 1 implies that including the wedge reduction shouldn't produce a notable stress increase. The opposite was observed from the Gothenburg case study. Prior to the analysis of the case study on the Gothenburg wall, it was in this work believed that not including the lateral springs in the top filling layer, or including the front-row pile would produce the maximum stress in the piles. Not including the front-row pile, or the lateral springs in the filling material, can be seen as a lower bound condition. These conditions produce a result that can be considered as a worst-case assumption when determining the stress distribution in the piles. In the same way, assuming at-rest-, or active earth pressure against the wall can also be interpret as a lower-, and upper bound solution where earth pressure at rest is the maximum soil state that can occur against the masonry wall.

In eight cases of structural layout considering all possible conditions, see Table 5.4.1, the maximum stress in the pile group exceed the upper stress limit in the pile material, indicating failure. Out of these, the representative case is likely with K0, considering that at-rest earth pressure has developed as a long-term condition. An assumption that the filling is partly removed and that the front-row pile has a reduced outer diameter, compared to what is found in old drawings, may be a representative case. But, to determine these parameters, an inspection of the structure is required.

The model is not considering any vertical support of the soil below the timber deck. This leads to the timber deck acting as a cantilever beam when the front-row pile is removed. Consequently, the stress in the timber deck from the analyses can be considered as conservative. Incorporating some vertical support from the soil in the model might have a positive influence, decreasing the stress in the timber deck. On the other hand, this finding might have a minor influence since the combinations without the front-row pile show large stresses in the piles anyway, deeming the structural layout unstable. Finally, the capacity of the timber deck is determined based on the timber resistance from Hemel (2023). Using other stress limits when evaluating the structural integrity might lead to other conclusions

than provided in this work.

6.3 Implementation in the industry

The assessment of how to proceed, after an evaluation of the integrity of the quay wall structures have been conducted, is out of the scope of this thesis project. However, if visual inspections are carried out, a validation model is available to assess the structural integrity, and restoration methods can be determined other options than replacing the walls could be available. The restoration or replacement of critical elements may then be more feasible than a complete demolition of the structure and substitution with a modern one. Avoiding with this, the destruction of an historical element and part of the heritage of the city. From the environmental point of view, it is presumable that the utilisation of materials may be lower if a restoration is chosen, which is also beneficial.

7 Conclusions

The FE-model did not produce the same pile stresses when benchmarked against the reference case in the first case study. However, two conclusions can be drawn from Case study 1. The maximum stress in the piles converge, when decreasing the spring spacing below $0.7\,\mathrm{m}$. The increased time for running the analyses when adding more springs was not significant for the model created in this work.

A parametric Finite Element model, using beam elements, and visual programming with Grasshopper to create a wedge reduction model, is shown to be an appropriate method, for conducting a sensitivity study on historical quay walls. After benchmarking the newly developed method against an existing case from Amsterdam in the Netherlands, the model was used for the Gothenburg case. The method can be developed further by including traffic loading in the model and changing from a 2D-, to a 3D-model, to analyse the behaviour of the masonry structure.

From the results in Case study 2, as summarised in Table 5.7, Table 5.9 and from Figure 5.26 to Figure 5.29, it is clear that the most critical condition is when the *Reduction* is applied with *No Filling*. The most critical case, of structural layout, is when the pile diameter is small and when *No Front Pile* is assumed. Assuming *K0* produce significantly higher stresses than *KA*.

It is clear that the assumption regarding the lateral springs in the filling layer was correct. All cases show an increase of stress when the filling is not included in the model, with or without the wedge reduction implemented. Evaluating the worst case (the smallest pile diameter, no front-row pile, inclination of 10:1, and at-rest earth pressure) when including the reduction factor as a condition, we obtain a maximum pile stress of $36\,\mathrm{MPa}$ without soil springs in the filling. The maximum stress decreased to $17.3\,\mathrm{MPa}$ when including the soil springs.

Similarly, the existence, or not, of the front-pile is determined as a critical case for the structure. When deteriorated to the point that it doesn't contribute to the structural integrity, the timber deck will fail regardless of the lateral soil stiffness condition, if at-rest earth pressure is acting against the wall. However, if the condition of the front-pile is adequate, its diameter plays a critical role in preventing excessive stresses in the piles.

The integrity of the front-row pile and the current state of the canal bed, and filling in the top soil layers, should be the main focus when inspecting the structures.

7.1 Recommendation for further studies

Even though further studies are needed, the outcome of this project narrows down the number of critical parameters when assessing the integrity of historical quay walls. The developed model, can be used as a guide for the industry projects. However, it shall not be considered as a decisive parameter in decision making. With this said, the following topics have been identified as suitable future studies:

- An evaluation of the behaviour of the historical quay walls found in Gothenburg when

loaded by short-term loading conditions; traffic-, tram-, or ice loading. Since our work does not consider Swedish national guidelines it could be implemented in this future study, to complement the results from this work.

- Evaluate the integrity of the masonry structure by composing a 3D FE-model, considering frictional and interlocking effects of the masonry blocks, in the front of the wall and in the backfill. The response of the masonry structure due to displacements and rotations of the foundation could be evaluated.
- A validation of the developed model against full-scale tests conducted in Europe with similar soil-conditions, such as the one conducted by Hemel (2023).
- Evaluation and analysis of restoration and construction methodology for repairing historical quay wall structures, considering unseen aspects that this project might had missed, and investigating economical aspects.

Bibliography

Algers, B., Forsby, L., & Wilhelm, T. (1961a). *BYGG - 1. allmäna grunder* (3rd ed., Vol. 1). AB Byggmästarens förlag.

Algers, B., Forsby, L., & Wilhelm, T. (1961b). *BYGG - 2. materiallära, 3.konstruktionsteknik* (3rd ed., Vol. 2). AB Byggmästarens förlag.

Ashour, M., & Norris, G. (2000). Modeling lateral soil-pile response based on soil-pile interaction. *Journal of Geotechnical and Geoenvironmental Engineering*, *126*(5), 420–428. https://doi.org/10.1061/(ASCE)1090-0241(2000)126:5(420)

Avilés L., J. (2015). Manual de diseño de obras civiles. cap. c.2.1 diseño de estructuras de cimentaciónes. sección c: Estructuras. tema 2: Diseño de estructuras especiales.

Aygül, M., Darholm, T., & Thorsell, M. (2017). Numerical simulation of protection barriers for bridge piers against ship collision, 1264–1273. https://doi.org/10.2749/vancouver.2017.1264

Beck, A. T., & Da Silva, C. R. (2011). Timoshenko versus euler beam theory: Pitfalls of a deterministic approach. *Structural Safety*, *33*(1), 19–25. https://doi.org/10.1016/j.strusafe.2010.04.006

Bergdahl, U., Malmborg, B. S., & Ottosson, E. (1993, May). *Plattgrundläggning*. Svensk Byggtjänst AB och Statens geotekniska institut.

Bergström, U., Pile, O., Curtis, P., & Eliasson, T. (2022, February). *Göteborgsområdets berggrund, jordarter och geologiska utveckling* (2021:31). SGU - Sveriges geologiska undersökning. Retrieved May 22, 2025, from https://resource.sgu.se/dokument/publikation/sgurapport/sgurapport202131rapport/s2131-rapport.pdf

Broms, B. B. (1964). Lateral resistance of piles in cohesive soils. *Journal of the Soil Mechanics and Foundations Division*, *90*(2), 27–63. https://doi.org/10.1061/JSFEAQ.0000611

Broms, B. B., & Asce, M. (1964). Lateral resistance of piles in cohesionless soils. *Journal of the Soil Mechanics and Foundations Division*.

Cecconi, M., Pane, V., Vecchietti, A., & Bellavita, D. (2019). Horizontal capacity of single piles: An extension of broms' theory for c- soils. *Soils and Foundations*.

Christensen, N. H., & Hansen, J. B. (1961). THE ULTIMATE RESISTANCE OF RIGID PILES AGAINST TRANSVERSAL FORCES. (12), 16. https://www.geo.dk/bibliotek/196x/the-ultimate-resistance-of-rigid-piles-against-transversal-forces-model-tests-with-transversally-loaded-ridig-piles-in-sand/#t=article

Design of timber structures - volume 1 (2nd ed.). (2016). Swedish Wood. Retrieved May 8, 2025, from https://www.svenskttra.se/siteassets/5-publikationer/pdfer/design-of-timber-structures-1-2016.pdf

Erling Reinius. (1963). *Vattenbyggnad - hamnar och farleder* (3rd ed.).

Eurocode 5: Design of timber structures - part 1-1: General - common rules and rules for buildings. (2009, May 20).

Eurokod 1: Laster på bärverk - del 2: Trafiklast på broar. (2011, May).

Exploateringsförvaltningen, Göteborgs stad. (2024, March). *Gång- och cykelbro packhuskajen - hugo hammars kaj, PM geoteknik* (Geotechnical report No. Dnr: EXF-2023-01034). Gothenburg. Retrieved February 5, 2025, from https://www4.goteborg.se/prod/intraservice/namndhandlingar/samrumportal.nsf/93ec9160f537fa30c12572aa004b6c1a/088fefcb4564db10c1258b1f003ecd25/$FILE/013.%20PM%20Geoteknik%20inkl%20bilaga.pdf

Goldbohm, P. C., M. Wolfert, A. R., G. De Gijt, J., De Bruijne, M. L., & Van Heesch, M. A. (2018). Beneficial inner-city quay walls? *Journal of Advanced Civil and Environmental Engineering*, *1*(1), 1. https://doi.org/10.30659/jacee.1.1.1-8

Göteborg Stad - Teknisk handbok (2024, October 23). Retrieved June 3, 2025, from tekniskhandbok.goteborg.se

Göteborg stad. (2025, February 5). *Göteborg stad - kanalmurarna* [Goteborg.se]. Retrieved February 5, 2025, from https://goteborg.se/wps/portal/start/goteborg-vaxer/hitta-projekt/stadsomrade-centrum/centrum/kanalmurarna

Hemel, M. (2023). *Amsterdam quays under pressure* [Doctoral dissertation, Delft University of Technology]. https://doi.org/10.4233/UUID:102EDFF8-8960-4633-830C-369AEF8E279F

Hemel, M.-J., Korff, M., & Peters, D. J. (2022). Analytical model for laterally loaded pile groups in layered sloping soil. *Marine Structures*, *84*, 103229. https://doi.org/10.1016/j.marstruc.2022.103229

Hougaard, C. E. (2024, June 27). *FEM-design WIKI* [Analysis]. Retrieved May 15, 2025, from https://wiki.fem-design.strusoft.com/xwiki/bin/view/Manuals/User%20Manual/Analysis/

Including nonlinearity in an ABAQUS analysis [ABAQUS online documentation: Version 6.6-1]. (2006, March 1). Retrieved May 15, 2025, from https://classes.engineering.wustl.edu/2009/spring/mase5513/abaqus/docs/v6.6/books/gss/default.htm?startat=ch07s03.html

Jaime P., A., & Fernández O., A. R. (2015). Manual de diseño de obras civiles. cap. b.2.6 estructuras de retención.

Korff, M., Hemel, M.-J., & Peters, D. J. (2022). Collapse of the grimburgwal, a historic quay in amsterdam, the netherlands. *Proceedings of the Institution of Civil Engineers - Forensic Engineering*, *175*(4), 96–105. https://doi.org/10.1680/jfoen.21.00018

Larsson, R. (2007, December). *SGI-information1-jords egenskaper* (Skrift No. 5). Statens geotekniska institut (SGI). Linköping.

Luongo, D., Nicodemo, G., Venmans, A., Korff, M., Sartorelli, L., Maljaars, H., & Peduto, D. (2025). The quay walls of amsterdam, netherlands: An approach for collapse risk mitigation at the municipal scale based on multisource monitoring and surveying data. *Journal of Geotechnical and Geoenvironmental Engineering*, *151*(2), 05024014. https://doi.org/10.1061/JGGEFK.GTENG-12981

Olsson, C., & Holm, G. (1993, May). *Pålgrundläggning* (Vol. 1993). Svensk Byggtjänst AB och Statens geotekniska institut.

PLAXIS - scientific manual 2d. (2024, May 7). Retrieved May 15, 2025, from https://bentleysystems.service-now.com/community?id=kb_article&sysparm_article=KB0107989

Reed L., M., & Dawkins, W. P. (2000, November). *Theoretical manual for pile foundations*. Geotechnical; Structures Laboratory U.S. Army Engineer Research; Development Center, Oklahoma State University.

Rollins, K. M., Peterson, K. T., & Weaver, T. J. (1998). Lateral load behavior of full-scale pile group in clay. *Journal of Geotechnical and Geoenvironmental Engineering*, *124*(6), 468–478. https://doi.org/10.1061/(ASCE)1090-0241(1998)124:6(468)

Sharma, S., Longo, M., & Messali, F. (2023). A novel tier-based numerical analysis procedure for the structural assessment of masonry quay walls under traffic loads. *Frontiers in Built Environment*, *9*, 1194658. https://doi.org/10.3389/fbuil.2023.1194658

Soutsos, M., & Domone, P. (2017). *Construction materials : Their nature and behaviour, fifth edition*. Taylor & Francis Group. http://ebookcentral.proquest.com/lib/chalmers/detail.action?docID=5475737

Svahn, P.-O., & Alén, C. (2006). *Transversalbelastade pålar - statiskt verkningssätt och dimensioneringsanvisningar* (No. 101) (ISSN 0347-1047 ISRN IVA/PAL/R—06–SE). Pålkommisionen. Linköping.

Bro och broliknande konstruktion, Bärighetsberäkning (2023, January 11). Retrieved February 6, 2025, from https://puben.trafikverket.se/dpub/visa-dokument/c3b4701a-7908-4791-b5cb-82de7fd0e448

van Hulten, C. (2021, November 30). *Recognizing critically damaged quay wall structures using a three-dimensional numerical model* [Master's Thesis]. Delft University of Technology.

Vårt Göteborg, Göteborgs stads tidning. (2023, December 18). *Vårt göteborg* [Vartgoteborg.se]. Retrieved February 5, 2025, from https://vartgoteborg.se/p/nar-kanalmurarna-rustas-upp-frigors-ytor-till-hamnstrak/

A Rheological constants

Rheological constants for different soil conditions can be found in Table A.1.

	Peat	Clay	Loam	Sand	Gravel
Over consolidated	-	1	2/3	1/2	1/3
Normally consolidated	1	2/3	1/2	1/3	1/4
Decomposed, weathered	-	1/2	1/2	1/3	1/4

Table A.1: Values for the Rheological constant (Hemel, 2023)

B Brinch-Hansen - Bearing factors

$$K_q = \frac{K_q^0 + K_q^\infty \times \alpha_q \times \frac{z}{D}}{1 + \alpha_q \frac{z}{D}}$$

$$K_c = \frac{K_c^0 + K_c^\infty \times \alpha_c \times \frac{z}{D}}{1 + \alpha_c \frac{z}{D}}$$

$$K_q^0 = e^{(\frac{\pi}{2} \times \tan\varphi)} \times \cos(\varphi) \times \tan(\frac{\pi}{4} + \frac{\varphi}{2}) - e^{(-\frac{\pi}{4} + \varphi) \times \tan\varphi} \times \cos(\varphi) \times \tan(\frac{\pi}{4} - \frac{\varphi}{2})$$

$$K_c^0 = \{e^{(\frac{\pi}{2} + \varphi)} \times \cos(\varphi) \times \tan(\frac{\pi}{4} + \frac{\varphi}{2}) - 1] \times \cot(\varphi)\}$$

$$K_q^\infty = K_c^\infty \times K_c^\infty \times \tan(\varphi)$$

$$K_c^\infty = N_c d_c^\infty$$

$$d_c^\infty = 1.58 + 4.09 \times \tan^4(\varphi)$$

$$N_c = [e^{\pi \times \tan(\varphi)} \tan^2(\frac{\pi}{4} + \frac{\varphi}{2}) - 1] \times \cot(\varphi)$$

For OCR = 1, $K_0 = 1 - \sin(\varphi)$.

$$\alpha_q = \frac{K_0^\infty}{K_q^\infty - K_q^0} \times \frac{K_0 \times \sin(\varphi)}{\sin(\frac{\pi}{4} + \frac{\varphi}{2})}$$

$$\alpha_c = \frac{K_c^\infty}{K_c^\infty - K_c^0} \times 2\sin(\frac{\pi}{4} + \frac{\varphi}{2})$$

c = Cohesion [kN/m^2]

C Case study 1 - Python codes

Utilised Python code to compute the limit stress from the soil and to obtain the coordinates for the wedges, for the reference case, is shown below.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Jan 29 15:03:31 2025

@author: juan
"""

import numpy as np
import matplotlib.pyplot as plt
import random

"Data from Table 6.2 "

Depth = np.array([
                    [0, 1.8],
                    [1.8, 2.4],
                    [2.4, 6],
                    [6, 8],
                    [8, 10],
                    [10, 12],
                    [12, 13.6]
                   ]) #Depth ranges in [m]

   = np.array([ [18],
                [9],
                [1],
                [7.2],
                [7.9],
                [7.2],
                [10]
              ]) #   'values in [kN/m^3]
   =   *1e3 #   'values in [N/m^3]

q_c = np.array([ [0],
                 [0],
                 [200],
                 [200],
                 [5000],
                 [1000],
                 [10000]
                ]) #cone resistance [kPa]

c = np.array([ [0],
               [0],
               [6.2],
               [3.7],
               [2.2],
               [3.7],
               [0.1]
             ]) #Cohesion [kN/m^2]
```

```
c= c*1e3 #Cohesion [N/m^2]

   = np.array([ [32.5],
                [32.5],
                [14.9],
                [28.8],
                [25],
                [23.6],
                [35]
              ])



def   _c  (K0c, Kinfc,   ):
    "   must be in degrees"
         = ( K0c/(Kinfc-K0c) ) * 2 * np.sin( (np.pi/4) + (np.radians(  )
    /2) )
    return

def   _q  (K0q, Kinfq,   , K0):
    "   must be in degrees"
         = (K0q/(Kinfq-K0q)) * ( (K0 * np.sin(np.radians(  ))) / np.sin(
    (np.pi/4) + np.radians(  )/2) )

    return

def K_0 (  ):
    "   must be in degrees"
    K = 1 - np.sin(np.radians(  ))

    return K

def N_c(  ):
    "   must be in degrees"
    N = ( np.exp(np.pi * np.tan(np.radians(  ))) * np.tan( (np.pi/4 ) + (
    np.radians(  )/2) )**2 -1) * (1/np.tan(np.radians(  )))

    return N

def d_c_inf (  ):
    "   must be in degrees"
    d = 1.58 + 4.09 * np.tan(np.radians(  ))**4

    return d

def K_infc(Nc, d_c_inf):
    "   must be in degrees"
    K = Nc * d_c_inf

    return K

def K_infq (  , Kinfc, K_0):
    "   must be in degrees"

    K = Kinfc * K_0 * np.tan(np.radians(  ))

    return K
```

```
106
107 def K_0c (  ):
108
109     K = ( np.exp( ( (np.pi/2) + np.radians(  ) ) * np.tan(np.radians(  ))
        ) * np.cos(np.radians(  ) * np.tan( (np.pi/4)+(np.radians(  )/2) )
       -1 ) ) * (1/np.tan(np.radians(  )) )
110
111     return K
112
113 def K_0q(  ):
114
115     K = ( np.exp( ( (np.pi/2) + np.radians(  ) ) * np.tan( np.radians(  )
       ) ) * np.cos( np.radians(  ) ) * np.tan( (np.pi/4) + (np.radians(  )
       /2) ) ) - np.exp( (-(np.pi/2) + np.radians(  ) ) * np.tan( np.radians
       (  ) ) ) * np.cos( np.radians(  ) ) * np.tan( (np.pi/4) - (np.radians
       (  )/2) )
116
117     return K
118
119 def K_c (K0c,Kinfc,  _c  ,z,D):
120     """
121     Where:
122
123         z = Depth [m]
124         D = Pile Diameter [m]
125
126     """
127
128     K = ( K0c + Kinfc *   _c   * (z/D) ) / ( 1 +   _c   * (z/D) )
129
130     return K
131
132 def K_q (K0q,Kinfq,  _q  ,z,D):
133     """
134     Where:
135
136         z = Depth [m]
137         D = Pile Diameter [m]
138
139     """
140
141     K = ( K0q + Kinfq *   _q   * (z/D) ) / ( 1 +   _q   * (z/D) )
142
143     return K
144
145 def  _v (  , Depth_start, Depth_end, P0):
146
147        = P0 +    * ( Depth_end - Depth_start)
148
149     return
150
151 def K_a (  ):
152
153     K = np.tan(45 - (np.radians(  )/2) )
154
155     return K
156
```

```
def K_p (  ):

    K = np.tan(45 + (np.radians(  )/2) )

    return K

 v  = np.zeros([len(  )+1,1])

" To obtain vertical stres : "

for i in range (1,len ( v ) ):

    if i == 1:
         v [i,0] =  _v (  [i-1,0], Depth[i-1,0], Depth[i-1,1], 0)

    else:
         v [i,0] =  _v (  [i-1,0], Depth[i-1,0], Depth[i-1,1], 0) +  v [i
    -1,0]



dz = 0.1 # [m]

D = 0.20 # Pile diameter [m]

R = D / 2  # Radius




Deppts = np.zeros([len(Depth) + 1, 1])
Deppts[1:, 0] = Depth[:, 1]
Deppts = Deppts.flatten()
 v  =  v .flatten()

ndz = np.max(Depth[:,1])/dz
ndz= int(ndz)

 p_BH  = np.zeros(ndz+1)
Depth_dz = np.zeros([ndz+1,1])


for j in range(ndz):

    dzp = dz * (j)  # Ensures dzp increases predictably
    i = 0
    Depth_dz[j] = dzp

    while i < len(  ):

        if dzp <= Depth[i, 1]:
            phi =   [i]
            cdz = c[i, 0]

            K0 = K_0(phi)
            K0q = K_0q(phi)
            K0c = K_0c(phi)
```

```
            Nc = N_c(phi)
            dcinf = d_c_inf(phi)
            Kinfc = K_infc(Nc, dcinf)
            Kinfq = K_infq(phi, Kinfc, K0)
              c   =   _c  (K0c, Kinfc, phi)
              q   =   _q  (K0q, Kinfq, phi, K0)
            Kc = K_c(K0c, Kinfc,   c  , dzp, D)
            Kq = K_q(K0q, Kinfq,   q  , dzp, D)
             vp  = np.interp(dzp, Deppts,  v )

            # Ensure all variables are scalars
            Kq = Kq.item() if isinstance(Kq, np.ndarray) else Kq
             vp  =  vp .item() if isinstance( vp , np.ndarray) else  vp
            Kc = Kc.item() if isinstance(Kc, np.ndarray) else Kc
            cdz = cdz.item() if isinstance(cdz, np.ndarray) else cdz

             p_BH [j] = Kq *  vp  + Kc * cdz  # Compute  p_BH

            # Print debugging info
            #print(f'j={j}, i={i}, dzp={dzp},  vp ={ vp }, Kq={Kq}, Kc={
   Kc}, cdz={cdz},  p_BH [{j}]={ p_BH [j]}')
            break  # Breaks from the while loop since we found a valid `i
   `

        else:
            i += 1
            if i >= len(  ):
                break

" To compute plastic stresses for Brom's extention method "

 p_BE  = np.zeros(ndz+1) #Brom's extention method

for j in range(ndz+1):

    dzp = dz * (j)  # Ensures dzp increases predictably
    i = 0
    #Depth_dz[j] = dzp
         = 3.35

    while i < len(  ):

        if dzp <= Depth[i, 1]:

            phi =   [i]
            cdz = c[i, 0]
             z  =   [i]

             p_BE [j] =      *  z  * dzp * ( K_p(phi) - K_a(phi) ) +
    * 2 * cdz *( np.sqrt( K_p(phi) ) + np.sqrt( K_a(phi) ) )

            break

        else:
            i += 1
            if i >= len(  ):
                break
```

```
267
268
269 Max=len(Depth_dz)-1 #Max plotting index value
270 " Plotting Stress "
271 plt.figure()
272 plt.plot( v ,-Deppts, label='Overburden pressure',color='b')
273 plt.xlabel( 'Vertical Stress [Pa]')
274 plt.ylabel( "Depth [m]")
275 plt.title('Stress')
276 plt.legend()
277
278 " Plotting Stress "
279 plt.figure()
280 plt.plot( p_BE [:Max],Depth_dz[:Max], label='Plastic stress Broms
    modified',color='b',linestyle='-')
281 plt.xlabel( 'Vertical Stress [Pa]')
282 plt.ylabel( "Depth [m]")
283 plt.title('Plastic Stress')
284 plt.gca().invert_yaxis()
285 plt.legend()
286
287 " Plotting Stress "
288 plt.figure()
289 plt.plot( p_BH [:Max],Depth_dz[:Max], label='Plastic stress BH',color='r
    ', linestyle='--')
290 plt.plot( p_BE [:Max],Depth_dz[:Max], label='Plastic stress Broms
    modified',color='b',linestyle='-')
291 plt.xlabel( 'Vertical Stress [Pa]')
292 plt.ylabel( "Depth [m]")
293 plt.title('Plastic Stress')
294 plt.gca().invert_yaxis()
295 plt.legend()
296
297 " To compute the coordinates of each depth"
298
299 Coords = np.zeros([ndz +1,3]) #Coordinates matrix representing x, y, z
300 Coords[:,2] = Depth_dz[:,0]
301
302
303
304 X_Coords = np.zeros([ndz+1,ndz+1])
305 Y_Coords = np.zeros_like(X_Coords)
306
307 for col in range( ndz ):
308
309     Coords = Coords * 0
310
311
312     for i in range( ndz - 1 - col, -1, -1):  # Range in ( amount of dz's,
    -1, -1)
313
314         if Depth_dz [i,0] > Depth[5,1]:
315
316             j = 6
317
318         elif Depth_dz [i,0] > Depth[4,1]:
319
```

```
            j = 5

        elif Depth_dz [i,0] > Depth[3,1]:

            j = 4

        elif Depth_dz [i,0] > Depth[2,1]:

            j = 3

        elif Depth_dz [i,0] > Depth[1,1]:

            j = 2

        elif Depth_dz [i,0] > Depth[0,1]:

            j = 1

        if c[j,0] != 0:
             _m  = np.radians(  [j,0])/5
             _m  = np.radians(45) +  _m /2

        else:
             _m  = np.radians(  [j,0]/3)
             _m  = np.radians(45) +  _m /2

        if i == ndz-1:

            Coords[i,0] = np.tan( _m ) * dz
            Coords[i,1] = np.tan( _m ) * Coords[i,0]

        else:

            Coords[i,0] = np.tan( _m ) * dz + Coords [i+1,0]
            Coords[i,1] = np.tan( _m ) * np.tan( _m ) * dz + Coords[i
    +1,1]



     X_Coords[:,col] = Coords[:,0]
     Y_Coords[:,col] = Coords[:,1]


" For comparisson "

BH_comp =  p_BH [0:75]
BM_comp =  p_BE [0:75]

D_dz_comp = Depth_dz[0:75]

" Plotting Stress "
plt.figure()
plt.plot(BH_comp,D_dz_comp, label='Plastic stress BH',color='r',
    linestyle='--')
plt.plot(BM_comp,D_dz_comp, label='Plastic stress Broms modified',color='
    b',linestyle='-.')
plt.xlabel( 'Vertical Stress [Pa]')
```

```
plt.ylabel( "Depth [m]")
plt.title('Plastic Stress')
plt.gca().invert_yaxis()
plt.legend()

"___Wedge calculations___"

# Create 3D figure
fig = plt.figure(figsize=(8, 6))
#%matplotlib tk
ax = fig.add_subplot(111, projection='3d')

# Generate cylinder surface
theta = np.linspace(0, 2 * np.pi, 50)  # Angles for circular base
height = np.max(Coords[:, 2]) - np.min(Coords[:, 2])  # Cylinder height
z = np.linspace(-height, 0, 50)  # Cylinder height (negative to match -
    Coords[:,2])
Theta, Z = np.meshgrid(theta, z)  # Create a grid

X = R * np.cos(Theta)  # X coordinates
Y = R * np.sin(Theta)  # Y coordinates

# Plotting the coordinates
ax.plot(Coords[:, 0], Coords[:, 1], -Coords[:, 2], label="Original Data",
     color="r")

# Plot the cylinder
ax.plot_wireframe(X, Y, Z, color='b', alpha=0.3)  # Semi-transparent blue

# Add labels and title
ax.set_xlabel('X Coordinate')
ax.set_ylabel('Y Coordinate')
ax.set_zlabel('Z Coordinate')
ax.set_title('Interactive 3D Plot')

# Show legend
ax.legend()

# Display the plot
plt.show()

Coords[:,1] =Coords[:,1] + 0.5 * D

"""
"Plotting in 3D "
from mpl_toolkits.mplot3d import Axes3D
%matplotlib tk

# Create the 3D plot
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')

# Plotting the coordinates
Plott = ax.plot(Coords[:, 0], Coords[:, 1], -Coords[:, 2])

# Add labels and title
ax.set_xlabel('X Coordinate')
```

```
ax.set_ylabel('Y Coordinate')
ax.set_zlabel('Z Coordinate')
ax.set_title('Interactive 3D Plot')

# Show legend
ax.legend()

# Display the plot
plt.show()

"""

"_______Save Coordinates to text_______"

np.savetxt("X_Coords", X_Coords[:,200-74])
np.savetxt("Y_Coords", Y_Coords[:,200-74]+0.1)
np.savetxt("X_Coords2", X_Coords[:,200-75])
np.savetxt("Y_Coord2s", Y_Coords[:,200-75]+0.1)
np.savetxt("Z_Coords", Depth_dz )
```

The python code utilised to obtain the reduction factors from Grasshopper is shown below.

```
# -*- coding: utf-8 -*-
"""
Created on Tue Apr  8 16:00:34 2025

@author: SE1G4E
"""

import numpy as np
import pandas as pd


" Importing from Excel worksheet data from Grasshopper "

df = pd.read_excel('Wedges volumes.xlsx', engine='openpyxl')

V_Sand_P1_UR = df['V Sand P1 UR']
V_Sand_P1_UR = V_Sand_P1_UR[:16].to_numpy() #Wedges volumes from P1
    Unreduced Sand

V_Clay_P1_UR = df['V Clay P1 UR']
V_Clay_P1_UR = V_Clay_P1_UR[:36].to_numpy() #Wedges volumes from P1
    Unreduced Clay

V_WS_P1_UR = df['V Wadsand P1 UR']
V_WS_P1_UR = V_WS_P1_UR[:56].to_numpy() #Wedges volumes from P1 Unreduced
     Wadsand

V_SC_P1_UR = df['V Sea Clay P1 UR']
V_SC_P1_UR = V_SC_P1_UR[:76].to_numpy() #Wedges volumes from P1 Unreduced
     Sea Clay

V_Peat_P1_UR = df['V Peat P1 UR']
V_Peat_P1_UR = V_Peat_P1_UR[:109].to_numpy() #Wedges volumes from P1
    Unreduced Peat

V_Sand_P2_UR = df['V Sand P2 UR']
V_Sand_P2_UR = V_Sand_P2_UR[:16].to_numpy() #Wedges volumes from P2
    Unreduced Sand

V_Clay_P2_UR = df['V Clay P2 UR']
V_Clay_P2_UR = V_Clay_P2_UR[:36].to_numpy() #Wedges volumes from P2
    Unreduced Clay

V_WS_P2_UR = df['V Wadsand P2 UR']
V_WS_P2_UR = V_WS_P2_UR[:56].to_numpy() #Wedges volumes from P2 Unreduced
     Wadsand

V_SC_P2_UR = df['V Sea Clay P2 UR']
V_SC_P2_UR = V_SC_P2_UR[:76].to_numpy() #Wedges volumes from P2 Unreduced
     Sea Clay

V_Peat_P2_UR = df['V Peat P2 UR']
V_Peat_P2_UR = V_Peat_P2_UR[:109].to_numpy() #Wedges volumes from P2
    Unreduced Peat
```

```
45
46 V_Sand_P1_R = df['V Sand P1 R']
47 V_Sand_P1_R = V_Sand_P1_R[:16].to_numpy() #Wedges volumes from P1 Reduced
       Sand
48
49 V_Clay_P1_R = df['V Clay P1 R']
50 V_Clay_P1_R = V_Clay_P1_R[:43].to_numpy() #Wedges volumes from P1 Reduced
       Clay
51
52 V_WS_P1_R = df['V Wadsand P1 R']
53 V_WS_P1_R = V_WS_P1_R[:83].to_numpy() #Wedges volumes from P1 Reduced
      Wadsand
54
55 V_SC_P1_R = df['V Sea Clay P1 R']
56 V_SC_P1_R = V_SC_P1_R[:115].to_numpy() #Wedges volumes from P1 Reduced
      Sea Clay
57
58 V_Peat_P1_R = df['V Peat P1 R']
59 V_Peat_P1_R = V_Peat_P1_R[:119].to_numpy() #Wedges volumes from P1
      Reduced Peat
60
61 V_Sand_P2_R = df['V Sand P2 R']
62 V_Sand_P2_R = V_Sand_P2_R[:16].to_numpy() #Wedges volumes from P2 Reduced
       Sand
63
64 V_Clay_P2_R = df['V Clay P2 R']
65 V_Clay_P2_R = V_Clay_P2_R[:36].to_numpy() #Wedges volumes from P2 Reduced
       Clay
66
67 V_WS_P2_R = df['V Wadsand P2 R']
68 V_WS_P2_R = V_WS_P2_R[:56].to_numpy() #Wedges volumes from P2 Reduced
      Wadsand
69
70 V_SC_P2_R = df['V Sea Clay P2 R']
71 V_SC_P2_R = V_SC_P2_R[:76].to_numpy() #Wedges volumes from P2 Reduced Sea
       Clay
72
73 V_Peat_P2_R = df['V Peat P2 R']
74 V_Peat_P2_R = V_Peat_P2_R[:109].to_numpy() #Wedges volumes from P2
      Reduced Peat
75
76 A_Sand_P1_UR = df['A Sand P1 UR']
77 A_Sand_P1_UR = A_Sand_P1_UR[:16].to_numpy() #Wedges Areas from P1
      Unreduced Sand
78
79 A_Clay_P1_UR = df['A Clay P1 UR']
80 A_Clay_P1_UR = A_Clay_P1_UR[:36].to_numpy() #Wedges Areas from P1
      Unreduced Clay
81
82 A_WS_P1_UR = df['A Wadsand P1 UR']
83 A_WS_P1_UR = A_WS_P1_UR[:56].to_numpy() #Wedges Areas from P1 Unreduced
      Wadsand
84
85 A_SC_P1_UR = df['A Sea Clay P1 UR']
86 A_SC_P1_UR = A_SC_P1_UR[:76].to_numpy() #Wedges Areas from P1 Unreduced
      Sea Clay
87
```

```
A_Peat_P1_UR = df['A Peat P1 UR']
A_Peat_P1_UR = A_Peat_P1_UR[:109].to_numpy() #Wedges Areas from P1 Unreduced Peat

A_Sand_P2_UR = df['A Sand P2 UR']
A_Sand_P2_UR = A_Sand_P2_UR[:16].to_numpy() #Wedges Areas from P2 Unreduced Sand

A_Clay_P2_UR = df['A Clay P2 UR']
A_Clay_P2_UR = A_Clay_P2_UR[:36].to_numpy() #Wedges Areas from P2 Unreduced Clay

A_WS_P2_UR = df['A Wadsand P2 UR']
A_WS_P2_UR = A_WS_P2_UR[:56].to_numpy() #Wedges Areas from P2 Unreduced Wadsand

A_SC_P2_UR = df['A Sea Clay P2 UR']
A_SC_P2_UR = A_SC_P2_UR[:76].to_numpy() #Wedges Areas from P2 Unreduced Sea Clay

A_Peat_P2_UR = df['A Peat P2 UR']
A_Peat_P2_UR = A_Peat_P2_UR[:109].to_numpy() #Wedges Areas from P2 Unreduced Peat

A_Sand_P1_R = df['A Sand P1 R']
A_Sand_P1_R = A_Sand_P1_R[:18].to_numpy() #Wedges Areas from P1 Unreduced Sand

A_Clay_P1_R = df['A Clay P1 R']
A_Clay_P1_R = A_Clay_P1_R[:44].to_numpy() #Wedges Areas from P1 Unreduced Clay

A_WS_P1_R = df['A Wadsand P1 R']
A_WS_P1_R = A_WS_P1_R[:84].to_numpy() #Wedges Areas from P1 Unreduced Wadsand

A_SC_P1_R = df['A Sea Clay P1 R']
A_SC_P1_R = A_SC_P1_R[:114].to_numpy() #Wedges Areas from P1 Unreduced Sea Clay

A_Peat_P1_R = df['A Peat P1 R']
A_Peat_P1_R = A_Peat_P1_R[:121].to_numpy() #Wedges Areas from P1 Unreduced Peat

A_Sand_P2_R = df['A Sand P2 R']
A_Sand_P2_R = A_Sand_P2_R[:16].to_numpy() #Wedges Areas from P2 Unreduced Sand

A_Clay_P2_R = df['A Clay P2 R']
A_Clay_P2_R = A_Clay_P2_R[:36].to_numpy() #Wedges Areas from P2 Unreduced Clay

A_WS_P2_R = df['A Wadsand P2 R']
A_WS_P2_R = A_WS_P2_R[:56].to_numpy() #Wedges Areas from P2 Unreduced Wadsand

A_SC_P2_R = df['A Sea Clay P2 R']
```

```
A_SC_P2_R = A_SC_P2_R[:76].to_numpy() #Wedges Areas from P2 Unreduced Sea
    Clay

A_Peat_P2_R = df['A Peat P2 R']
A_Peat_P2_R = A_Peat_P2_R[:109].to_numpy() #Wedges Areas from P2
   Unreduced Peat


" Impoting Soil Data "

from Plastic_Stress import   , c

 _Sand  =   [6,0].item()
 _Clay  =   [5,0].item()
 _WS  =   [4,0].item()
 _SC  =   [3,0].item()
 _Peat  =   [2,0].item()

c_Sand = c[6,0].item()
c_Clay = c[5,0].item()
c_WS = c[4,0].item()
c_SC = c[3,0].item()
c_Peat = c[2,0].item()


" Fixing the unregularities from data "
"_________________Pile 1_________________"

" For Clay Reduced Volumes "

ClayR = V_Clay_P1_R[::-1] #Fliping the vector due to index from GH are
   fliped
V_Clay_P1_R = np.zeros([43-7,1])

j = 0

for i in range (0, 14, 2):

    Suma = ClayR[i] + ClayR[i+1]
    Suma = Suma.item()

    V_Clay_P1_R [j] = Suma

    j = j + 1

V_Clay_P1_R[7:,0] = ClayR[14:]
V_Clay_P1_R = V_Clay_P1_R[::-1] #Fliping it back again (Top cell closser
   to surface)

" For Waadsand Reduced Volumes "

WSR = V_WS_P1_R[::-1] #Fliping the vector due to index from GH are fliped
V_WS_P1_R = np.zeros([83-27,1])

j = 0

for i in range (0, 54, 2):
```

```
184
185     Suma = WSR[i] + WSR[i+1]
186     Suma = Suma.item()
187
188     V_WS_P1_R [j] = Suma
189
190     j = j + 1
191
192 V_WS_P1_R[27:,0] = WSR[54:]
193 V_WS_P1_R = V_WS_P1_R[::-1] #Fliping it back again (Top cell closser to
        surface)
194
195 " For Sea Clay Reduced Volumes "
196
197 SCR = V_SC_P1_R[::-1] #Fliping the vector due to index from GH are fliped
198 V_SC_P1_R = np.zeros([115-43,1])
199
200 j = 0
201
202 for i in range (0, 4, 2):
203
204     Suma = SCR[i] + SCR[i+1]
205     Suma = Suma.item()
206
207     V_SC_P1_R [j] = Suma
208
209     j = j + 1
210
211 V_SC_P1_R[j] = SCR[4] + SCR [5]
212
213 j = j + 1
214
215 for i in range (6, 84, 2):
216
217     Suma = SCR[i] + SCR[i+1]
218     Suma = Suma.item()
219
220     V_SC_P1_R [j] = Suma
221
222     j = j + 1
223
224 V_SC_P1_R[j] = SCR[84] + SCR [85]
225
226 V_SC_P1_R[43:,0] = SCR[86:]
227 V_SC_P1_R = V_SC_P1_R[::-1] #Fliping it back again (Top cell closser to
        surface)
228
229 " For Peat Reduced Volumes "
230
231 PeatR = V_Peat_P1_R[::-1] #Fliping the vector due to index from GH are
        fliped
232 V_Peat_P1_R = np.zeros([120-39,1])
233
234 j = 0
235
236 for i in range (0, 78, 2):
237
```

```
    Suma = PeatR[i] + PeatR[i+1]
    Suma = Suma.item()

    V_Peat_P1_R [j] = Suma

    j = j + 1

V_Peat_P1_R[39:,0] = PeatR[77:]
V_Peat_P1_R = V_Peat_P1_R[::-1] #Fliping it back again (Top cell closser
    to surface)


" Obtaining reduction factors "

" For un-reducted Pile 1 Weights "

ndz = 109
WP1_UR = np.zeros([ndz,1])

WP1_UR[(ndz-len(V_Sand_P1_UR)):,0] = V_Sand_P1_UR *  _Sand
WP1_UR[(ndz-len(V_Clay_P1_UR)):,0] += V_Clay_P1_UR *  _Clay
WP1_UR[(ndz-len(V_WS_P1_UR)):,0] += V_WS_P1_UR *  _WS
WP1_UR[(ndz-len(V_SC_P1_UR)):,0] += V_SC_P1_UR *  _SC
WP1_UR[(ndz-len(V_Peat_P1_UR)):,0] += V_Peat_P1_UR *  _Peat

" For un-reducted Pile 2 Weights "

WP2_UR = np.zeros([ndz,1])

WP2_UR[(ndz-len(V_Sand_P2_UR)):,0] = V_Sand_P2_UR *  _Sand
WP2_UR[(ndz-len(V_Clay_P2_UR)):,0] += V_Clay_P2_UR *  _Clay
WP2_UR[(ndz-len(V_WS_P2_UR)):,0] += V_WS_P2_UR *  _WS
WP2_UR[(ndz-len(V_SC_P2_UR)):,0] += V_SC_P2_UR *  _SC
WP2_UR[(ndz-len(V_Peat_P2_UR)):,0] += V_Peat_P2_UR *  _Peat

" For reducted Pile 2 "

WP2_R = np.zeros([ndz,1])

WP2_R[(ndz-len(V_Sand_P2_R)):,0] = V_Sand_P2_R *  _Sand
WP2_R[(ndz-len(V_Clay_P2_R)):,0] += V_Clay_P2_R *  _Clay
WP2_R[(ndz-len(V_WS_P2_R)):,0] += V_WS_P2_R *  _WS
WP2_R[(ndz-len(V_SC_P2_R)):,0] += V_SC_P2_R *  _SC
WP2_R[(ndz-len(V_Peat_P2_R)):,0] += V_Peat_P2_R *  _Peat

" For reducted Pile 1 "

WP1_R = np.zeros([ndz,1])

WP1_R[(ndz-len(V_Sand_P1_R)):,0] = V_Sand_P1_R *  _Sand

#Clay Contribution
ClayR = np.zeros_like(V_Clay_P1_UR)
ClayR[:len(V_Clay_P1_R)] = V_Clay_P1_R[:,0]

WP1_R[(ndz-len(ClayR)):,0] += ClayR[:] *  _Clay

```

```
#Waadsand Contribution
WSR = np.zeros_like(V_WS_P1_UR)
WSR[:len(V_WS_P1_R)] = V_WS_P1_R[:,0]

WP1_R[(ndz-len(WSR)):,0] += WSR[:] *  _WS

#Sea Clay contribution
SCR = np.zeros_like(V_SC_P1_UR)
SCR[:len(V_SC_P1_R)] = V_SC_P1_R[:,0]

WP1_R[(ndz-len(SCR)):,0] += SCR[:] *  _SC

#Peat Contribution

PeatR = np.zeros_like(V_Peat_P1_UR)
PeatR[:len(V_Peat_P1_R)] = V_Peat_P1_R[:,0]

WP1_R[(ndz-len(PeatR)):,0] += PeatR[:] *  _Peat

"_____________________________________________________For Areas/
   Shear_____________________________________________________ "

" Fixing the unregularities from data "
"_______________Pile 1_____________"

" For Sand Reduced Areas "

SandR = A_Sand_P1_R
A_Sand_P1_R = np.zeros_like(A_Sand_P1_UR)

A_Sand_P1_R[:15] = SandR[:15]
A_Sand_P1_R[15] = np.sum(SandR[ [15,16,17] ])

" For Clay Reduced Areas "

ClayR = A_Clay_P1_R[::-1] #Fliping the Vector due to index from GH are
   fliped
A_Clay_P1_R = np.zeros_like(A_Clay_P1_UR)

j = 0

for i in range (0, 16, 2):

    Suma = ClayR[i] + ClayR[i+1]
    Suma = Suma.item()

    A_Clay_P1_R [j] = Suma

    j = j + 1

A_Clay_P1_R[7:] = ClayR[15:]
A_Clay_P1_R = A_Clay_P1_R[::-1] #Fliping it back again (Top cell closser
   to surface)

" For Waadsand Reduced Areas "

WSR = A_WS_P1_R[::-1] #Fliping the Vector due to index from GH are fliped
```

```
A_WS_P1_R = np.zeros_like(A_WS_P1_UR)

j = 0

for i in range (0, 56, 2):

    Suma = WSR[i] + WSR[i+1]
    Suma = Suma.item()

    A_WS_P1_R [j] = Suma

    j = j + 1

A_WS_P1_R[28:] = WSR[56:]
A_WS_P1_R = A_WS_P1_R[::-1] #Fliping it back again (Top cell closser to
    surface)

" For Sea Clay Reduced Areas "

SCR = A_SC_P1_R[::-1] #Fliping the Aector due to index from GH are fliped
A_SC_P1_R = np.zeros([114-43])

j = 0

for i in range (0, 86, 2):

    Suma = SCR[i] + SCR[i+1]
    Suma = Suma.item()

    A_SC_P1_R [j] = Suma

    j = j + 1


A_SC_P1_R[43:] = SCR[86:]
A_SC_P1_R = A_SC_P1_R[::-1] #Fliping it back again (Top cell closser to
    surface)

" For Peat Reduced Aolumes "

PeatR = A_Peat_P1_R[::-1] #Fliping the Aector due to index from GH are
    fliped
A_Peat_P1_R = np.zeros([119-39])

j = 0

for i in range (0, 78, 2):

    Suma = PeatR[i] + PeatR[i+1]
    Suma = Suma.item()

    A_Peat_P1_R [j] = Suma

    j = j + 1

A_Peat_P1_R[39:71] = PeatR[78:110]
A_Peat_P1_R[71:] = PeatR[112:]
```

```
A_Peat_P1_R = A_Peat_P1_R[::-1] #Fliping it back again (Top cell closser
    to surface)


" Obtaining reduction factors "

" For un-reducted Pile 1 Areas "

ndz = 109
 P1_UR  = np.zeros([ndz,1])

 P1_UR [(ndz-len(A_Sand_P1_UR)):,0] = A_Sand_P1_UR * c_Sand
 P1_UR [(ndz-len(A_Clay_P1_UR)):,0] += A_Clay_P1_UR * c_Clay
 P1_UR [(ndz-len(A_WS_P1_UR)):,0] += A_WS_P1_UR * c_WS
 P1_UR [(ndz-len(A_SC_P1_UR)):,0] += A_SC_P1_UR * c_SC
 P1_UR [:,0] += A_Peat_P1_UR[:] * c_Peat

" For un-reducted Pile 2 Areas "

 P2_UR  = np.zeros([ndz,1])

 P2_UR [(ndz-len(V_Sand_P2_UR)):,0] = V_Sand_P2_UR * c_Sand
 P2_UR [(ndz-len(V_Clay_P2_UR)):,0] += V_Clay_P2_UR * c_Clay
 P2_UR [(ndz-len(V_WS_P2_UR)):,0] += V_WS_P2_UR * c_WS
 P2_UR [(ndz-len(V_SC_P2_UR)):,0] += V_SC_P2_UR * c_SC
 P2_UR [(ndz-len(V_Peat_P2_UR)):,0] += V_Peat_P2_UR * c_Peat

" For reducted Pile 2 "

 P2_R  = np.zeros([ndz,1])

 P2_R [(ndz-len(V_Sand_P2_R)):,0] = V_Sand_P2_R * c_Sand
 P2_R [(ndz-len(V_Clay_P2_R)):,0] += V_Clay_P2_R * c_Clay
 P2_R [(ndz-len(V_WS_P2_R)):,0] += V_WS_P2_R * c_WS
 P2_R [(ndz-len(V_SC_P2_R)):,0] += V_SC_P2_R * c_SC
 P2_R [(ndz-len(V_Peat_P2_R)):,0] += V_Peat_P2_R * c_Peat

" For reducted Pile 1 "

 P1_R  = np.zeros([ndz,1])

 P1_R [(ndz-len(A_Sand_P1_R)):,0] = A_Sand_P1_R * c_Sand

#Clay Contribution
ClayR = np.zeros_like(A_Clay_P1_UR)
ClayR[:len(A_Clay_P1_R)] = A_Clay_P1_R[:]

 P1_R [(ndz-len(ClayR)):,0] += ClayR[:] * c_Clay

#Waadsand Contribution
WSR = np.zeros_like(V_WS_P1_UR)
WSR[:len(A_WS_P1_R)] = A_WS_P1_R[:]

 P1_R [(ndz-len(WSR)):,0] += WSR[:] * c_WS

#Sea Clay contribution
SCR = np.zeros_like(A_SC_P1_UR)
```

```
458 SCR[:len(A_SC_P1_R)] = A_SC_P1_R[:]
459
460  P1_R [(ndz-len(SCR)):,0] += SCR[:] * c_SC
461
462 #Peat Contribution
463
464 PeatR = np.zeros_like(A_Peat_P1_UR)
465 PeatR[:len(A_Peat_P1_R)] = A_Peat_P1_R[:]
466
467  P1_R [:,0] += PeatR[:] * c_Peat
468
469  _w_P1  = np.zeros([ndz,1])
470
471  _w_P2  = np.zeros_like( _w_P1 )
472
473  _c_P1  = np.zeros_like( _w_P1 )
474
475  _c_P2  = np.zeros_like( _w_P1 )
476
477 for i in range(ndz):
478
479      _w_P1 [i,0] = WP1_R[i]/WP1_UR[i]
480      _w_P2 [i,0] = WP2_R[i]/WP2_UR[i]
481      _c_P1 [i,0] =  P1_R [i]/ P1_UR [i]
482      _c_P2 [i,0] =  P2_R [i]/ P2_UR [i]
483
484
485 CSV = pd.DataFrame({
486     'Psi_w_P1':  _w_P1 .flatten(),
487     'Psi_w_P2':  _w_P2 .flatten(),
488     'Psi_c_P1':  _c_P1 .flatten(),
489     'Psi_c_P2':  _c_P2 .flatten()
490     })
491
492 CSV.to_csv('Psi_factors.csv', index=False, sep=';')
```

D Grasshopper Wedge model - Intermediate steps

The following figures shows the intermediate steps to obtain the reduction wedges for the Grimburgwal case.

From the Python module, it was obtained the coordinates for each point of the wedges, create points from these coordinates, followed by polylines, see Figure D.1. From these polylines, surfaces were created on the ZX-plane, see Figure D.2. It is good to mention that this procedure is the same for both sides.

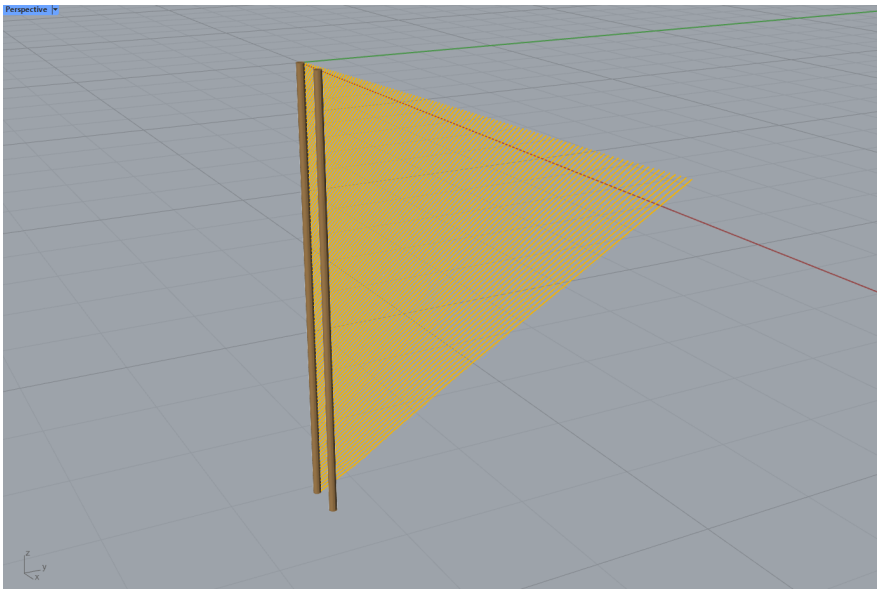


Figure D.1: Model of the polylines for each wedge for Pile 1 on the left side

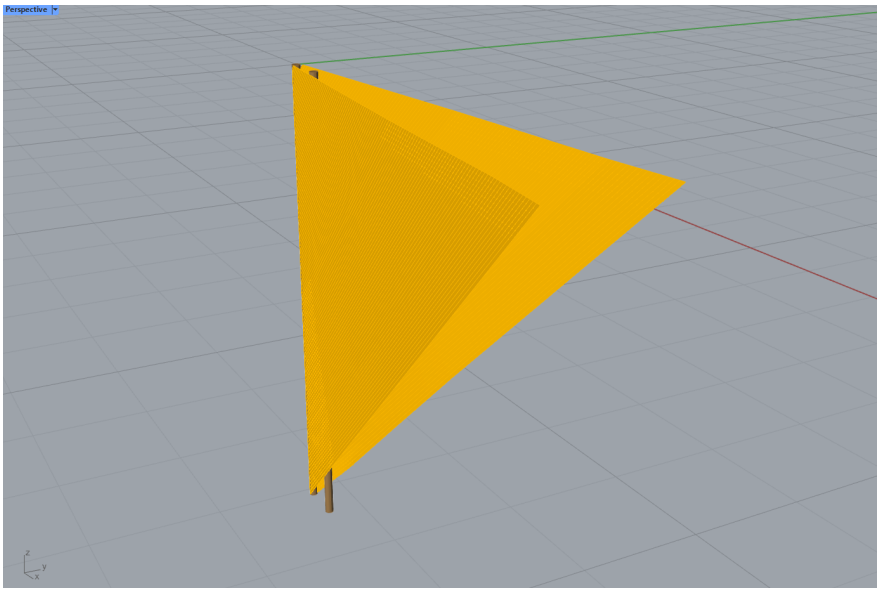


Figure D.2: Lateral surfaces modelled for Pile 1

In an similar way, the surfaces show in Figure D.3, that are mainly in the ZY-plane were modelled. These surfaces are, in a general perspective the shear surfaces. Finally, with the

lateral surfaces previously created and between each ZY-surface, it was closed every space between these different surface, creating various solids, which would represent the wedges.

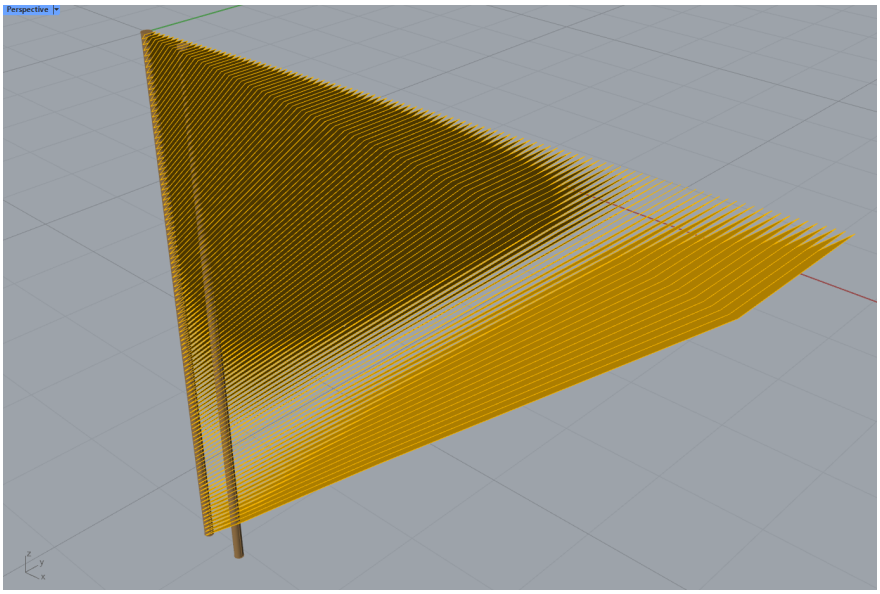


Figure D.3: ZY-plane surfaces for Pile 1

The reduction due to canal and pile group effect are shown in Figures D.4 and D.5, the pile rows are assumed to be separated 1 m. Then it was reduced from the general models 0.5 m from each side (left and right). The same way, it was modelled a volume to reduce the respective amount due to the canal topography, see Figure D.6, where the reduced volumes due to the canal are shown in Figures D.7 and D.8.

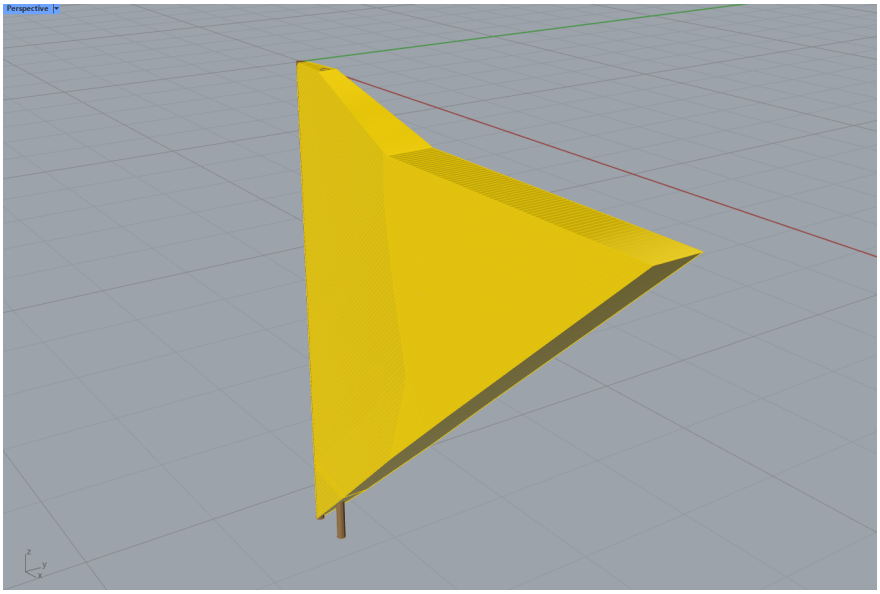


Figure D.4: Volume from Pile 1 reduced due to pile group and canal

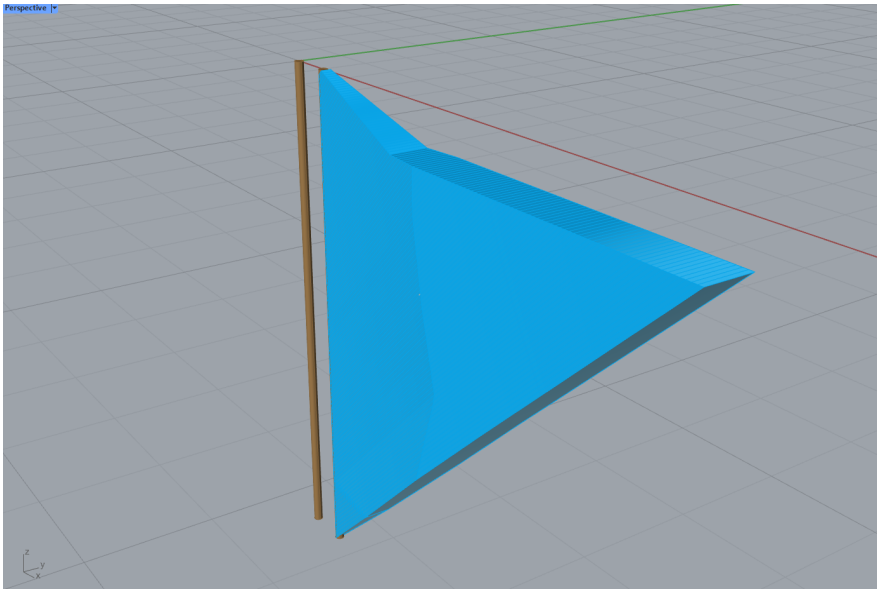


Figure D.5: Reduced soil volume from Pile 2

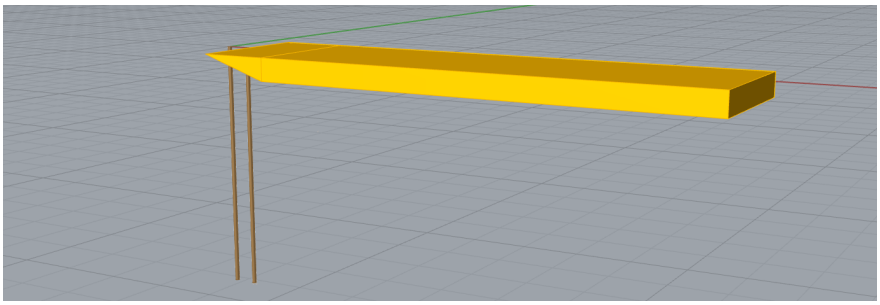


Figure D.6: Canal bed solid

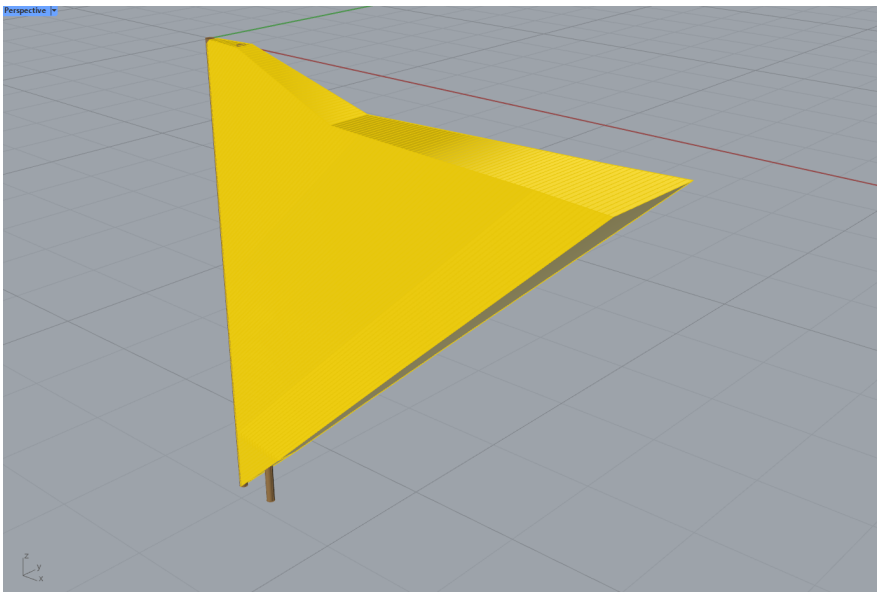


Figure D.7: Volume from soil Pile 1 reduced due to canal

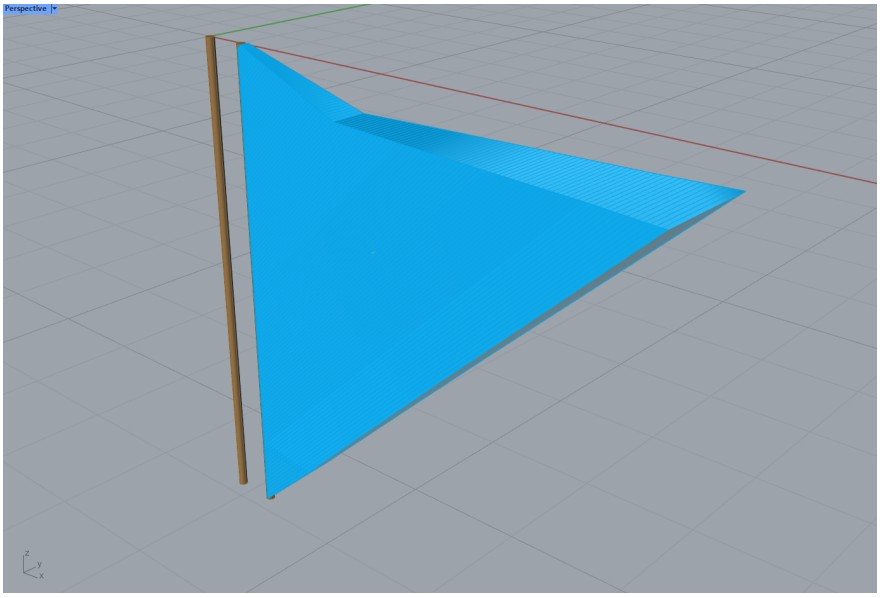


Figure D.8: Volume from soil Pile 2 reduced due to canal

The final reduce volumes of both piles are shown in Figure D.9.

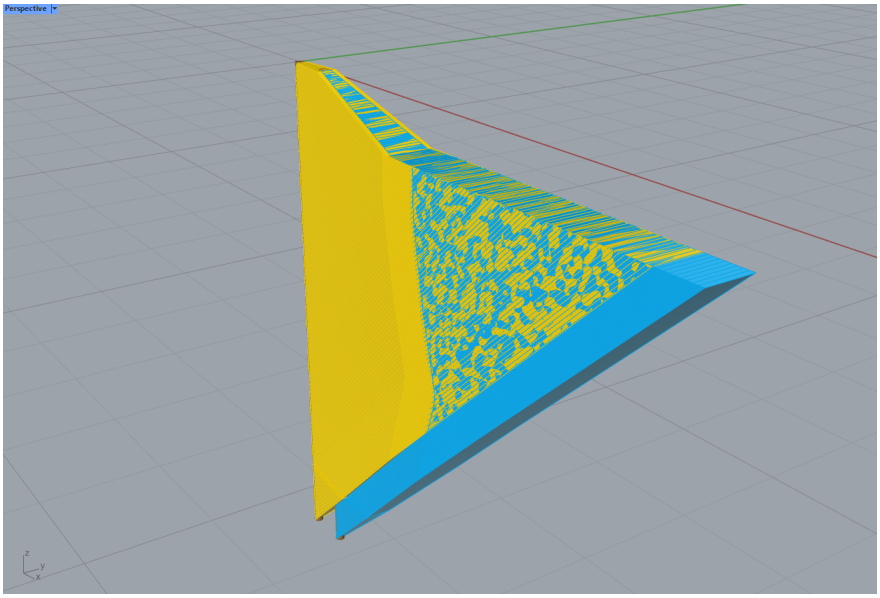


Figure D.9: Soil volume from Pile 1 and 2, reduced due to canal and pile group effects

The soil from both piles intersects, meaning that one more reduction will take place. In a way, the "mobilized" soil from P2 that intersects from P1 needs to be subtracted from P1. This follows from the theory mentioned in Hemel (2023).

At the moment, to carry out this reduction one can see that the new shape of the soil from P1 is irregular, see Figure D.10. This leads to a final general volume/shape of the pile row.

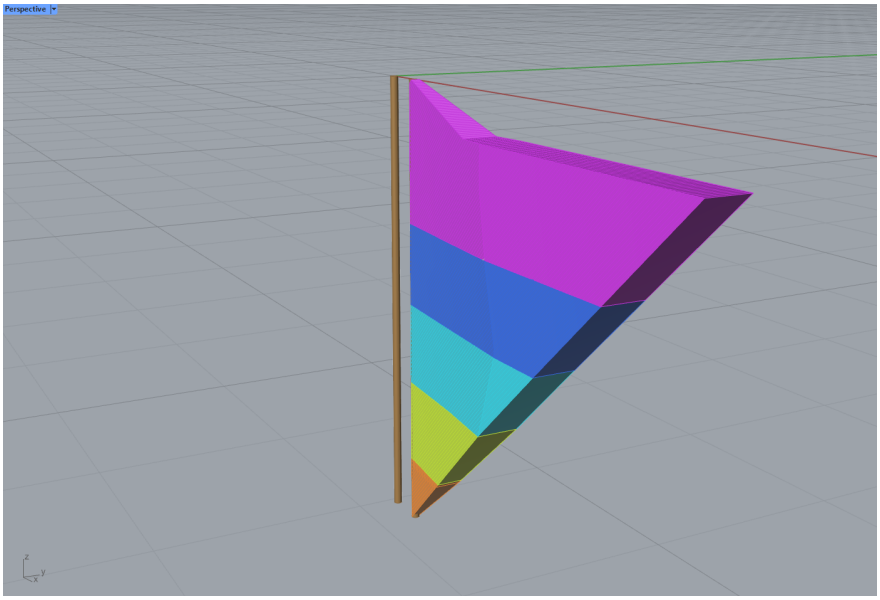


Figure D.10: Reduced volume from P2 with soil layers

E Verification of Abaqus model

In this appendix the verification of the modelling methods utilised in the FE-model is shown.

Verification of pile modelling

Verification of the beam element was conducted by modelling a 10 m beam with a rigid Boundary condition (BC) at the base of the pile and assigning a horizontal load at the pile top. 100 kN was the chosen load for this test case, named test case 1 for future references, and the beam element was given material properties according to Table E.1 and the test case can be seen in Figure E.1.

Table E.1: Material properties for pile in test case 1

Geometry	Young's modulus (E)	Horizontal load (H)	Mesh size
Circular pile, D=200 mm	11 GPa	100 kN	0.2 m

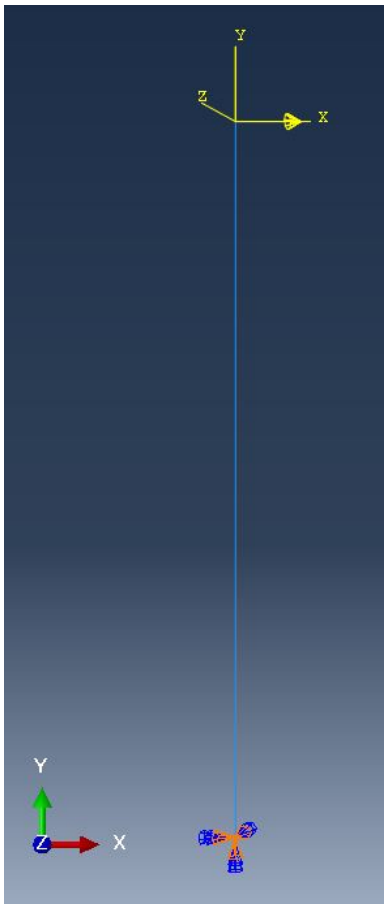


Figure E.1: Test case 1 - Geometry and loads

The result was compared to a hand-calculation, assuming a rigid cantilever beam with the same properties as the FE-model of test case 1. The resulting moment of inertia I, for the pile with geometry according to E.1, for a circular pile is calculated as $\frac{D^4 \times \pi}{64}$, where D is the pile diameter, resulting in $7.854 \times 10^{-5}\ m^4$.

The horizontal displacements for a rigid cantilever beam, loaded at the free edge can be calculated according to EQ E.1 leading to a displacement of $y_{max} = 38.58\ m$.

$$y_{max} = \frac{H \times L^3}{3 \times E \times I} \tag{E.1}$$

The resulting deflection of the pile seen in Figure E.2 where the resulting reaction force was obtained as $100\ kN$ and the displacement as $38.6\ m$.

The difference in the displacement at the top of the beam is neglectable whereas it's concluded that the beam element is correctly modelled by Abaqus.

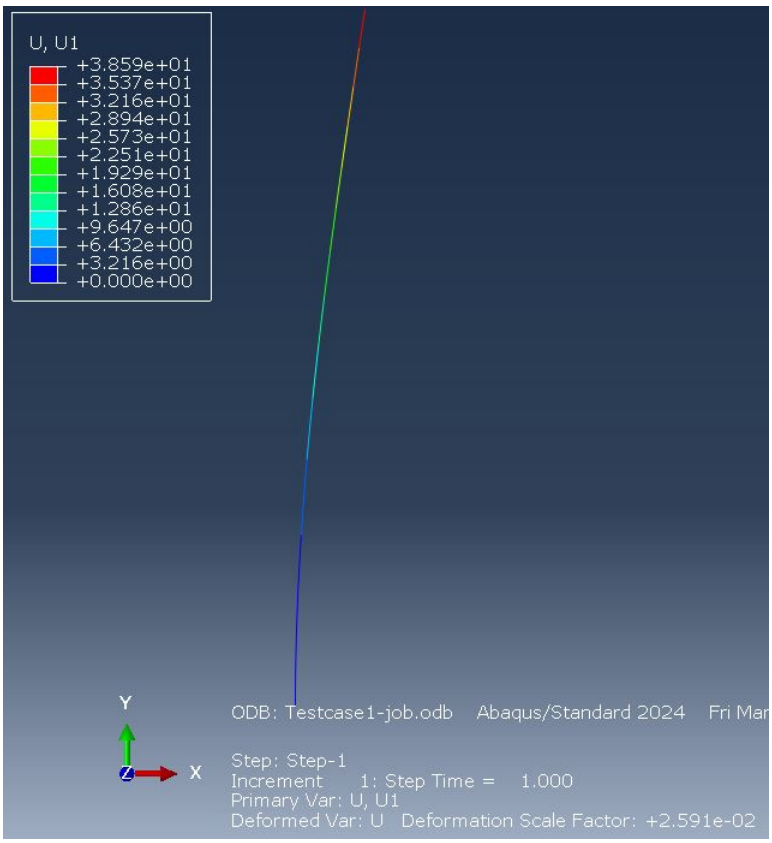


Figure E.2: Test case 1 - Horizontal displacements

The springs where verified by modelling a long simply supported beam loaded in the midpoint. A elasto-plastic spring was inserted below the applied vertical load. To prevent the supports at the beam edges from carrying part of the load before the plastic limit of the spring is reached the beam was modelled as 10 m long with a low elastic stiffness of 0.1 MPa and a rectangular $0.1 \times 0.1\ m^2$ section. The spring was given an elastic stiffness of 10 kN/m up to 1 m displacement and a plastic limit of 1.1 kN occuring at a displacement of 100 m.

A load case was studied (named 2 for future referencing) where the load is resisted by the spring up to 1 kN of applied load. Once the load is increased above 1 kN up until 10 kN the plastic limit is reached, forcing the remaining load to be carried by the supports. The load increase was done by incrementing the load 1 kN at a time until the whole load is applied to

the beam.

The parameters applied in the FE-model for test case 2 can be seen Table E.2 and the setup with the geometry and load application can be seen in Figure E.3 where the spring is inserted in the same position as the load.

Table E.2: Verification case for springs

Test case	Spring stiffness / plastic limit	Load	Young's modulus
2	10 kN/m / 1.1 kN	10 kN	1 MPa

Figure E.3: Test case 2, Geometry and load application

To validate if the results from the test cases are reasonable a comparison with a hand-calculation of a simply supported beam was conducted without any spring supports. This should lead to a higher beam displacement compared to test case 2. The moment of inertia of the rectangular section is calculated as $\frac{b \times h^3}{12}$ leading to $8.333\ m^{-5}$. The displacement in the mid-point is calculated with EQ E.2 leading to a displacement of $25000\ m$.

$$\frac{P \times l^3}{48 \times E \times I} \tag{E.2}$$

To check if the force is absorbed by the spring in test case 2 an initial model was created where the plastic limit is not included as a spring property.
The resulting displacement of the beam can be seen in Figure E.4. The beam displacement is 1 m in the mid-point in the vertical direction corresponding to the spring-displacement at 10 kN. It seems the full load is absorbed by the spring which is concluded from looking at Figure E.5 where the support reactions are almost 0.

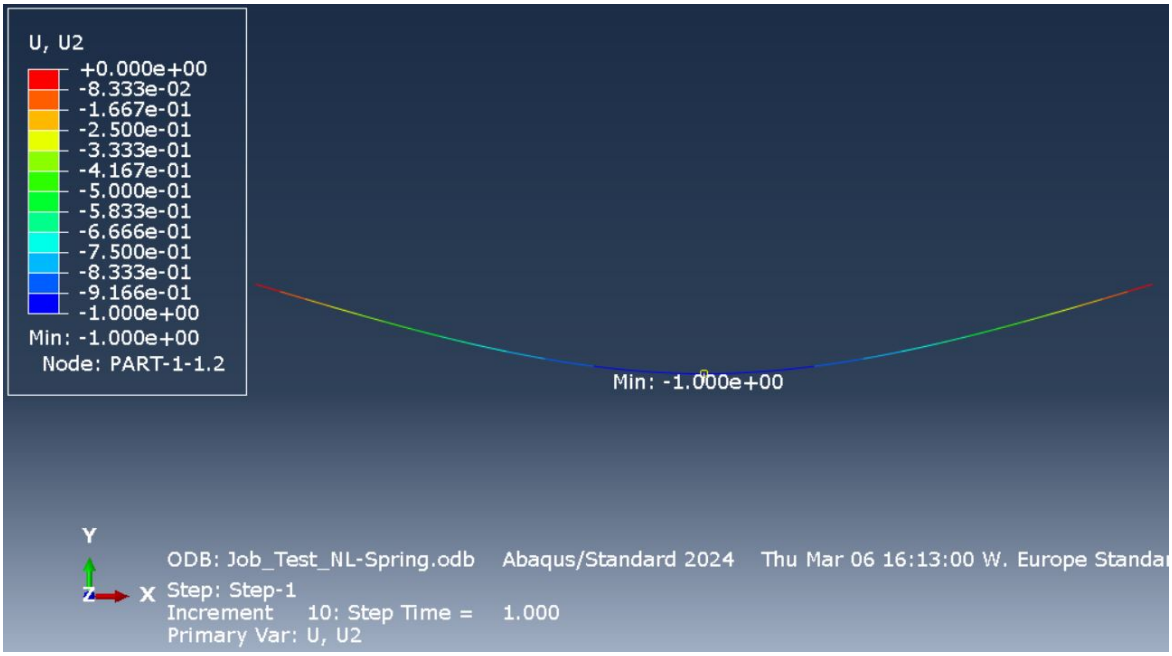


Figure E.4: Displacement with linear spring at 10 kN

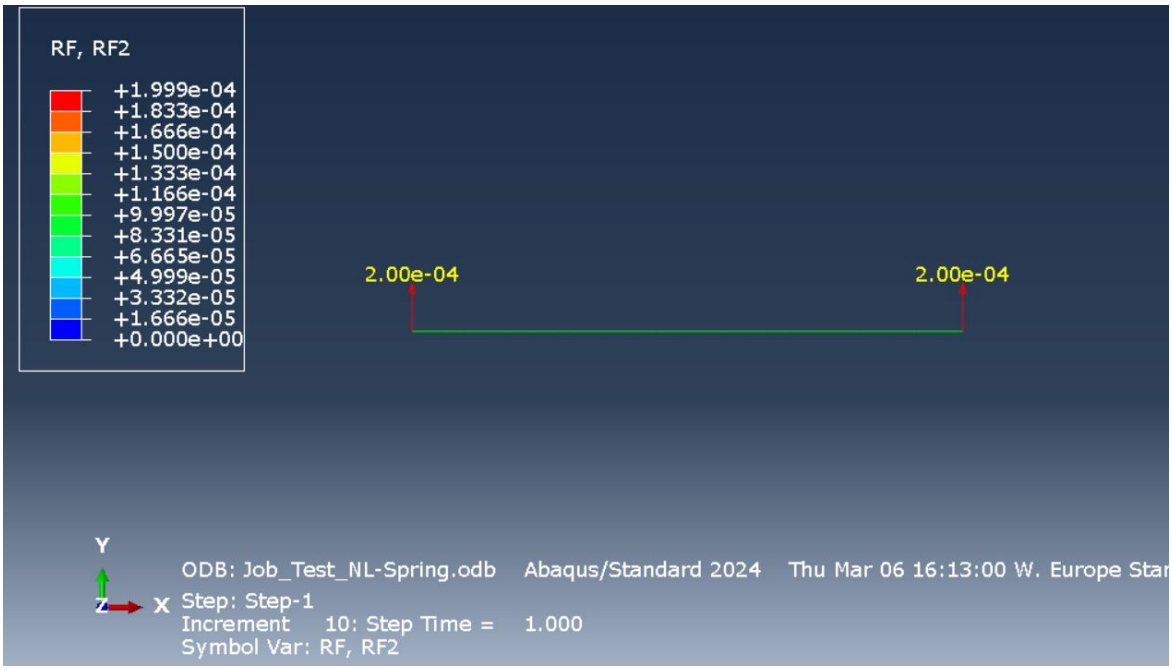


Figure E.5: Support reactions with linear spring at 10 kN

The displacement and support reactions when applying the full load of 10 kN are shown in Figure E.6 and Figure E.7 respectively. From there it's clear that the displacement of 22260 m is lower than the results from the hand-calculation.

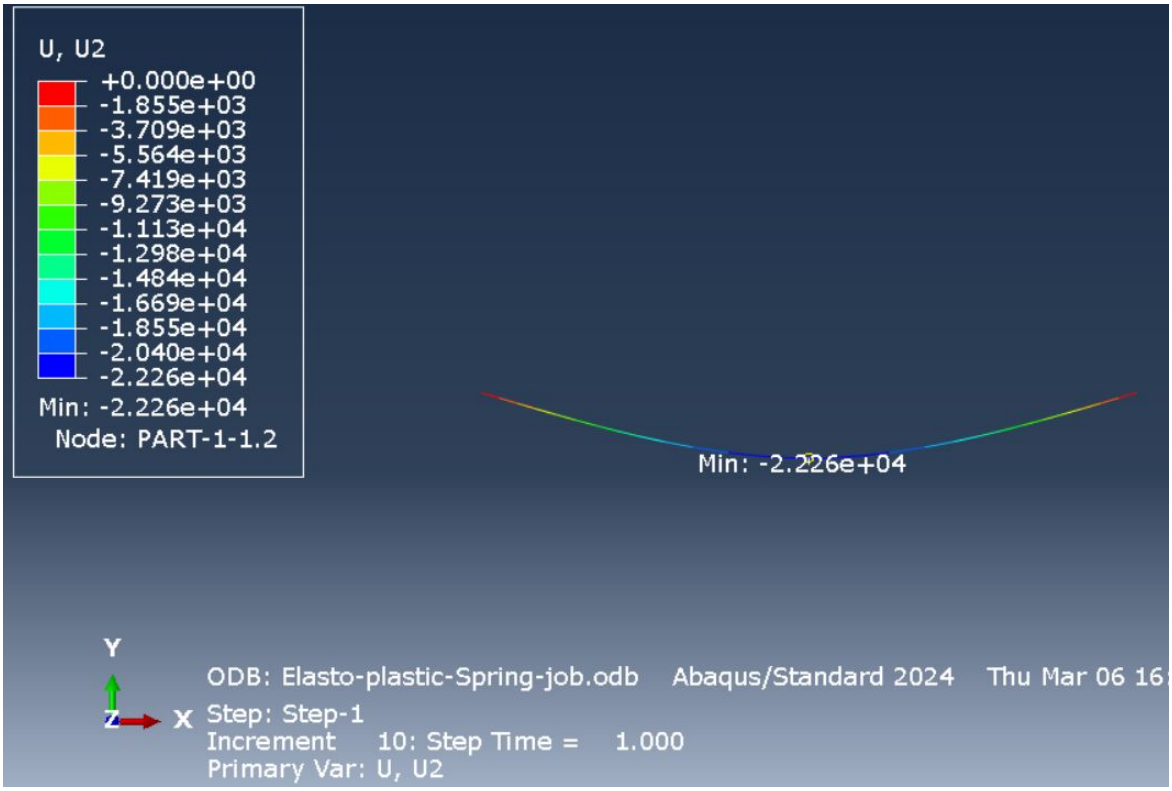


Figure E.6: Displacement with NL-springs at 10 kN

Evaluating the support reactions according to Figure E.7 it's clear that 9 kN out of the applied 10 kN is absorbed by the supports where the rest is resisted with the spring. We therefore conclude that the plastic limit is properly added to the spring. Noteable is that the displacement is roughly 90% of the total displacement of the hand-calculated value as $22260 \div 25000 = 89\%$.

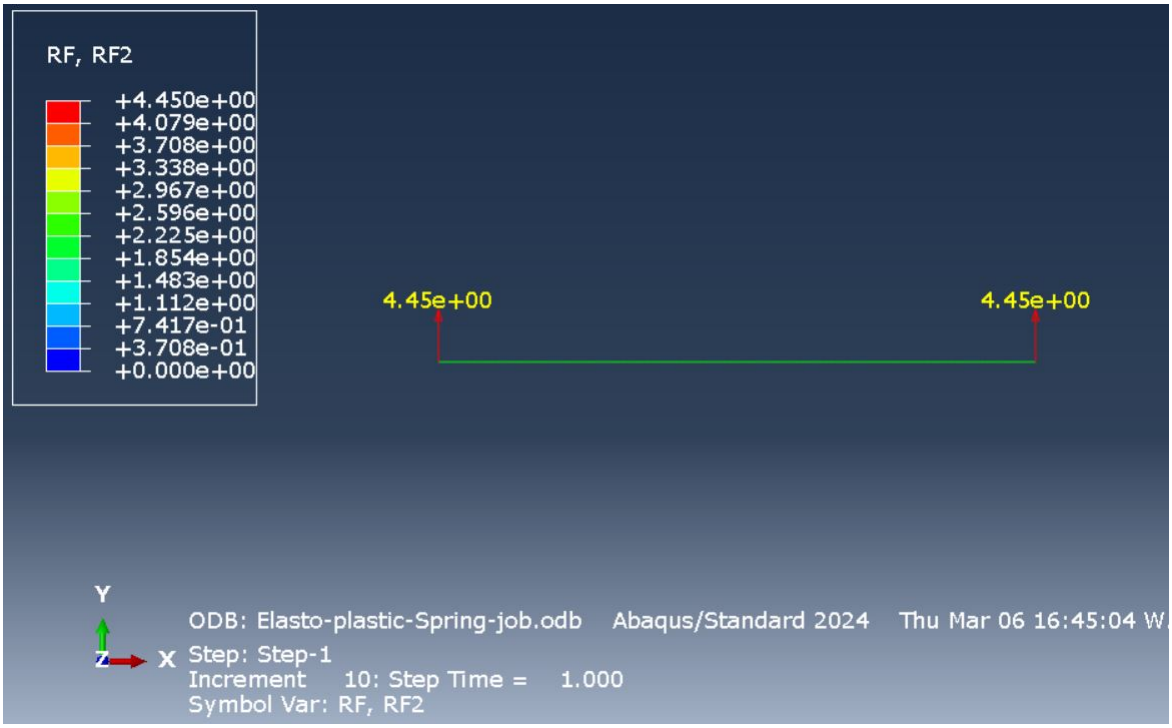


Figure E.7: Support reactions with NL-springs at 10 kN

When applying 1 kN of load to the beam (the 0.1 step out of 1) a displacement of 0.1 m is obtained according to Figure E.8 matching the linear spring properties before the plastic limit of 1 kN is reached.

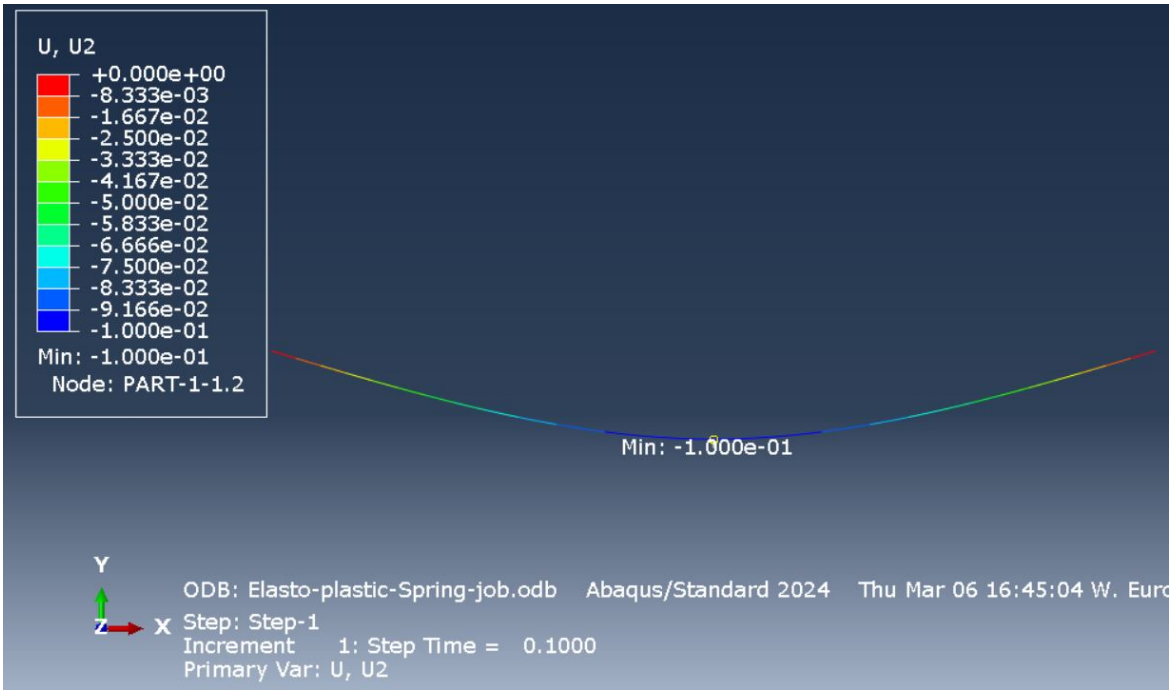


Figure E.8: Displacement with NL-springs at 1 kN

Further increasing the load to 2 kN yields a large increase of displacements up to roughly 2250 m which conclude that the remaining load is distributed to the supports after reaching

the plastic limit of the spring. This can be seen in Figure E.9 and the support reactions in Figure E.10 where the support reaction in kN becomes $RF2 = (2 - 1.1)/2 = 0.45$.

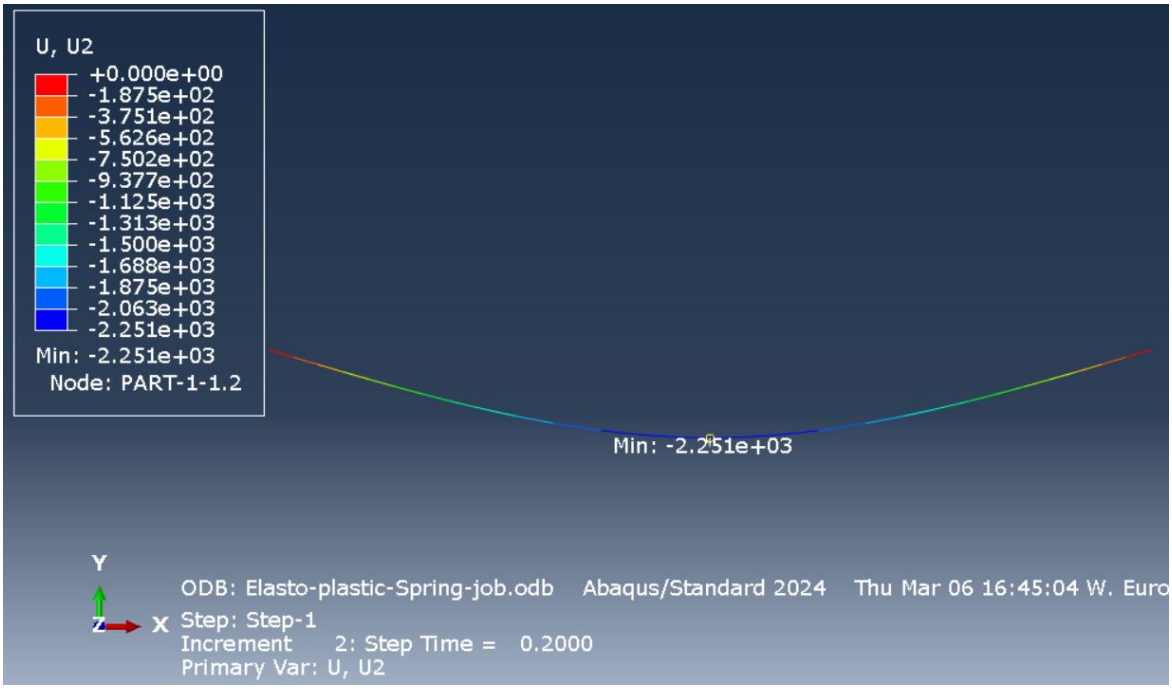


Figure E.9: Displacement with Non-linear spring at 2 kN

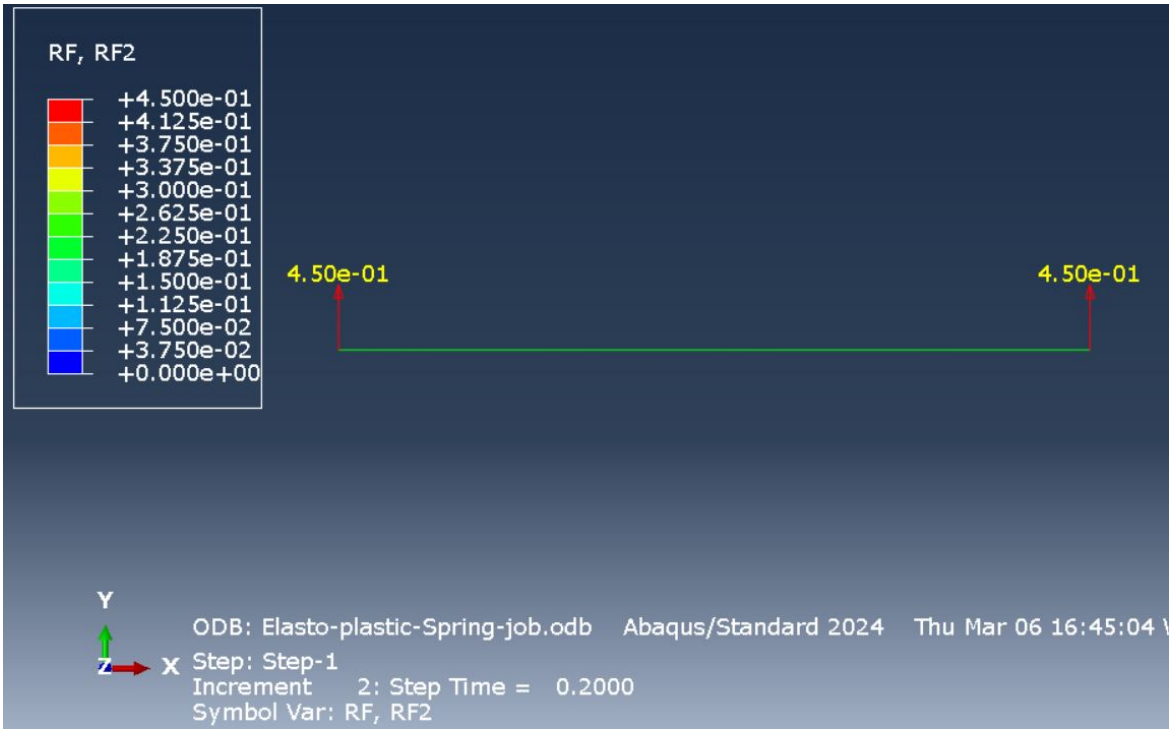


Figure E.10: Support reaction with NL-springs at 2 kN

Verification of timber deck modelling

To verify the method a simple beam with rigid end supports was modelled and a hinge was inserted between the supports. The total length of the beam was chosen as $4\ m$ and the hinge was inserted 1 m from the left support. The beam was loaded with a uniformly distributed load of $1\ kN/m$. The loads and boundary conditions can be seen in Figure E.11 and the properties of the hinge in Figure E.12.

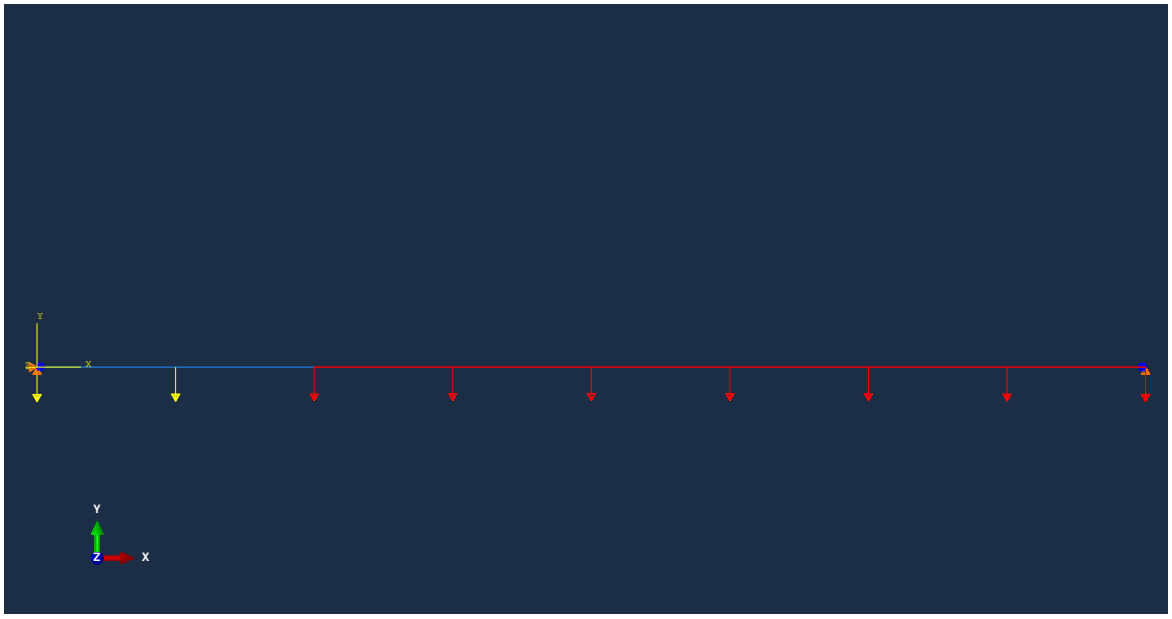


Figure E.11: Test 3 - Modelled loads and BC's

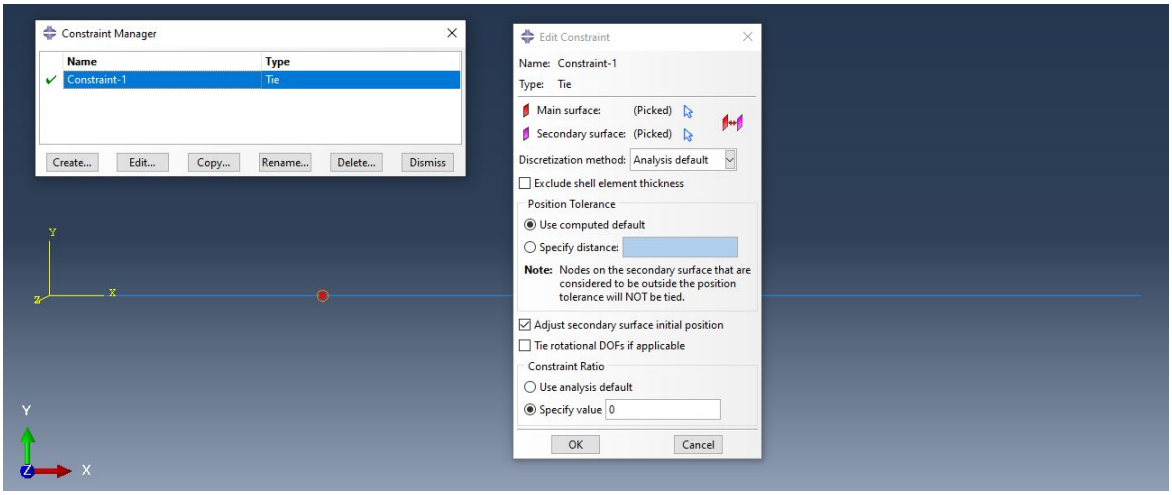


Figure E.12: Test 3 - Hinge properties

Looking at the results we conclude that the vertical forces are properly resisted by the supports, see Figure E.13. The resulting bending moment along the beam can be seen in Figure E.14.

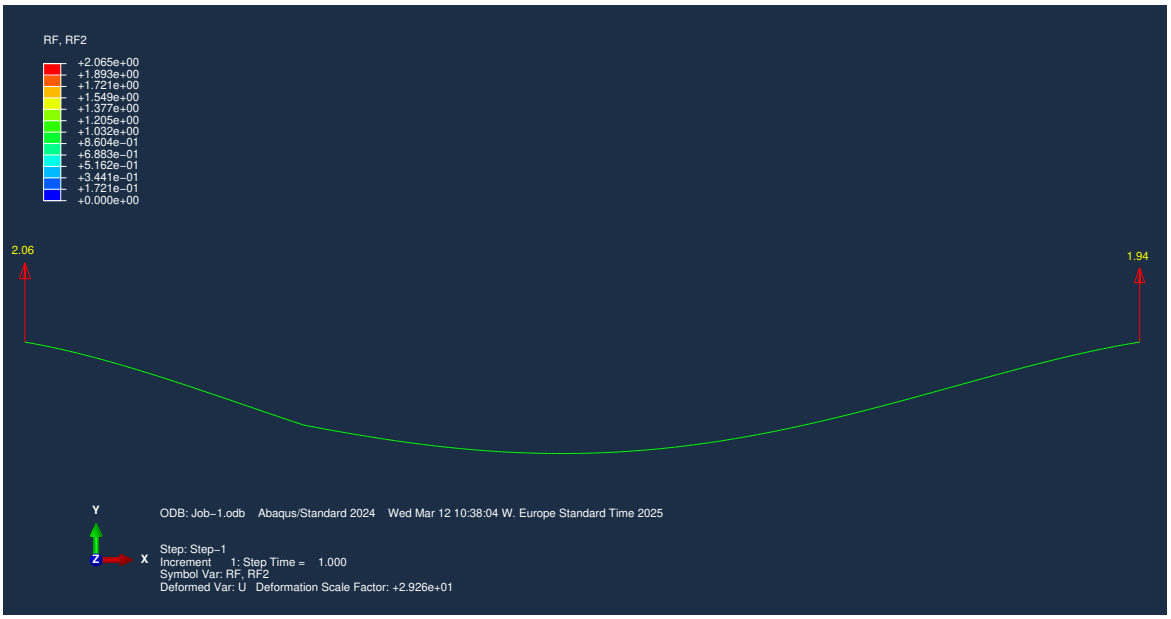


Figure E.13: Test 3 - Support reactions

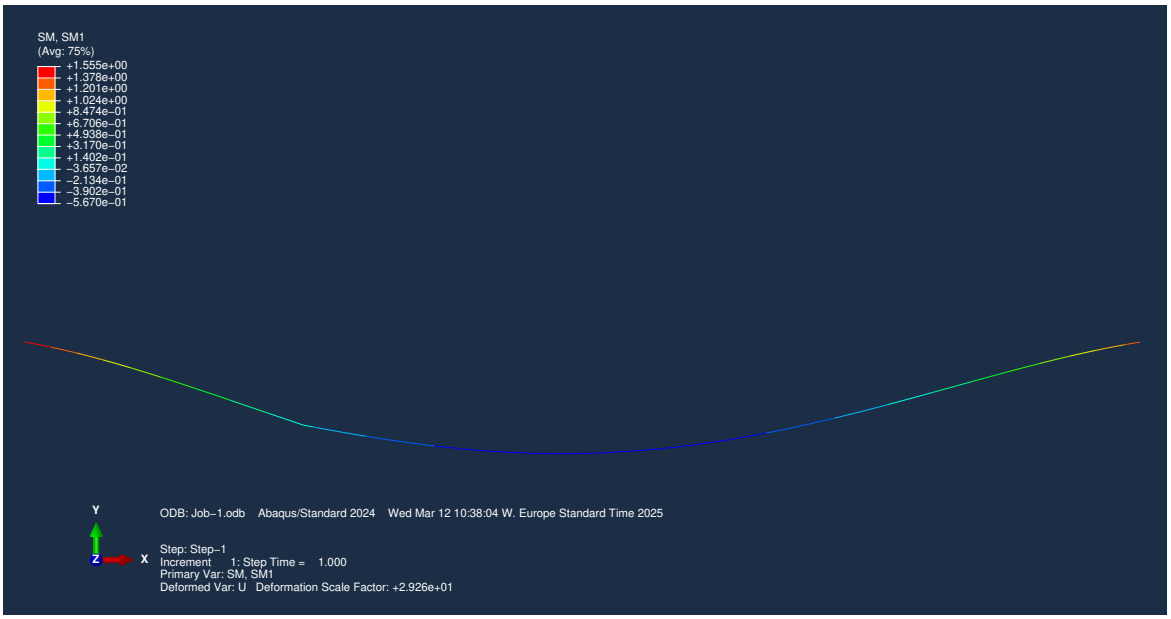


Figure E.14: Test 3 - Bending moment, beam with hinge

To verify that the 0 moment section occurs at the position of the hinge the moment distribution along the beam was plotted. The path chosen for the plot can be seen in Figure E.15 and the moment distribution in Figure E.16.

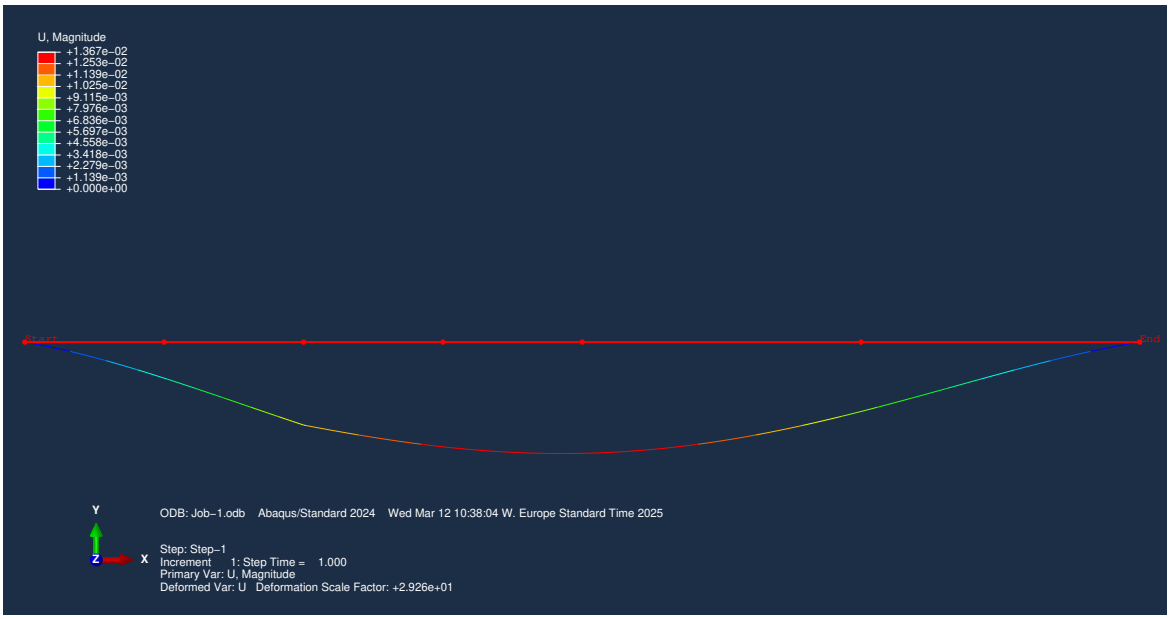


Figure E.15: Path created to plot bending moment

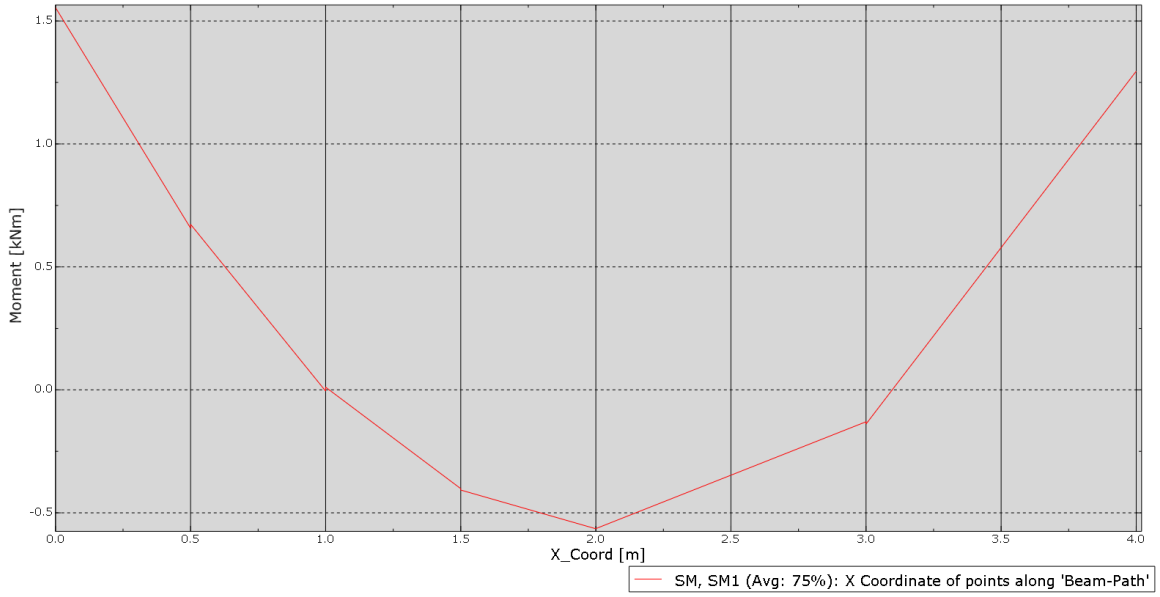


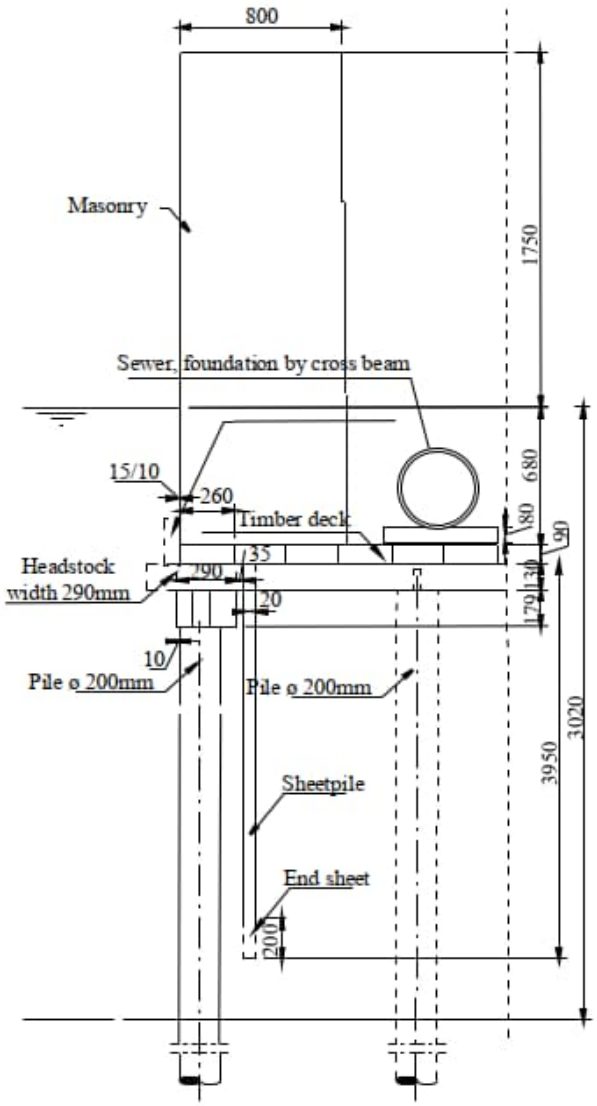
Figure E.16: Moment along beam.

From the graph we conclude that the 0 moment section occurs at the hinge position and the vertical force is distributed through the hinge. The chosen method to model hinges in Abaqus behaves as intended.

F Case study 1 - Input parameters

The input parameters used in the FE-models for Case study 1 can be seen below:

GRIMBURGWAL INPUT-DATA TO FE-MODEL



$E := 7000\ \boldsymbol{MPa} = \left(7 \cdot 10^{6}\right)\ \boldsymbol{kPa}$

(According to Hemel (2023)

$D_{min} := 0.2\ \boldsymbol{m}$ - Minimum pile diameter

$D_{max} := 0.25\ \boldsymbol{m}$ - Maximum pile diameter

$I_{min} := \frac{D_{min}^{4} \cdot \pi}{64} = \left(7.854 \cdot 10^{3}\right)\ \boldsymbol{cm}^{4}$

$I_{max} := \frac{D_{max}^{4} \cdot \pi}{64} = \left(1.917 \cdot 10^{4}\right)\ \boldsymbol{cm}^{4}$

$W_{min} := \frac{D_{min}^{3} \cdot \pi}{32} = 785.398\ \boldsymbol{cm}^{3}$

$W_{max} := \frac{D_{max}^{3} \cdot \pi}{32} = \left(1.534 \cdot 10^{3}\right)\ \boldsymbol{cm}^{3}$

$E \cdot I_{min} = 549.779\ \boldsymbol{kN \cdot m}^{2}$

$E \cdot I_{max} = 1342\ \boldsymbol{kN \cdot m}^{2}$

$t_{sheet} := 60\ \boldsymbol{mm}$

$L_{sheet} := 3.95\ \boldsymbol{m}$ - Length of sheet-pile

$L_{pile} := 11\ \boldsymbol{m}$ - Length of pile

Interface between pile cap and timber deck / beams assumed as hinged.

$h_{wall} := 1.75\ \boldsymbol{m} + 0.68\ \boldsymbol{m} = 2.43\ \boldsymbol{m}$ - Wall height

$\gamma_{masonry} := 25\ \frac{\boldsymbol{kN}}{\boldsymbol{m}^{3}}$ - Density of masonry wall

$B_{wall} := 0.8\ \boldsymbol{m}$ - Base width of wall

$e_{pile} := \frac{D_{min}}{2} = 0.1\ \boldsymbol{m}$ - Distance from pile mid to edge of masonry wall

$e_{sheetp.p} := 0.26\ \boldsymbol{m} - \frac{D_{min}}{2} + 0.035\ \boldsymbol{m} + \frac{t_{sheet}}{2} = 0.225\ \boldsymbol{m}$ - Distance c pile to c sheet pile

$N_{pile.top} := h_{wall} + 0.09\ \boldsymbol{m} + 0.13\ \boldsymbol{m} = 2.65\ \boldsymbol{m}$ - Top level of pile

$b_{head} := 0.29\ m$ $h_{head} := 0.13\ m$ - Geometry of headstocks

$I_{head} := \frac{b_{head} \cdot h_{head}^{3}}{12} = \left(5.309 \cdot 10^{-5}\right)\ m^4$ - Moment of inertia, headstock

$b_{deck} := 1\ m$ $h_{deck} := 0.09\ m$ - Geometry of timber deck

$I_{deck} := \frac{b_{deck} \cdot h_{deck}^{3}}{12} = \left(6.075 \cdot 10^{-5}\right)\ m^4$ - Moment of inertia, small pile

$I_{beam} := I_{head} + I_{deck} = \left(1.138 \cdot 10^{-4}\right)\ m^4$ - Moment of inertia, beam+deck

Resulting height of timber deck to obtain same bending stiffness as if stiffness of beam and deck was summed up:

$h_{eq} := \sqrt[3]{\frac{I_{beam}}{b_{deck}} \cdot 12} = 0.111\ m$ - Equivalent height of deck in Abaqus

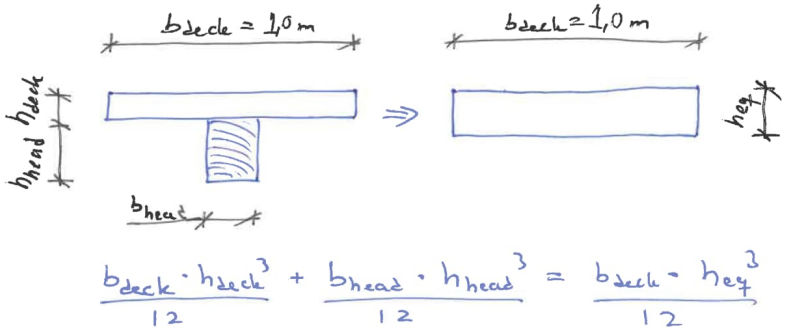


Figure: Principle for determining eqiuvalent height of deck to obtain same stiffness as headstock + deck

Comparison if deck and headstock had been a homogenuos section:

$A_{deck} := b_{deck} \cdot h_{deck} = 0.09\ m^2$ $y_{deck} := \frac{h_{deck}}{2} = 0.045\ m$

$A_{head} := b_{head} \cdot h_{head} = 0.038\ m^2$ $y_{head} := \frac{h_{head}}{2} + y_{deck} = 0.11\ m$

$y_{gc} := \frac{A_{deck} \cdot y_{deck} + A_{head} \cdot y_{head}}{A_{deck} + A_{head}} = 0.064\ m$

$y_{gc.deck} := y_{deck} - y_{gc} = -0.019\ m$

$y_{gc.head} := y_{head} - y_{gc} = 0.046\ m$

$I_{deck\_head} := I_{deck} + I_{head} + A_{deck} \cdot y_{gc.deck}^{2} + A_{head} \cdot y_{gc.head}^{2} = \left(2.261 \cdot 10^{-4}\right)\ m^4$

$\frac{I_{deck\_head}}{I_{beam}} = 1.986$ - Ratio between moment of inertia, homogenous / summed

Soil stresses

Vertical at pile deck

$\gamma_{sand.1.8} := 18\ \frac{kN}{m^3}$ $h_{1.8} := 1.8\ m$ - First sand layer

$\sigma_{v'.lay1} := \gamma_{sand.1.8} \cdot h_{1.8} = 32.4\ kPa$ - Vertical stress, first layer

$\gamma_{sand.2.4} := 9\ \frac{kN}{m^3}$ $h_{2.4} := 2.4\ m - h_{1.8} = 0.6\ m$ - Second sand layer

$\sigma_{v'.lay2} := \gamma_{sand.1.8} \cdot h_{1.8} + \gamma_{sand.2.4} \cdot h_{2.4} = 37.8\ kPa$ - Vertical stress, bot sand layer

$\gamma_{peat} := 1\ \frac{kN}{m^3}$ $h_{peat.wall} := h_{wall} - 2.4\ m = 0.03\ m$ - Density / height of peat-layer above timber deck

$\sigma_{v'.deck} := \sigma_{v'.lay2} + \gamma_{peat} \cdot h_{peat.wall} = 37.83\ kPa$ - Vertical stress on timber deck

$K_{A.wall} := 0.25$ - Active soil pressure coefficient on wall

$\varphi := 32.5\ deg$ - Friction angle sand (Material against wall)

$K_{0.SE} := 1 - \sin(\varphi) = 0.463$ - Comparison, K0 - Rankine theory

$K_{A.SE} := \left(\tan\left(45\ deg - \frac{\varphi}{2}\right)\right)^2 = 0.301$ - Comparison, KA - Rankine theory

$K_{A.SP} := 0.59$ - Active soil pressure coefficient on sheet pile wall

$\varphi_p := 14.9\ deg$ - Friction angle peat

$K_{0.SP.SE} := 1 - \sin(\varphi_p) = 0.743$ - Comparison, K0 - Rankine theory

$K_{A.SP.SE} := \left(\tan\left(45\ deg - \frac{\varphi_p}{2}\right)\right)^2 = 0.591$ - Comparison, KA - Rankine theory

Input to Abaqus:

$\sigma_{v'.deck} = 37.83\ kPa$ - Imported vertical stress acting on the timber deck in the Abaqus-model

Wall forces

Horizontal earth pressure:

$\sigma_{H.lay1} := \sigma_{v'.lay1} \cdot K_{A.wall} = 8.1\ kPa$

$\sigma_{H.deck} := \sigma_{v'.deck} \cdot K_{A.wall} = 9.458\ kPa$

Height of soil:

$h_{1.8} = 1.8\ m$

$h_{2.4} = 0.6\ m$

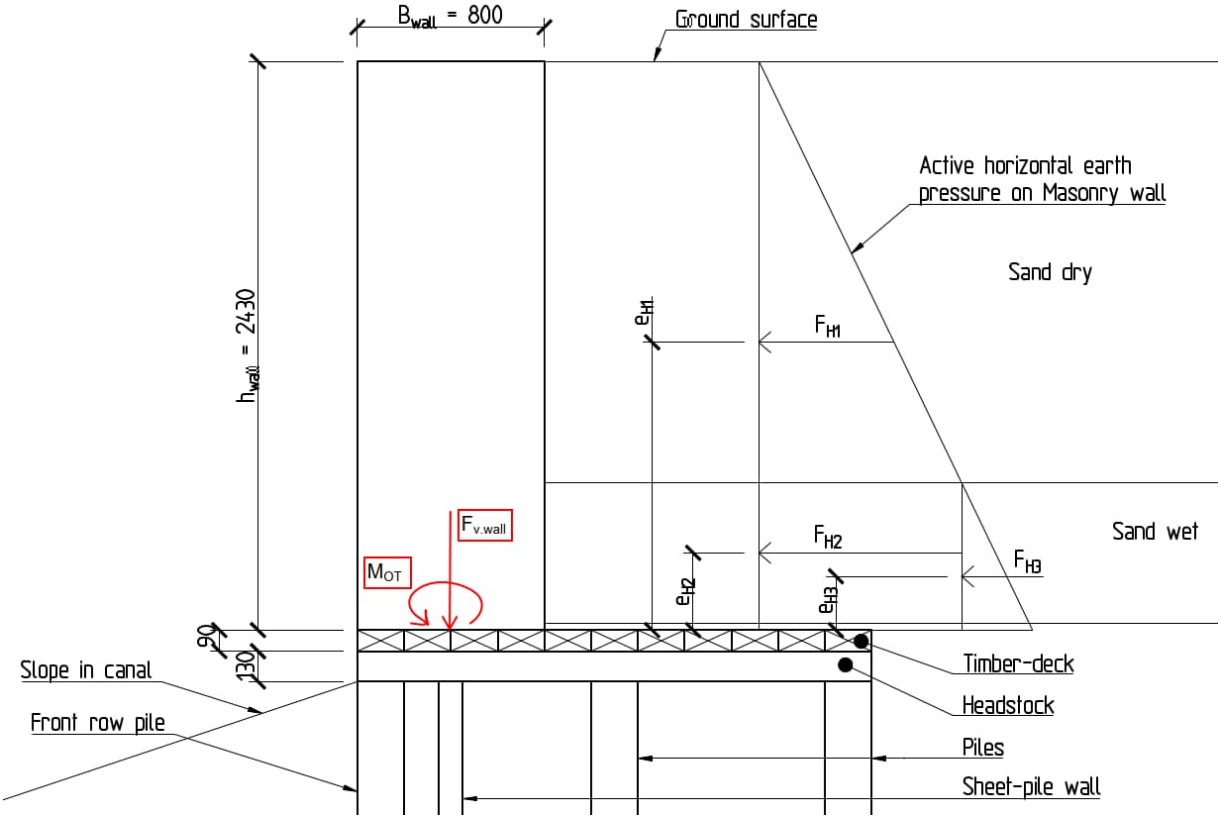


Figure: Forces acting on the masonry wall

Force components:

$F_{H1} := \sigma_{H.lay1} \cdot \frac{h_{1.8}}{2} = 7.29\ \frac{kN}{m}$

$F_{H2} := \sigma_{H.lay1} \cdot h_{2.4} = 4.86\ \frac{kN}{m}$

$F_{H3} := \left(\sigma_{H.deck} - \sigma_{H.lay1}\right) \cdot \frac{h_{2.4}}{2} = 0.407\ \frac{kN}{m}$

Lever arm of forces:

$e_{H1} := h_{2.4} + \frac{h_{1.8}}{3} = 1.2\ m$

$e_{H2} := \frac{h_{2.4}}{2} = 0.3\ m$

$e_{H3} := \frac{h_{2.4}}{3} = 0.2\ m$

Overturning moment:

$M_{OT} := F_{H1} \cdot e_{H1} + F_{H2} \cdot e_{H2} + F_{H3} \cdot e_{H3} = 10.287\ \frac{kN \cdot m}{m}$

Vertical force from wall:

$B_{wall} = 0.8\ m$ $h_{wall} = 2.43\ m$ $\gamma_{masonry} = 25\ \frac{kN}{m^3}$

$F_{v.wall} := B_{wall} \cdot h_{wall} \cdot \gamma_{masonry} = 48.6\ \frac{kN}{m}$

$e_{Fv} := \frac{M_{OT}}{F_{v.wall}} = 0.212\ m$ $B_{eff} := B_{wall} - 2 \cdot e_{Fv} = 0.377\ m$

Determination of stresses beneath Masonry wall:

if $e_{Fv} < \frac{B_{wall}}{6}$
  return "No lifting"
else if $\frac{B_{wall}}{6} < e_{Fv} < \frac{B_{wall}}{3}$
  return "Lifting, calculate B_effective"
else
  return "Unstable"

$=$ "Lifting, calculate B_effective"

$\frac{F_{v.wall}}{B_{wall}} + \frac{6 \cdot M_{OT}}{B_{wall}^2} = 157.195\ kPa$ - Maximum stress when e < B/6

$\frac{4 \cdot F_{v.wall}}{3} \cdot \frac{1}{\left(B_{wall} - 2 \cdot e_{Fv}\right)} = 172.044\ kPa$ - Maximum stress when B/6 < e < B/3

$$\sigma_{max} := \begin{cases} \left\| \dfrac{F_{v.wall}}{B_{wall}} + \dfrac{6 \cdot M_{OT}}{B_{wall}^{\,2}} \right. & \text{if } e_{Fv} < \dfrac{B_{wall}}{6} \\ \left\| \dfrac{4 \cdot F_{v.wall}}{3} \cdot \dfrac{1}{\left(B_{wall} - 2 \cdot e_{Fv}\right)} \right. & \text{else if } \dfrac{B_{wall}}{6} < e_{Fv} < \dfrac{B_{wall}}{3} \end{cases} = 172.044\ \textbf{kPa}$$ - Maximum stress, front of wall

$$\sigma_{min} := \begin{cases} \left\| \dfrac{F_{v.wall}}{B_{wall}} - \dfrac{6 \cdot M_{OT}}{B_{wall}^{\,2}} \right. & \text{if } e_{Fv} < \dfrac{B_{wall}}{6} \\ \left\| 0\ \textbf{kPa} \right. & \text{else if } \dfrac{B_{wall}}{6} < e_{Fv} < \dfrac{B_{wall}}{3} \end{cases} = 0\ \textbf{kPa}$$ - Minimum stress, back of wall

Load input beneath masonry wall in Abaqus:

$B_{eff} = 0.377\ \textbf{m}$ - Distribution width of load

$\sigma_{max} = 172.044\ \textbf{kPa}$ - Maximum pressure at wall front

$\sigma_{min} = 0\ \textbf{Pa}$ - Minimum pressure at back of wall

Resulting horizontal force from wall

$F_H := F_{H1} + F_{H2} + F_{H3} = 12.557\ \dfrac{\textbf{kN}}{\textbf{m}}$ - Horizontal force on wall

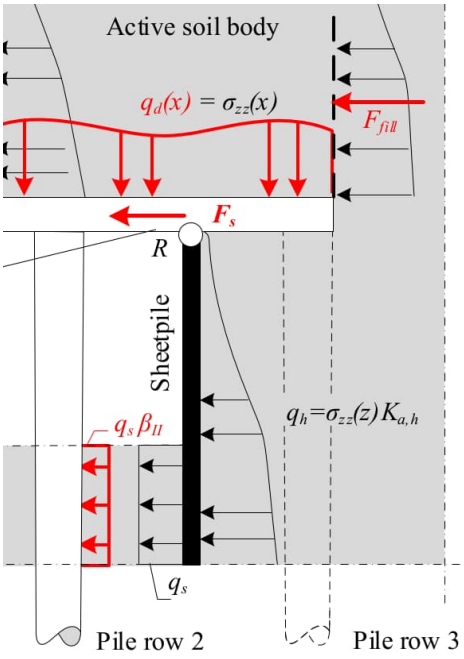
$q_{h.wall} := \dfrac{F_H}{B_{eff}} = 33.339\ \dfrac{1}{\textbf{m}} \cdot \dfrac{\textbf{kN}}{\textbf{m}}$ - Horizontal line-load from wall

Self weight timber:

$w_t := 370\ \dfrac{\textbf{kg}}{\textbf{m}^3}$ $\quad g = 9.807\ \dfrac{\textbf{m}}{\textbf{s}^2}$

$g_t := w_t \cdot g = 3.628\ \dfrac{\textbf{kN}}{\textbf{m}^3}$

Horizontal stress on sheet pile wall



Depth [m]	Sort [-]	γ' [kN/m^3]	q_c [kPa]	c [kN/m^2]	φ [deg]
0 – 1.8	Sand dry	18.0	-	0.0	32.5
1.8 – 2.4	Sand wet	9.0	-	0.0	32.5
2.4 – 6.0	Peat	1.0	200	6.2	14.9
6.0 – 8.0	Sea clay	7.2	200	3.7	28.8
8.0 – 10.0	Wadsand	7.9	5,000	2.2	25.0
10.0 - 12.0	Clay	7.2	1,000	3.7	23.6
12.0 – 20.0	Sand	10.0	10,000	0.1	35

$h_{1.8} = 1.8\ m$ - Thickness of dry sand layer

$h_{2.4} = 0.6\ m$ - Thickness of wet sand layer

$h_6 := (6 - 2.4)\ m = 3.6\ m$ - Thickness of peat layer

$L_{sp} := 3.95\ m$ - Length sheet pile wall

$N_{pile.top} = 2.65\ m$ - Level top of pile

Figure: Representation of loads against sheet-pile wall according to Hemel (2023)

$h_{wall} = 2.43\ m$ - Depth at bottom of wall

$t_{deck} := 0.09\ m$ - Thickness of timber deck

The length of 3.95 m is measured to the top of the head-stock.

$N_{uk.sp} := h_{wall} + t_{deck} + L_{sp} = 6.47\ m$ - Depth, bottom of sheet pile wall

Assume all forces below deck is taken by the sheet pile wall

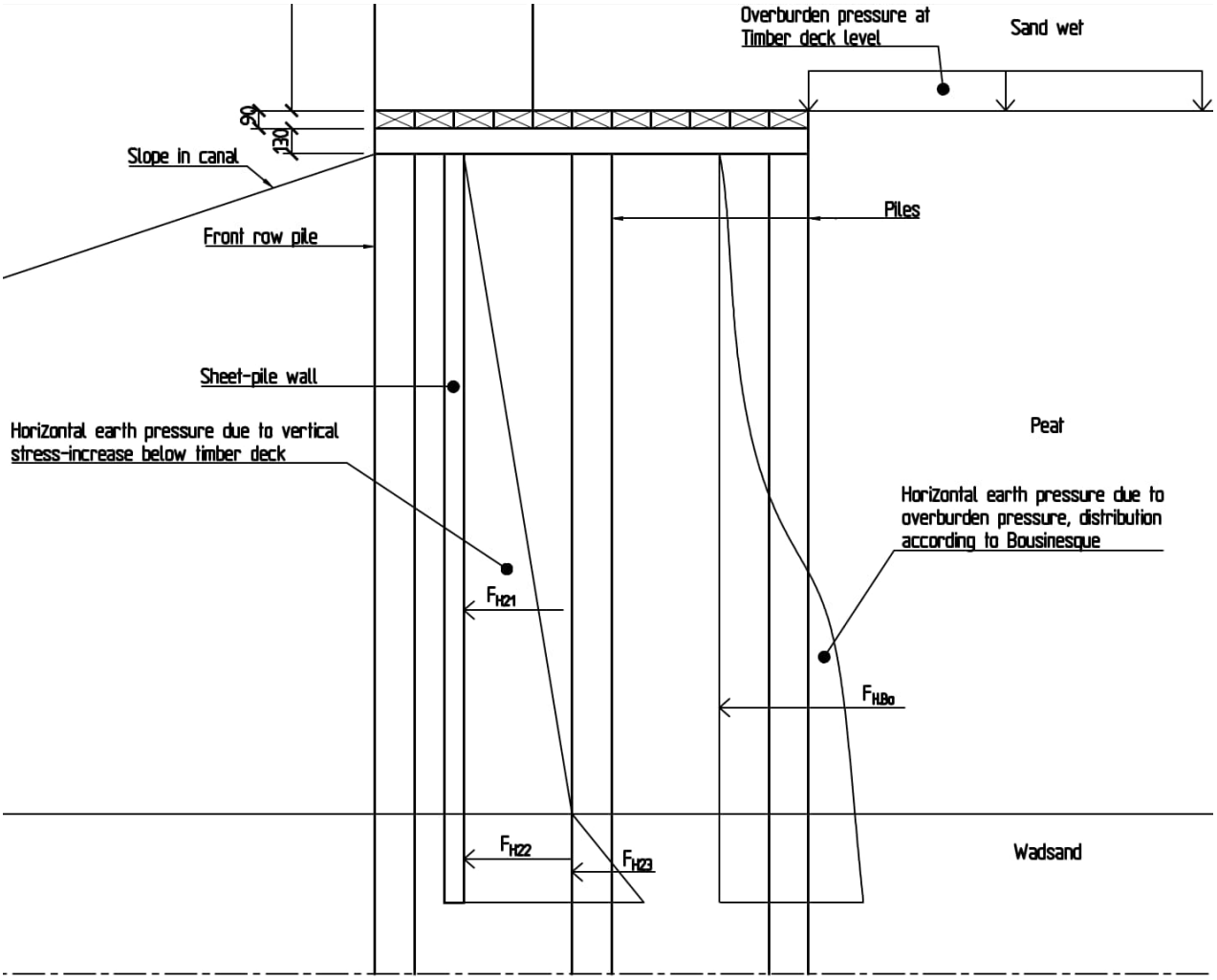


Figure: Forces acting on the sheet-pile wall

Calculation of forces acting on the sheet-pile wall / front-row pile:

$\sigma_{v'.deck} = 37.83\ \boldsymbol{kPa}$ - Vertical stress at top of timber deck

$t_{deck} = 0.09\ \boldsymbol{m}$ - Thickness of timber deck

$\sigma_{v'.sp.top} := \sigma_{v'.deck} + \gamma_{peat} \cdot t_{deck} = 37.92\ \boldsymbol{kPa}$ - Surcharge load at top of timber deck

$\gamma_{peat} = 1\ \frac{\boldsymbol{kN}}{\boldsymbol{m}^3}$ $h_6 = 3.6\ \boldsymbol{m}$ - Density / thickness of peat-layer

$\sigma_{v.6} := \sigma_{v'.lay2} + \gamma_{peat} \cdot h_6 = 41.4\ \boldsymbol{kPa}$ - Vertical stress at bottom of peat layer

$\gamma_{cl} := 7.2\ \frac{\boldsymbol{kN}}{\boldsymbol{m}^3}$ $h_{cl} := N_{uk.sp} - 6\ \boldsymbol{m} = 0.47\ \boldsymbol{m}$ - Densiy of clay / thickness of clay layer until bottom of sheet-pile wall

$\sigma_{v'.sp.bot} := \sigma_{v.6} + \gamma_{cl} \cdot h_{cl} = 44.784\ \boldsymbol{kPa}$ - Effective stress at bottom of sheet pile wall

$L_{bot.sp} := \frac{L_{sp}}{3} = 1.317\ m$ - Length, bottom part of sheet-pile wall

$N_{qs.top} := N_{uk.sp} - L_{bot.sp} = 5.153\ m$ - Level at top of q_h distributed load

$K_{A.SP} = 0.59$ - Active earth-pressure coefficient

$\sigma_{H.sp.top} := K_{A.SP} \cdot \sigma_{v'.sp.top} = 22.373\ kPa$

$\sigma_{H.sp.bot} := K_{A.SP} \cdot \sigma_{v'.sp.bot} = 26.423\ kPa$

- Horizontal stress at top / bottom of sheet-pile wall

Resulting force components:

$F_{H.4} := \sigma_{H.sp.top} \cdot L_{sp} = 88.373\ \frac{kN}{m}$ $e_4 := \frac{L_{sp}}{2} = 1.975\ m$

$F_{H.5} := \left(\sigma_{H.sp.bot} - \sigma_{H.sp.top}\right) \cdot \frac{L_{sp}}{2} = 7.998\ \frac{kN}{m}$ $e_5 := L_{sp} \cdot \frac{2}{3} = 2.633\ m$

$e_{qs} := \frac{5}{6} \cdot L_{sp} = 3.292\ m$ - Excentricity to resultant from uniformely pressure q_s from top of sheet-pile wall

$F_{qs} := \frac{\left(F_{H.4} \cdot e_4 + F_{H.5} \cdot e_5\right)}{e_{qs}} = 59.422\ \frac{kN}{m}$ - Force resultant of q_s

<u>Resulting forces on sheet pile when simplifying stress field</u>

$q_s := \frac{F_{qs}}{L_{bot.sp}} = 45.131\ kPa$ - Horizontal lineload, bottom sheet pile

$F_s := F_{H.4} + F_{H.5} - F_{qs} = 36.949\ \frac{kN}{m}$ - Reaction force, top of sheet pile, transfered to front pile row.

Load input to Abaqus, simplified stress field:

$F_H = 12.557 \ \frac{kN}{m}$ - Horizontal force on masonry wall

$q_{h.wall} = 33.339 \ kPa$ - Horizontal stress under masonry wall

$q_s = 45.131 \ kPa$ - Horizontal stress bottom part of sheet pile wall

$L_{bot.sp} = 1.317 \ m$ - Bottom-length of sheet pile resisting earth pressure

$N_{qs.top} - N_{pile.top} = 2.503 \ m$

$N_{uk.sp} - N_{pile.top} = 3.82 \ m$

- Depth interval of sheet pile with horizontal earth pressure

$F_s = 36.949 \ \frac{kN}{m}$ - Horizontal force top of sheet pile wall

- The horizontal stress under the masonry wall is added on the partitioned part beneath the masonry wall

- The horizontal stress on bottom part of sheet pile wall is assigned to the front pile beam-element affected by the load

- The horizontal force on top of sheet pile wall is added to the top of the front pile row

Horizontal stress on sheet pile - Bousinesque stress field

Vertical effective stress field when the depth "counting" starts below the timber deck

$\sigma_{v'.sp.top} = 37.92\ \boldsymbol{kPa}$ - Surcharge load at top of sheet-pile

$\gamma_{peat} = 1\ \frac{\boldsymbol{kN}}{\boldsymbol{m}^3}$ $h_6 = 3.6\ \boldsymbol{m}$ - Density / thickness of peat layer

$\sigma_{v.peat.bot} := \gamma_{peat} \cdot h_6 = 3.6\ \boldsymbol{kPa}$ - Stress at bottom of peat layer

$\gamma_{cl} = 7.2\ \frac{\boldsymbol{kN}}{\boldsymbol{m}^3}$ $h_{cl} = 0.47\ \boldsymbol{m}$ - Density / thickness of clay layer

$\sigma_{v'.sp.bot2} := \sigma_{v.peat.bot} + \gamma_{cl} \cdot h_{cl} = 6.984\ \boldsymbol{kPa}$ - Vertical stress at bottom of sheet-pile wall

$L_{bot.sp} = 1.317\ \boldsymbol{m}$ - Length of bottom third of sheet-pile wall

$N_{qs.top} = 5.153\ \boldsymbol{m}$ - Level at top of distributed force q_s

$K_{A.SP} = 0.59$ - Active earth-pressure coefficent

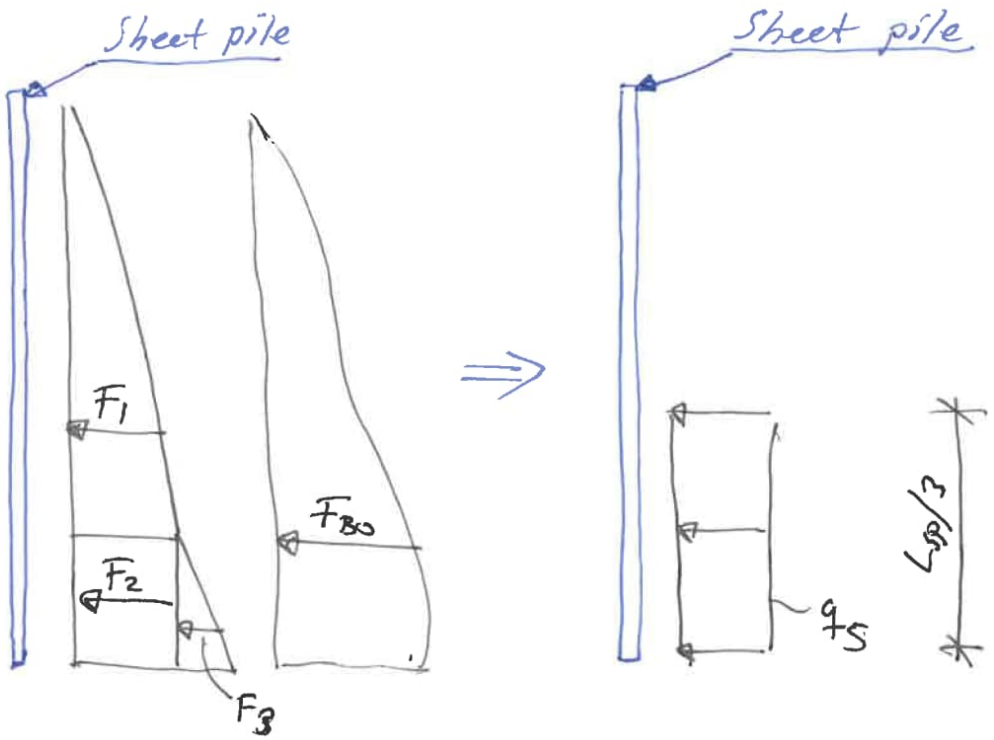


Figure: Principle for determining distributed earth pressure at bottom of sheet-pile wall from horizontal force components

Stress field according to Bousinesque

Equation for Bousinesque stress field:

$$\beta(r,z) := \operatorname{atan}\left(\frac{r}{z}\right) \qquad \alpha(B,r,z) := \operatorname{atan}\left(\frac{B+r}{z}\right) - \beta(r,z)$$

$$\sigma_V(r,z,Q,B) := \frac{Q}{\pi} \cdot \left(\alpha(B,r,z) + \sin\left(\alpha(B,r,z)\right) \cdot \cos\left(\alpha(B,r,z) + 2 \cdot \beta(r,z)\right)\right)$$

$r := 0.8\ m$ - Distance from end of timber deck to sheet-pile wall

$L_{sp} = 3.95\ m$ - Length of sheet-pile wall

$B := 20\ m$ - Assumed width of stress field from overburden pressure

$z_{sp} := 0.1\ m, 0.2\ m .. L_{sp}$ - Depth interval beneath timber deck

CHANGING RADIUS GIVES STRESS-FIELD FOR 2-4 PILE ROWS, 0.8 m = 2 rows, 1.8 m = 3 rows, 2.8 m = 4 rows

$$\sigma_V\left(r, z_{sp}, \sigma_{v'.sp.top}, B\right) = \begin{bmatrix} 0.015 \\ 0.117 \\ 0.362 \\ \vdots \end{bmatrix} kPa$$

- Bousinesque vertical stress field

$$\int_{0\ m}^{L_{sp}} \sigma_V\left(r, z_{sp}, \sigma_{v'.sp.top}, B\right) \mathrm{d}z_{sp} = 34.084\ \frac{kN}{m}$$

- Integrated vertical force component

Finding position of horizontal force resultant from the Bousinesque stress field:

Guess Values

$$x := \frac{L_{sp}}{2}$$

Constraints

$$\frac{1}{2} \cdot \int_{0\ m}^{L_{sp}} \sigma_V\left(r, z_{sp}, \sigma_{v'.sp.top}, B\right) \mathrm{d}z_{sp} = \int_{0\ m}^{x} \sigma_V\left(r, z_{sp}, \sigma_{v'.sp.top}, B\right) \mathrm{d}z_{sp}$$

Solver

$$e_{Bo} := \mathbf{find}(x) = 2.663\ m$$

$e_{Bo} = 2.663\ m$ - Distance to force resultant of Bousinesque stress field from top of sheet-pile wall

Resulting force and lever arm to the Bousinesque stress distribution at sheet-pile wall:

$$F_{H.Bo} := K_{A.SP} \cdot \int_{0\ m}^{L_{sp}} \sigma_V\left(r, z_{sp}, \sigma_{v'.sp.top}, B\right) \mathrm{d}z_{sp} = 20.11\ \frac{kN}{m} \qquad e_{Bo} = 2.663\ m$$

$\sigma_{H.sp.top2} := 0\ kPa$ - Horizontal stress, top sheet pile wall

$\sigma_{H.b.peat} := K_{A.SP} \cdot \sigma_{v.peat.bot} = 2.124\ kPa$ - Horizontal stress, bottom of peat layer

$\sigma_{H.sp.bot2} := K_{A.SP} \cdot \sigma_{v'.sp.bot2} = 4.121\ kPa$ - Horizontal stress, bottom sheet pile wall

$$F_{H.21} := \sigma_{H.b.peat} \cdot \frac{h_6}{2} = 3.823\ \frac{kN}{m} \qquad e_{21} := h_6 \cdot \frac{2}{3} = 2.4\ m$$

$$F_{H.22} := \sigma_{H.b.peat} \cdot h_{cl} = 0.998\ \frac{kN}{m} \qquad e_{22} := L_{sp} - \frac{h_{cl}}{2} = 3.715\ m$$

$$F_{H.23} := \left(\sigma_{H.sp.bot2} - \sigma_{H.b.peat}\right) \cdot \frac{h_{cl}}{2} = 0.469\ \frac{kN}{m} \qquad e_{23} := L_{sp} - \frac{h_{cl}}{3} = 3.793\ m$$

$e_{qs} = 3.292\ m$ - Excentricity to resultant from uniformely pressure q_h from top of sheet-pile wall

$F_{A.tot.bo} := F_{H.21} + F_{H.22} + F_{H.23} + F_{H.Bo} = 25.4\ \frac{kN}{m}$ Horizontal force against sheet-pile wall

- Horizontal force resultant against sheet-pile wall from the distributed force q_h:

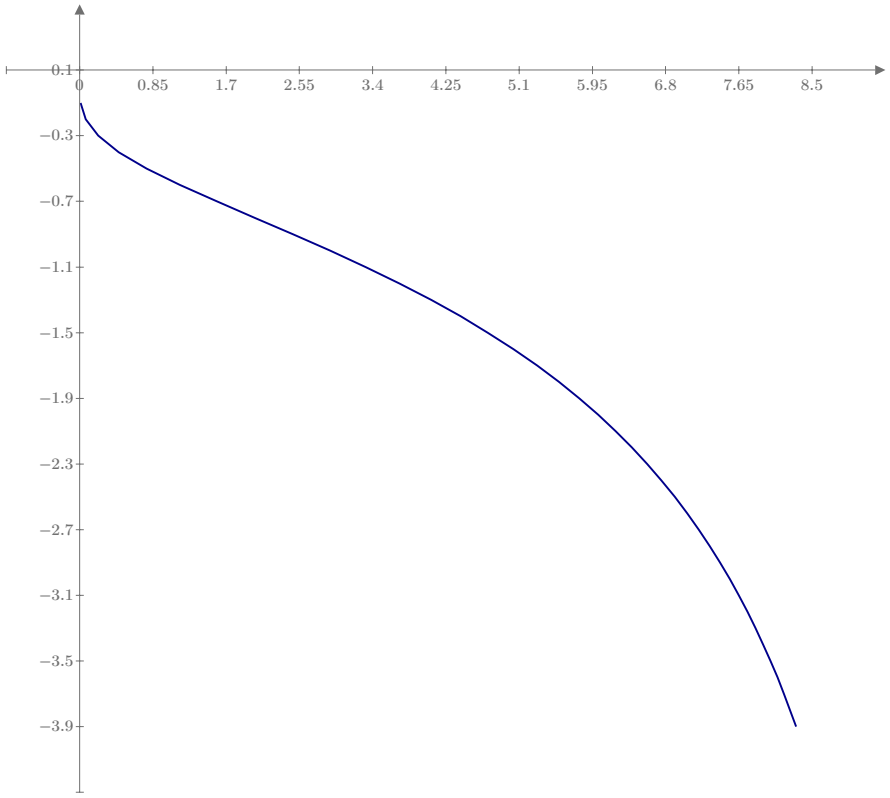
$$F_{qs.Bo} := \frac{\left(F_{H.21} \cdot e_{21} + F_{H.22} \cdot e_{22} + F_{H.23} \cdot e_{23} + F_{H.Bo} \cdot e_{Bo}\right)}{e_{qs}} = 20.725\ \frac{kN}{m}$$

Resulting forces on sheet pile when utilizing Bousinesque stress field

$q_{s.Bo} := \frac{F_{qs.Bo}}{L_{bot.sp}} = 15.741\ kPa$ - Horizontal lineload, bottom sheet pile

$F_{s.Bo} := F_{A.tot.bo} - F_{qs.Bo} = 4.675\ \frac{kN}{m}$ - Reaction force, top of sheet pile, transfered to front pile row.

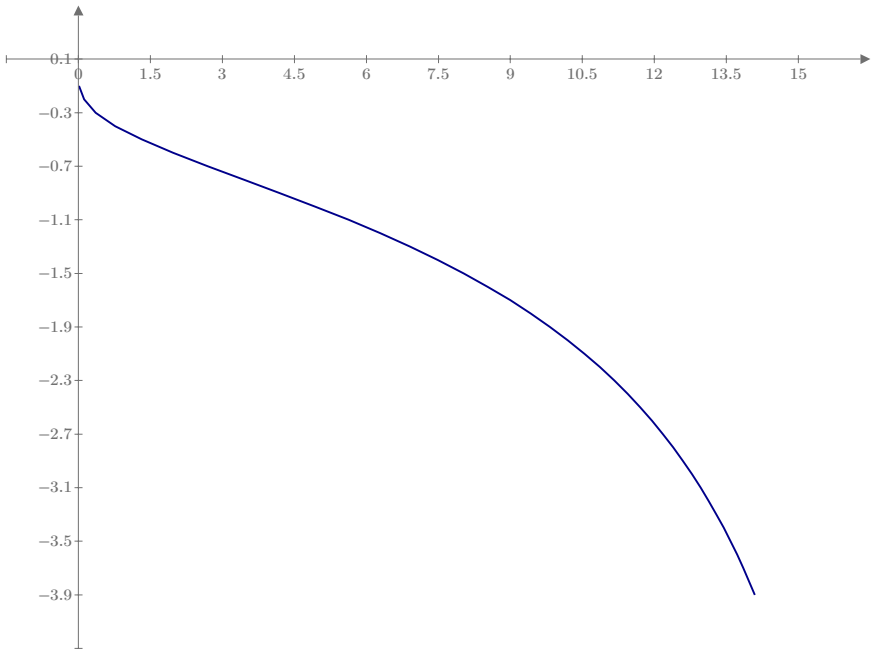
Bousinsque stress field against sheet-pile wall



$-z_{sp}\ \ (m)$

$K_{A.SP} \cdot \sigma_V\left(r, z_{sp}, \sigma_{v'.sp.top}, B\right)\ \ (kPa)$

Plot: Horizontal stress distribution due to overburden pressure at timber-deck level



$-z_{sp}\ \ (m)$

$\sigma_V\left(r, z_{sp}, \sigma_{v'.sp.top}, B\right)\ \ (kPa)$

Plot: Vertical stress distribution due to overburden pressure at timber-deck level

Geometrical input to assign springs in Abaqus

$$Depth := \begin{bmatrix} 1.8 \\ 2.4 \\ 6 \\ 8 \\ 10 \\ 12 \\ 20 \end{bmatrix} m \qquad \Delta D_{piles} := 2.6\ m \qquad N_{pile.top} = 2.65\ m$$

- Depth to different soil layers according to Hemel 2023, table 6.2

$$Depth\_piles := Depth - \Delta D_{piles} = \begin{bmatrix} -0.8 \\ -0.2 \\ 3.4 \\ 5.4 \\ 7.4 \\ 9.4 \\ 17.4 \end{bmatrix} m$$

- Equivalent depth of soil layers counting top of pile as the 0 base

Data for verification of results in Abaqus

Horizontal forces, not accounting for spring reaction-forces

$F_{qs.Bo} = 20.725 \ \frac{kN}{m}$ - Force resultant on Sheet-pile wall bottom part

$F_{s.Bo} = 4.675 \ \frac{kN}{m}$ - Horizontal force at top of sheet-pile wall

$F_H = 12.557 \ \frac{kN}{m}$ - Horizontal force from soil on wall

$F_{H.tot} := F_{qs.Bo} + F_{s.Bo} + F_H = 37.958 \ \frac{kN}{m}$ - Total applied horizontal force in system

Vertical forces

$\sigma_{v'.deck} = 37.83 \ kPa$ $B_{deck} := 1 \ m$ $D_{pile} := D_{min} = 0.2 \ m$

$F_{v.\sigma v} := \sigma_{v'.deck} \cdot B_{deck} = 37.83 \ \frac{kN}{m}$

$F_{V.wall} := \frac{\sigma_{max} \cdot B_{eff}}{2} = 32.4 \ \frac{kN}{m}$

$A_{pile} := \frac{D_{pile}^{\ 2} \cdot \pi}{4} = 0.031 \ m^2$ $L_{pile} = 11 \ m$ $g_t = 3.628 \ \frac{kN}{m^3}$

$G_{pile} := \frac{L_{pile} \cdot A_{pile} \cdot g_t}{m} = 1.254 \ \frac{kN}{m}$

$G_{deck} := h_{eq} \cdot B_{deck} \cdot g_t = 0.403 \ \frac{kN}{m}$

$F_{V.tot} := 2 \cdot G_{pile} + G_{deck} + F_{V.wall} + F_{v.\sigma v} = 73.14 \ \frac{kN}{m}$ - Total applied vertical force in the system

G Case study 1 - Python code for Abaqus

The Python script used to run the Abaqus analysis in Case study 1 is shown below:

```
# -*- coding: mbcs -*-
# Do not delete the following import lines
from abaqus import *
from abaqusConstants import *
import __main__

import section
import regionToolset
import displayGroupMdbToolset as dgm
import part
import material
import assembly
import step
import interaction
import load
import mesh
import optimization
import job
import sketch
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import connectorBehavior
import numpy as np
import math as ma

import os
import sys
import csv

sys.path.append(r"c:\program files\python312\lib\site-packages")

cwd = os.getcwd()

sys.path.append(f"{cwd}")


### IMPORTING REDUCTION FACTORS FOR SPRINGS ###


#OBSERVE, MAKE SURE THE CORRECT Psi_factors.csv FILE IS USED!

    #Reading csv-file and storing reduction factors
with open('Psi_factors.csv', 'r' ) as csvfile:
    Psi_pile = np.genfromtxt(csvfile, dtype=float, delimiter=';')
with open('Psi_factors_dz05.csv', 'r' ) as csvfile:
    Psi_pile_05 = np.genfromtxt(csvfile, dtype=float, delimiter=';')


include_red = 1 #Set to 1 to include soil-wedge reduction
NL = OFF  #Set to ON or OFF to include or not include Non-linear geometry

```

```
### DEFINING ITTERATIONS TO SIMULATE ###


#Defining the amount of itterations of pile diameters
div_D = 1

min_D = 0.2 #Minimum diameter of piles to analyze
D_dif = 0.05 #Increment of diameter increase

#Defining the amount of itterations of ndz divisions
div_dz = 1

max_dz = 0.1
dz_dif = 0.1

# Defining amount of different number of pile rows
div_red = 3

#Setting amount of pile rows

num_rows = div_dz * div_D * div_red  #Set number of itterations

dz_it = np.zeros(num_rows)
D_it = np.zeros(num_rows)
red_it = np.zeros(num_rows)
name_job_it = []

#Assigning values for the itterative analysis
for i in range(div_D):

    D_inp = round(min_D + i*D_dif, 2)
    #D_name = f'D{i+1}'
    D_name = f'{round(D_inp*100)}' #Rounded to ensure sucessful naming of
     job

    for j in range(div_dz):

        dz_inp = round(max_dz - j*dz_dif, 2)
        dz_name = f'{round(dz_inp*100)}' #Rounded to ensure sucessful
    naming of job

        for k in range(div_red):

            row_base = i*div_dz*div_red + j*div_red
            row_nr = row_base + k

            red_name = 2+k

            dz_it[row_nr] = dz_inp
            D_it[row_nr] = D_inp
            red_it[row_nr] = k

            name_job_it.append(f'{D_name}_{dz_name}_nP{red_name}_red1_OP
    ') #Name of the current job

Loop = len(D_it) #Number of itterations in the analysis

```

```
107
108 ### LOOPING OVER ANALYSIS ITTERATIONS AND ASSIGNING INPUT PARAMETRES ###
109
110
111 for it in range(Loop):
112
113     #Units are imported as kN, m, kPa, kN/m^3
114
115     Mdb()
116
117     name_job = name_job_it[it]
118
119     #Pile geometry
120     #L = Abaqus_inp.L
121     L = 11
122
123     if red_it[it] == 0:
124         x_pile = np.array([0,1])
125     elif red_it[it] == 1:
126         x_pile = np.array([0,1,2])
127     elif red_it[it] == 2:
128         x_pile = np.array([0,1,2,3])
129
130     pile_top = 0
131     Depth_init = 2.6        #Initial depth to top of piles
132     dx = x_pile[1]-x_pile[0]
133
134     #Wall geometry
135     B_wall = 0.8
136     B_eff = 0.377
137
138     #Partitioning division of deck-width
139
140     if B_wall < dx:
141         u_partition_deck = B_wall/dx
142         print('Wall width less than pile spacing')
143     else:
144         u_partition_deck = 1
145         print('Wall width larger than pile spacing')
146
147     if B_eff < B_wall:
148         u_partition_load = B_eff / B_wall
149         print('Wall-edge liftup')
150     else:
151         u_partition_load = 1
152         print('No lifting of wall')
153
154
155     #OBSERVE! GENERALLY LOAD INPUT IS CALCULATED MANUALLY, CHECK MATHCAD
    "GRIMBURGWAL INPUT"
156
157
158     #Load from wall on Timber deck
159     sigma_wall_max = 172.044
160
161     #Load from vertical stress on Timber deck
162     sigma_v_soil = -37.83
```

```
    sigma_p = 1

    #Horizontal loading from wall
    q_h_wall = -33.339

    #Headstock moment
    Mrot = 1.9
    include_rot = 0 #Set to 1 to include rot moment in pile-top

    #Load on first pile row from sheet pile wall
    F_s_2 = -4.675  #2 Pile rows
    F_s_3 = -1.813  #3 Pile rows
    F_s_4 = -1.131  #4 Pile rows

    if red_it[it] == 0:
        F_s = F_s_2
    elif red_it[it] == 1:
        F_s = F_s_3
    elif red_it[it] == 2:
        F_s = F_s_4

    #Load from sheet pile wall to first pile row
    q_s_2 = -15.741 #2 Pile rows
    q_s_3 = -9.905  #3 Pile rows
    q_s_4 = -6.938  #4 Pile rows

    if red_it[it] == 0:
        q_s = q_s_2
    elif red_it[it] == 1:
        q_s = q_s_3
    elif red_it[it] == 2:
        q_s = q_s_4

    L_bot_sp = 1.317

    z1_sp = -2.503

    #BCs for pile base, SET = SUPPORTED, UNSET = UNSUPPORTED
    U1_PB = UNSET
    U2_PB = SET
    U3_PB = UNSET

    #Setting Mesh-properties

    #ndz = Abaqus_inp.ndz
    dz = dz_it[it]
    ndz = ma.floor(L/dz)
    n_div = 2   #Number of divisions in mesh in between nodes

    #Depths
    Depth_div = [-3.4,-5.4,-7.4,-9.4,-20]

    z_depth = np.zeros(ndz)

    for i in range(ndz):
        z_depth[i] = -(dz/2+i*dz)

```

```
    #z_depth contains the z position of each spring

    #Meshsize
    meshsize_pile = L/(n_div*ndz)


    #ADDING ANALYSIS STEP


    Step_name = 'Step-1'

    mdb.models['Model-1'].StaticStep(name=Step_name, previous='Initial', nlgeom=NL)
    mdb.models['Model-1'].steps[Step_name].setValues(initialInc=0.01, maxNumInc=1000, maxInc=0.05)


    #MATERIAL DEFINITIONS


    Material = 'Timber'
    Youngs_mod = 7000000
    Density = 3.628         #In [kN/m^3]
    nu = 0.2

    #Pile section

    #D = Abaqus_inp.D
    D = D_it[it]
    Radius = D/2
    Section = 'Pile'
    name_profile=f'Section r={Radius}'

    #Timber deck section
    b_deck = 1      #Width of timber deck
    h_deck = 0.111  #Height of timber deck

    Section_Deck = 'Timber_Deck'
    name_profile_deck = f'Section b={b_deck}_h={h_deck}'


### CREATING PILE PARTS ###


    mdb.models['Model-1'].Material(name=Material)
    mdb.models['Model-1'].materials[Material].Density(table=((Density, ), ))
    mdb.models['Model-1'].materials[Material].Elastic(table=((Youngs_mod, nu), ))

    print(f"Material {Material} created")

    mdb.models['Model-1'].CircularProfile(name=name_profile, r=Radius)
    mdb.models['Model-1'].BeamSection(name=Section, integration=DURING_ANALYSIS,
        poissonRatio=0.0, profile=name_profile, material=Material,
        temperatureVar=LINEAR, beamSectionOffset=(0.0, 0.0),
```

```
        consistentMassMatrix=False)

    print(f"Section {Section} created")

    mdb.models['Model-1'].RectangularProfile(name=name_profile_deck, a=
    b_deck, b=h_deck)
    mdb.models['Model-1'].BeamSection(name=Section_Deck,
        integration=DURING_ANALYSIS, poissonRatio=0.0, profile=
    name_profile_deck,
        material='Timber', temperatureVar=LINEAR, beamSectionOffset=(0.0,
     0.0),
        consistentMassMatrix=False)

    print(f"Section {Section} created")



### CALCULATION OF SPRING PROPERTIES FOR THE DIFFERENT DEPTHS ###


    #Calculation of plastic limit of springs

    "Data from Table 6.2 "

    Depth = np.array([
                        [0, 3.4],
                        [3.4, 5.4],
                        [5.4, 7.4],
                        [7.4, 9.4],
                        [9.4, L]
                       ]) #Depth ranges in [m]

    gamma = np.array([ [1],
                    [7.2],
                    [7.9],
                    [7.2],
                    [10]
                  ]) # gamma'values in [kN/m^3]

    '''
    q_c = np.array([ [200],
                     [200],
                     [5000],
                     [1000],
                     [10000]
                    ]) #cone resistance [kPa]
    '''

    q_c = [200,200,5000,1000,10000]

    c = np.array([ [6.2],
                    [3.7],
                    [2.2],
                    [3.7],
                    [0.1]
                  ]) #Cohesion [kN/m^2]

```

```
    fi = np.array([ [14.9],
                    [28.8],
                    [25],
                    [23.6],
                    [35]
                  ])

    def alpha_c(K0c, Kinfc, fi):
        "fi must be in degrees"
        alpha = ( K0c/(Kinfc-K0c) ) * 2 * np.sin( (np.pi/4) + (np.radians
    (fi)/2) )
        return  alpha

    def alpha_q(K0q, Kinfq, fi, K0):
        "fi must be in degrees"
        alpha = (K0q/(Kinfq-K0q)) * ( (K0 * np.sin(np.radians(fi))) / np.
    sin( (np.pi/4) + np.radians(fi)/2) )

        return alpha

    def K_0 (fi):
        "fi must be in degrees"
        K = 1 - np.sin(np.radians(fi))

        return K

    def N_c(fi):
        "fi must be in degrees"
        N = ( np.exp(np.pi * np.tan(np.radians(fi))) * np.tan( (np.pi/4 )
     + (np.radians(fi)/2) )**2 -1) * (1/np.tan(np.radians(fi)))

        return N

    def d_c_inf (fi):
        "fi must be in degrees"
        d = 1.58 + 4.09 * np.tan(np.radians(fi))**4

        return d

    def K_infc(Nc, d_c_inf):
        "fi must be in degrees"
        K = Nc * d_c_inf

        return K

    def K_infq (fi, Kinfc, K_0):
        "fi must be in degrees"

        K = Kinfc * K_0 * np.tan(np.radians(fi))

        return K

    def K_0c (fi):

        K = ( np.exp( ( (np.pi/2) + np.radians(fi) ) * np.tan(np.radians(
    fi)) ) * np.cos(np.radians(fi)) * np.tan( (np.pi/4)+(np.radians(fi/2)
    ) ) -1 ) * (1/np.tan(np.radians(fi)) )
```

```
        return K

    def K_0q(fi):

        K = ( np.exp( ( (np.pi/2) + np.radians(fi) ) * np.tan( np.radians
    (fi)) ) * np.cos( np.radians(fi) ) * np.tan( (np.pi/4) + (np.radians(
    fi)/2) ) ) - np.exp( (-(np.pi/2) + np.radians(fi) ) * np.tan( np.
    radians(fi) ) ) * np.cos( np.radians(fi) ) * np.tan( (np.pi/4) - (np.
    radians(fi)/2) )

        return K

    def K_c (K0c,Kinfc,alpha_c,z,D):

        ### Where  z = Depth [m], D = Pile Diameter [m]

        K = ( K0c + Kinfc * alpha_c * (z/D) ) / ( 1 + alpha_c * (z/D) )

        return K

    def K_q (K0q,Kinfq,alpha_q,z,D):

        K = ( K0q + Kinfq * alpha_q * (z/D) ) / ( 1 + alpha_q * (z/D) )

        return K

    def sigma_v(gamma, Depth_start, Depth_end, P0):

        sigma = P0 + gamma * ( Depth_end - Depth_start)

        return sigma

    def K_a (fi):

        K = np.tan(45 - (np.radians(fi)/2) )

        return K

    def K_p (fi):

        K = np.tan(45 + (np.radians(fi)/2) )

        return K

    sigmav = np.zeros([len(gamma)+1,1])

    " To obtain vertical stress: "

    for i in range (1,len (sigmav) ):

        if i == 1:
            sigmav[i,0] = sigma_v(gamma[i-1,0], Depth[i-1,0], Depth[i
    -1,1], 0)

        else:
```

```
            sigmav[i,0] = sigma_v(gamma[i-1,0], Depth[i-1,0], Depth[i
    -1,1], 0) + sigmav[i-1,0]

    sigmav = sigmav + sigma_p   #Adding overburden pressure to vertical
    stress

    R = D / 2  # Radius

    Deppts = np.zeros([len(Depth) + 1, 1])
    Deppts[1:, 0] = Depth[:, 1]
    Deppts = Deppts.flatten()
    sigmav = sigmav.flatten()

    sigmap_BH = np.zeros([ndz,len(x_pile)])
    Depth_dz = np.zeros([ndz,1])

    for j in range(ndz):

        dzp = dz/2+j*dz  # Ensures dzp increases predictably
        i = 0
        Depth_dz[j] = dzp

        while i < len(fi):

            if dzp <= Depth[i, 1]:
                phi = fi[i]
                cdz = c[i, 0]

                K0 = K_0(phi)
                K0q = K_0q(phi)
                K0c = K_0c(phi)
                Nc = N_c(phi)
                dcinf = d_c_inf(phi)
                Kinfc = K_infc(Nc, dcinf)
                Kinfq = K_infq(phi, Kinfc, K0)
                alphac = alpha_c(K0c, Kinfc, phi)
                alphaq = alpha_q(K0q, Kinfq, phi, K0)
                Kc = K_c(K0c, Kinfc, alphac, dzp+Depth_init, D) #Initial
    depth added here
                Kq = K_q(K0q, Kinfq, alphaq, dzp+Depth_init, D) #Initial
    depth added here
                sigmavp = np.interp(dzp, Deppts, sigmav)

                # Ensure all variables are scalars
                Kq = Kq.item() if isinstance(Kq, np.ndarray) else Kq
                sigmavp = sigmavp.item() if isinstance(sigmavp, np.
    ndarray) else sigmavp
                Kc = Kc.item() if isinstance(Kc, np.ndarray) else Kc
                cdz = cdz.item() if isinstance(cdz, np.ndarray) else cdz

                for k in range(len(x_pile)):
                    if k == 0:
                        Psi_w = Psi_pile[j,1]
                        Psi_c = Psi_pile[j,3]
                    else:
                        Psi_w = Psi_pile[j,0]
                        Psi_c = Psi_pile[j,2]
```

```
481
482                     if dz == 0.5:
483                         if k == 0:
484                             Psi_w = Psi_pile_05[j,1]
485                             Psi_c = Psi_pile_05[j,3]
486                         else:
487                             Psi_w = Psi_pile_05[j,0]
488                             Psi_c = Psi_pile_05[j,2]
489
490                     if include_red == 1:
491
492                         sigmap_BH[j,k] = Psi_w * Kq * sigmavp + Psi_c *
    Kc * cdz  # Compute sigmap_BH for reduced soil wedges
493
494                     else:
495                         sigmap_BH[j,k] = Kq * sigmavp + Kc * cdz    #
    Compute plastic limit for unreduced soil wedges
496
497                 # Print debugging info
498                 #print(f'j={j}, i={i}, dzp={dzp}, sigmavp={sigmavp}, Kq={
    Kq}, Kc={Kc}, cdz={cdz}, sigmap_BH[{j}]={sigmap_BH[j]}')
499                 break  # Breaks from the while loop since we found a
    valid `i`
500
501             else:
502                 i += 1
503                 if i >= len(fi):
504                     break
505
506     " To compute plastic stresses for Brom's extention method "
507
508     sigmap_BE = np.zeros(ndz) #Brom's extention method
509
510     for j in range(ndz):
511
512         dzp = dz * (j + 1)  # Ensures dzp increases predictably
513         i = 0
514         Depth_dz[j] = dzp
515         alpha = 3.35
516
517         while i < len(fi):
518
519             if dzp <= Depth[i, 1]:
520
521                 phi = fi[i]
522                 cdz = c[i, 0]
523                 gammaz = gamma[i]
524
525                 sigmap_BE[j] = alpha * gammaz * dzp * ( K_p(phi) - K_a(
    phi) ) + alpha * 2 * cdz *( np.sqrt( K_p(phi) ) + np.sqrt( K_a(phi) )
     )
526
527                 break
528
529             else:
530                 i += 1
531                 if i >= len(fi):
```

```
                    break

    #Calculation of elastic stiffness of springs

    #Values of Menard stiffness

    k_h = [0] * len(Depth_div)  #Initialize zero-matrix

    alpha_r_peat = 1
    alpha_r_clay = 2/3
    alpha_r_sand = 1/3

    #Rheological constant per layer
    alpha_r = [alpha_r_peat, alpha_r_clay, alpha_r_clay, alpha_r_clay, alpha_r_sand]

    alpha_peat = 3.5
    alpha_clay = 2.5
    alpha_sand = 0.85

    #Empirical alpha-factor per layer
    alpha_emp = [alpha_peat, alpha_clay, alpha_clay, alpha_clay, alpha_sand]

    E_m = [0] * len(Depth_div)  #Pressiometric modulus per layer

    R_0 = 0.3

    for i in range(len(q_c)):
        E_m[i] = q_c[i]*alpha_emp[i]

    for i in range(len(Depth_div)):
        if D < 0.3:
            k_h[i] = 1/ (1/(3*E_m[i]) * (1.3*R_0*(2.65*Radius/R_0)**alpha_r[i]+alpha_r[i]*Radius))

        elif D>= 0.3:
            k_h[i] = 1/ (2*Radius/E_m[i] * (4*2.65**alpha_r[i]+3*alpha_r[i])/18)


### CREATING GEOMETRY ####


    #Create beam geometry and partitions
    for i in range(len(x_pile)):    #Looping over the amount of pile rows

        #Creating geometry for first pile row
        Pile = f'Pile_{(i+1)}'

        s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__', sheetSize=5.0)
        g, v, d, c = s1.geometry, s1.vertices, s1.dimensions, s1.constraints
        s1.setPrimaryObject(option=STANDALONE)
        s1.Line(point1=(x_pile[i], pile_top), point2=(x_pile[i],-L))
        s1.VerticalConstraint(entity=g[2], addUndoState=False)
```

```
        p = mdb.models['Model-1'].Part(name=Pile, dimensionality=
    TWO_D_PLANAR,
            type=DEFORMABLE_BODY)
        p = mdb.models['Model-1'].parts[Pile]
        p.BaseWire(sketch=s1)
        s1.unsetPrimaryObject()
        p = mdb.models['Model-1'].parts[Pile]

        del mdb.models['Model-1'].sketches['__profile__']

        print(f"Geometry for {Pile} created")

        #Create datum point
        for j in range(ndz):

            z = -(dz/2 + j*dz)     #z-location for itteration

            p = mdb.models['Model-1'].parts[Pile]
            p.DatumPointByCoordinate(coords=(x_pile[i], z, 0.0))
            p = mdb.models['Model-1'].parts[Pile]

        for k in range(ndz):

    #Partition the Pile and creating nodes for spring connection

            p = mdb.models['Model-1'].parts[Pile]
            e, v, d = p.edges, p.vertices, p.datums
            p.PartitionEdgeByPoint(edge=e[k], point=d[k+2])

        print(f"Partitioning of {Pile} successful")

    #Create set of the pile
        p = mdb.models['Model-1'].parts[Pile]
        e = p.edges
        all_edges = p.edges[:]
        all_nodes = p.nodes[:]
        name_pile_set = f'Set_{i+1}'
        set_name = name_pile_set
        p.Set(edges=all_edges, name=set_name)

        print(f"Sets created:{set_name}")

    #Assign material and set rotation
        p = mdb.models['Model-1'].parts[Pile]
        region = p.sets[set_name]
        p = mdb.models['Model-1'].parts[Pile]
        p.SectionAssignment(region=region, sectionName=Section, offset
    =0.0,
            offsetType=MIDDLE_SURFACE, offsetField='',
            thicknessAssignment=FROM_SECTION)
        p = mdb.models['Model-1'].parts[Pile]
        region=p.sets[set_name]
        p = mdb.models['Model-1'].parts[Pile]
        p.assignBeamSectionOrientation(region=region, method=N1_COSINES,
    n1=(x_pile[i], pile_top, -1.0))

        print(f"Section assigned for {Pile}")
```

```
637
638     #Adding pile to assembly
639         a = mdb.models['Model-1'].rootAssembly
640         a.DatumCsysByDefault(CARTESIAN)
641         p = mdb.models['Model-1'].parts[Pile]
642         a.Instance(name=f'Pile_{i+1}', part=p, dependent=ON)
643 
644         print(f"{Pile} added to assembly")
645 
646 
647     #Meshing
648         p.seedPart(size=meshsize_pile, deviationFactor=0.1, minSizeFactor
    =0.1)
649         p.generateMesh()
650 
651         print(f"Mesh created for {Pile}, size={meshsize_pile}")
652 
653         print(f"Datum points for {Pile} created")
654 
655         #print(f"Node coordinates in {Pile}")
656         #for node in p.nodes:
657             #print(f"Node label: {node.label}, Coordinates: {node.
    coordinates}")
658 
659     #Creating fixed BC at base of pile
660 
661         v1 = a.instances[Pile].vertices
662         Pile_base_vertice = v1.findAt(((x_pile[i], -L, 0.0),))
663         region = regionToolset.Region(vertices=Pile_base_vertice)
664         mdb.models['Model-1'].DisplacementBC(name=f'BC_Bot_P{i+1}',
    createStepName='Initial',
665             region=region, u1=U1_PB, u2=U2_PB, ur3=U3_PB, amplitude=UNSET
    ,
666             distributionType=UNIFORM, fieldName='', localCsys=None)
667 
668     #Creating dummy spring at first spring position for each pile
669         name_spring = f'Spring_Pile_{i+1}_Dummy'
670 
671         a = mdb.models['Model-1'].rootAssembly
672         v1 = a.instances[Pile].vertices
673         verts1 = v1.findAt(((x_pile[i], z_depth[1], 0.0), ))
674         region=regionToolset.Region(vertices=verts1)
675         mdb.models['Model-1'].rootAssembly.engineeringFeatures.
    SpringDashpotToGround(
676         name=name_spring, region=region, orientation=None, dof=1,
677         springBehavior=ON, springStiffness=1, dashpotBehavior=OFF,
678         dashpotCoefficient=0.0)
679 
680 
681     #Create timber deck - connecting piles with a beam element
682 
683 
684     num_tie = 0 #Introducing first name of tie
685 
686     for i in range(len(x_pile)-1):
687 
688         Name_Deck = f'T_Deck_{i+1}'
```

```
689
690         s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile_Deck
    ', sheetSize=5.0)
691         g, v, d, c = s1.geometry, s1.vertices, s1.dimensions, s1.
    constraints
692         s1.setPrimaryObject(option=STANDALONE)
693         s1.Line(point1=(x_pile[i], pile_top), point2=(x_pile[i+1],
    pile_top))
694         #s1.VerticalConstraint(entity=g[2], addUndoState=False)
695         p = mdb.models['Model-1'].Part(name=Name_Deck, dimensionality=
    TWO_D_PLANAR,
696             type=DEFORMABLE_BODY)
697         p = mdb.models['Model-1'].parts[Name_Deck]
698         p.BaseWire(sketch=s1)
699         s1.unsetPrimaryObject()
700         p = mdb.models['Model-1'].parts[Name_Deck]
701         del mdb.models['Model-1'].sketches['__profile_Deck']
702
703     #Partition the Timber deck
704
705         if i == 0:
706             if u_partition_deck < 1:
707                 e, v, d = p.edges, p.vertices, p.datums
708                 p.PartitionEdgeByParam(edges=e[0], parameter=
    u_partition_deck)
709                 print(f"Partitioning of {Name_Deck} successful")
710             else:
711                 print("Timber deck is not partitioned")
712
713             if u_partition_load < 1:
714                 e, v, d = p.edges, p.vertices, p.datums
715                 p.PartitionEdgeByParam(edges=e[0], parameter=
    u_partition_load)
716                 print(f"Partitioning of {Name_Deck} for load application
    successful")
717             else:
718                 print(f"Timber deck is not partitioned for load
    application")
719
720     #Create set of the Timber deck
721         p = mdb.models['Model-1'].parts[Name_Deck]
722         e = p.edges
723         all_edges = p.edges[:]
724         all_nodes = p.nodes[:]
725         set_name = f'Set_{i+1}'
726         p.Set(edges=all_edges, name=set_name)
727         print(f"Sets created:{set_name}")
728
729     #Assign material and set orientation of beam
730         p = mdb.models['Model-1'].parts[Name_Deck]
731         region = p.sets[set_name]
732         p = mdb.models['Model-1'].parts[Name_Deck]
733         p.SectionAssignment(region=region, sectionName=Section_Deck,
    offset=0.0,
734             offsetType=MIDDLE_SURFACE, offsetField='',
735             thicknessAssignment=FROM_SECTION)
736         p = mdb.models['Model-1'].parts[Name_Deck]
```

```
        region=p.sets[set_name]
        p = mdb.models['Model-1'].parts[Name_Deck]
        p.assignBeamSectionOrientation(region=region, method=N1_COSINES, n1=(0.0, 0.0, -1.0))
        print(f"Section assigned for {Name_Deck}")

    #Adding Timber deck to assembly
        a = mdb.models['Model-1'].rootAssembly
        p = mdb.models['Model-1'].parts[Name_Deck]
        a.Instance(name=Name_Deck, part=p, dependent=ON)

        print(f"{Name_Deck} added to assembly")

    #Meshing Timber deck
        p.seedPart(size=meshsize_pile, deviationFactor=0.1, minSizeFactor=0.1)
        p.generateMesh()

        print(f"Mesh created for {Name_Deck}, size={meshsize_pile}")


### ADDING LOADS TO MODEL ###


    #Adding lineload from wall pressure, vertical and horizontal

        name_load_v = 'Wall-pressure'
        name_load_H = 'FH-Wall'

        if i == 0:
            mdb.models['Model-1'].ExpressionField(name='AnalyticalField_wall', localCsys=None,
                description='Load from wall on timber deck', expression=f'{sigma_wall_max}-{sigma_wall_max}*X/{B_eff}')
            a = mdb.models['Model-1'].rootAssembly
            s1 = a.instances[Name_Deck].edges
            edge_wall = s1.findAt(((x_pile[i], pile_top, 0),))
            region = regionToolset.Region(edges=edge_wall)
            mdb.models['Model-1'].LineLoad(name=name_load_v, createStepName='Step-1',
                region=region, comp1=0, comp2=-1.0, distributionType=FIELD, field='AnalyticalField_wall')

            mdb.models['Model-1'].LineLoad(name=name_load_H, createStepName='Step-1',
                region=region, comp1=q_h_wall)

    #Adding lineload from vertical stress on timber deck

        name_vertical_stress = f'Sigma_v_{i+1}'

        if i == 0:
            if u_partition_deck < 1:
                s1 = a.instances[Name_Deck].edges
                edge_deck = s1.findAt(((x_pile[i+1], pile_top, 0),))
                region = regionToolset.Region(edges=edge_deck)
```

```
                mdb.models['Model-1'].LineLoad(name=name_vertical_stress,
    createStepName='Step-1',
                    region=region, comp2=sigma_v_soil)

            else:
                print("No vertical stress applied to first timber deck
    span")

        else:
            s1 = a.instances[Name_Deck].edges
            edge_deck = s1.findAt(((x_pile[i+1], pile_top, 0),))
            region = regionToolset.Region(edges=edge_deck)
            mdb.models['Model-1'].LineLoad(name=name_vertical_stress,
    createStepName='Step-1',
                region=region, comp2=sigma_v_soil)


### CONNECTING TIMBER DECK TO PILES


    #Creating tie between timber deck and pile

        Pile1 = f'Pile_{(i+1)}'
        Pile2 = f'Pile_{(i+2)}'
        num_tie = num_tie+1

        if i == 0:
            #Finding tie point for TD, first pile
            a = mdb.models['Model-1'].rootAssembly
            v1_Deck = a.instances[Name_Deck].vertices
            region1 = v1_Deck.findAt(((x_pile[i], pile_top, 0),))
            region_Deck=regionToolset.Region(vertices=region1)

            #Finding tie for first pile
            v1_Pile = a.instances[Pile1].vertices
            verts_Pile = v1_Pile.findAt(((x_pile[i], pile_top, 0),))
            region_Pile=regionToolset.Region(vertices=verts_Pile)

            #Creating tie for first pile to deck
            mdb.models['Model-1'].Tie(name=f'Tie-{num_tie}', main=
    region_Deck, secondary=region_Pile,
                positionToleranceMethod=COMPUTED, adjust=ON, tieRotations
    =OFF ,#SPECIFY HINGE HERE
                constraintRatioMethod=SPECIFIED, constraintRatio=0.0,
    thickness=ON) #SET RATIO TO 0 FOR HINGE

            num_tie = num_tie+1

            #Finding tie for second pile
            v2_Pile = a.instances[Pile2].vertices
            verts_Pile2 = v2_Pile.findAt(((x_pile[i+1], pile_top, 0),))
            region_Pile2=regionToolset.Region(vertices=verts_Pile2)

            #Finding tie for TD, second pile
            region2 = v1_Deck.findAt(((x_pile[i+1], pile_top, 0),))
            region_Deck2=regionToolset.Region(vertices=region2)

```

```
837             #Creating tie for second pile to deck
838             mdb.models['Model-1'].Tie(name=f'Tie-{num_tie}', main=
    region_Deck2, secondary=region_Pile2,
839                 positionToleranceMethod=COMPUTED, adjust=ON, tieRotations
    =OFF ,#SPECIFY HINGE HERE
840                 constraintRatioMethod=SPECIFIED, constraintRatio=0.0,
    thickness=ON) #SET RATIO TO 0 FOR HINGE
841 
842         else:
843             name_Deck_prev = f'T_Deck_{i}'
844 
845             #Finding node for previous Deck
846             v1_Deck_prev = a.instances[name_Deck_prev].vertices
847             verts_Deck_prev = v1_Deck_prev.findAt(((x_pile[i], pile_top,
    0),))
848             region_Deck_prev = regionToolset.Region(vertices=
    verts_Deck_prev)
849 
850             #Finding node for current Deck
851             v1_Deck = a.instances[Name_Deck].vertices
852             verts_Deck1 = v1_Deck.findAt(((x_pile[i], pile_top, 0),))
853             region_Deck1 = verts_Deck1=regionToolset.Region(vertices=
    verts_Deck1)
854 
855             #Creating rigid connection between current and previous Deck
    instance
856             mdb.models['Model-1'].Tie(name=f'Tie-{num_tie}', main=
    region_Deck_prev, secondary=region_Deck1,
857                 positionToleranceMethod=COMPUTED, adjust=ON, tieRotations
    =ON ,#SPECIFIED RIGID CONNECTION HERE
858                 thickness=ON)
859 
860             num_tie = num_tie +1
861 
862             #Finding tie for second pile
863             v2_Pile = a.instances[Pile2].vertices
864             verts_Pile2 = v2_Pile.findAt(((x_pile[i+1], pile_top, 0),))
865             region_Pile2=regionToolset.Region(vertices=verts_Pile2)
866 
867             #Finding tie for TD, second pile
868             region2 = v1_Deck.findAt(((x_pile[i+1], pile_top, 0),))
869             region_Deck2=regionToolset.Region(vertices=region2)
870 
871             #Creating tie for second pile to deck
872             mdb.models['Model-1'].Tie(name=f'Tie-{num_tie}', main=
    region_Deck2, secondary=region_Pile2,
873                 positionToleranceMethod=COMPUTED, adjust=ON, tieRotations
    =OFF ,#SPECIFY HINGE HERE
874                 constraintRatioMethod=SPECIFIED, constraintRatio=0.0,
    thickness=ON) #SET RATIO TO 0 FOR HINGE
875 
876     #Adding horizontal load on top of the front pile
877 
878     a = mdb.models['Model-1'].rootAssembly
879     v1 = a.instances['Pile_1'].vertices
880     verts1 = v1.findAt(((0, pile_top, 0.0), ))
881     region = regionToolset.Region(vertices=verts1)
```

```
    mdb.models['Model-1'].ConcentratedForce(name='FH_Pile-1', createStepName='Step-1',
        region=region, cf1=F_s, distributionType=UNIFORM, field='',
        localCsys=None)

    #Adding cap-moment on top of piles
    if include_rot == 1:
        for i in range(len(x_pile)):

            Pile_it = f'Pile_{i+1}'
            v1 = a.instances[Pile_it].vertices
            verts1 = v1.findAt(( (x_pile[i], pile_top, 0.0), ))
            region = regionToolset.Region(vertices=verts1)
            mdb.models['Model-1'].Moment(name=f'Mrot_{Pile_it}', createStepName='Step-1',
                region=region, cm3=Mrot, distributionType=UNIFORM, field='',
                localCsys=None)

    #Add horizontal load from bottom of sheet pile wall to front pile row

    N_qs = int(np.ceil(L_bot_sp/dz))
    e1 = a.instances['Pile_1'].edges

    for i in range(N_qs):

        name_qs = f'qs_{z1_sp}'

        edge1 = e1.findAt(((0, z1_sp, 0), ))
        region = regionToolset.Region(edges=edge1)
        mdb.models['Model-1'].LineLoad(name=name_qs, createStepName='Step-1',
            region=region, comp1=q_s)

        z1_sp = z1_sp - dz

    #Creating gravity loading
    mdb.models['Model-1'].Gravity(name='Gravity', createStepName='Step-1',
        comp2=-1.0, distributionType=UNIFORM, field='')


### CREATING JOB FOR SIMULATION


    #Creating the job

    mdb.Job(name=name_job, model='Model-1', description='', type=ANALYSIS,
            atTime=None, waitMinutes=0, waitHours=0, queue=None, memory=90,
            memoryUnits=PERCENTAGE, getMemoryFromAnalysis=True,
            explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE,
        echoPrint=ON,
            modelPrint=ON, contactPrint=OFF, historyPrint=OFF,
        userSubroutine='',
            scratch='', resultsFormat=ODB, numThreadsPerMpiProcess=1,
```

```
            multiprocessingMode=DEFAULT, numCpus=1, numGPUs=0)

    mdb.models['Model-1'].fieldOutputRequests['F-Output-1'].setValues(
    variables=(
            'S','MISES','U','RF','UR','CF','SF','NFORCSO','CSTRESS','
    CFORCE','GRAV'))

    #Write the input file to insert spring stiffness to
    mdb.jobs[name_job].writeInput(consistencyChecking=OFF)

    #Create job running analysis from input file

    name_job_inp = f'{name_job}_inp'

    name_inp = f'{name_job}.inp'

    mdb.JobFromInputFile(name=name_job_inp,
            inputFileName=f'{cwd}\\{name_inp}', type=ANALYSIS,
            atTime=None, waitMinutes=0, waitHours=0, queue=None, memory
    =90,
            memoryUnits=PERCENTAGE, getMemoryFromAnalysis=True,
            explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE,
            userSubroutine='', scratch='', resultsFormat=ODB,
            numThreadsPerMpiProcess=1, multiprocessingMode=DEFAULT,
    numCpus=1,
            numGPUs=0)


### CREATING SPRING PROPERTIES ###


    #Create spring properties to be paste into the model

    name_file = "Spring-properties.txt"

    open(name_file, "w")

    count = 0       #Initiating counting for total number of springs

    sigma_Plim = sigmap_BH

    for i in range(len(x_pile)):
        for j in range(ndz):

            for depth, stiffness in zip(Depth_div, k_h):
                if z_depth[j] >= depth:
                    Elastic = stiffness

                    break

            f = open(name_file, "a")

            count = count+1

            k = Elastic*dz*D #Elastic spring stiffness per spring [kN/m]

            P_lim = sigma_Plim[j,i]*dz*D  #Plastic limit per spring [kN]
```

```
983
984             x = P_lim / k   #Displacement of spring at yielding
985
986             #Spring name at z_depth j
987             name_spring = f'Spring_P{i+1}_{j+1}'
988
989             #First row, Spring elset and name
990             row_1 = f'*Spring, elset={name_spring}-spring, nonlinear\n'
991
992             #Second input, DOF
993             row_2 = '1\n'
994
995             #Third input, NL properties
996             row_3 = f'-{P_lim*1.001},-{x*1000000}\n'
997             row_4 = f'-{P_lim},-{x}\n'
998             row_5 = f'{P_lim},{x}\n'
999             row_6 = f'{P_lim*1.001},{x*1000000}\n'
1000
1001            #Fourth input, element assignment
1002            row_7 = f'*Element, type=Spring1, elset={name_spring}-spring\
    n'
1003            if i == len(x_pile)-1 and j == ndz-1:
1004                row_8 = f'{count}, Pile_{i+1}.{j+2}'
1005            else:
1006                row_8 = f'{count}, Pile_{i+1}.{j+2}\n'
1007
1008            f.write(row_1)
1009            f.write(row_2)
1010            f.write(row_3)
1011            f.write(row_4)
1012            f.write(row_5)
1013            f.write(row_6)
1014            f.write(row_7)
1015            f.write(row_8)
1016            f.close()
1017
1018        print(f'Spring stiffness for Pile_{i+1} added in {name_file}')
1019
1020    # Inserting spring properties in the job-file
1021
1022    with open(name_inp, 'r') as file:
1023        main_text = file.read()
1024
1025    with open(name_file, 'r') as file:
1026        replacement_text = file.read()
1027
1028    repl_1 = '*Spring, elset=Spring_Pile_1_Dummy-spring'
1029    repl_12 = '*Spring, elset=Spring_Pile_2_Dummy-spring'
1030    repl_13 = '*Spring, elset=Spring_Pile_3_Dummy-spring'
1031    repl_14 = '*Spring, elset=Spring_Pile_4_Dummy-spring'
1032
1033    repl_2 = "1"
1034    repl_3 = "1."
1035
1036    repl_4 = '*Element, type=Spring1, elset=Spring_Pile_1_Dummy-spring'
1037    repl_42 = '*Element, type=Spring1, elset=Spring_Pile_2_Dummy-spring'
1038    repl_43 = '*Element, type=Spring1, elset=Spring_Pile_3_Dummy-spring'
```

```
    repl_44 = '*Element, type=Spring1, elset=Spring_Pile_4_Dummy-spring'

    repl_5 = '1, Pile_1.3'
    repl_52 = '2, Pile_2.3'
    repl_53 = '3, Pile_3.3'
    repl_54 = '4, Pile_4.3'

    repl_row1 = f'{repl_1}\n{repl_2}\n{repl_3}\n{repl_4}\n{repl_5}'
    repl_row2 = f'{repl_12}\n{repl_2}\n{repl_3}\n{repl_42}\n{repl_52}'
    repl_row3 = f'{repl_13}\n{repl_2}\n{repl_3}\n{repl_43}\n{repl_53}'
    repl_row4 = f'{repl_14}\n{repl_2}\n{repl_3}\n{repl_44}\n{repl_54}'

    if len(x_pile) == 1:

        text_to_replace = repl_row1

    if len(x_pile) == 2:

        text_to_replace = f'{repl_row1}\n{repl_row2}'

    if len(x_pile) == 3:

        text_to_replace = f'{repl_row1}\n{repl_row2}\n{repl_row3}'

    if len(x_pile) == 4:

        text_to_replace = f'{repl_row1}\n{repl_row2}\n{repl_row3}\n{
    repl_row4}'

    updated_text = main_text.replace(text_to_replace, replacement_text)

    with open(name_inp, 'w') as file:
        file.write(updated_text)

    print(f"Text replacement completed successfully!\n {text_to_replace}\
    n is replaced with {name_file}.")



    #Submit job for analysis
    mdb.jobs[name_job_inp].submit(consistencyChecking=OFF)
    mdb.jobs[name_job_inp].waitForCompletion()


### POST PROCESSING ###


    #Name of database
    odb_name = f'{name_job_inp}.odb'

    #Open database
    odb_path = f'{cwd}/{odb_name}'
    odb = visualization.openOdb(path=odb_path)

    #Plot von Mises stresses on Deformed shape

    session_viewport = session.viewports['Viewport: 1']
```

```
    #Setting viewport
    session_viewport.setValues(displayedObject=odb)
    session_viewport.view.fitView()
    session_viewport.odbDisplay.display.setValues(plotState=(
        CONTOURS_ON_DEF, ))
    session_viewport.odbDisplay.contourOptions.setValues(
        contourStyle=CONTINUOUS)
    session_viewport.odbDisplay.commonOptions.setValues(
        deformationScaling=UNIFORM, uniformScaleFactor=100)
    session_viewport.odbDisplay.basicOptions.setValues(
        renderBeamProfiles=ON, beamScaleFactor=1)
    session_viewport.odbDisplay.setPrimaryVariable(
        variableLabel='BEAM_STRESS', outputPosition=ELEMENT_NODAL,
    refinement=(
        INVARIANT, 'Mises'), )

    #Printing von-Mises stress
    name_image = f'S-Mises_{name_job}'

    session.printOptions.setValues(vpBackground=ON)
    session.printToFile(fileName=name_image, format=PNG, canvasObjects=(
        session.viewports['Viewport: 1'], ))

    #Plot u1-displacements on Deformed shape

    #Setting viewport
    session_viewport.odbDisplay.setPrimaryVariable(
            variableLabel='U', outputPosition=NODAL, refinement=(
    COMPONENT, 'U1'),
            )
    session_viewport.odbDisplay.basicOptions.setValues(
        renderBeamProfiles=OFF,  )


    #Printing U1-displacements

    name_image_U1 = f'U1-Displacements_{name_job}'

    session.printOptions.setValues(vpBackground=ON)
    session.printToFile(fileName=name_image_U1, format=PNG, canvasObjects
    =(
        session.viewports['Viewport: 1'], ))

    #Saving maximum von mises stresses

        #Naming of csv-output file
    csv_path = os.path.join(cwd, 'Output_Abaqus.csv')
    csv_path_max = os.path.join(cwd, 'Output_Abaqus_MaxStress.csv')

    #Set current ODB
    odb = session.openOdb(name=odb_name)

    #Extracting stress field from model
    Stress_field = odb.steps['Step-1'].frames[-1].fieldOutputs['S']
    Displ_field = odb.steps['Step-1'].frames[-1].fieldOutputs['U']
```

```
    #Initializing list to save output data
    vonMises = []
    instances = []
    node_lab = []
    Displ1 = []
    Instance_displ = []

    #Initializing list to save output data for springs
    vonMises_springs = []

    unit = 1000 #1 for m and kPa, 1000 for mm and MPa

    #Storing stresses and corresponding instance
    for stress_value in Stress_field.values:
        if stress_value.instance is not None:
            vonMises.append(stress_value.mises)
            instances.append(stress_value.instance.name)
        else:
            vonMises_springs.append(stress_value.mises)

    #Storing displacements
    for displ_value in Displ_field.values:
        if displ_value.instance is not None:
            Displ1.append(displ_value.data[0])
            Instance_displ.append(displ_value.instance.name)

    #Changing units in data output
    Displ1 = [round((x * unit),2) for x in Displ1]
    vonMises = [round((x / unit),3) for x in vonMises]

    #Saving max stress for each pile

    if unit == 1:
        D_unit = '[m]'
        S_unit = '[kPa]'
    else:
        D_unit = '[mm]'
        S_unit = '[MPa]'

    unique_piles = set(instances)

    max_values = {pile: 0 for pile in unique_piles}

        #Storing maximum stress for each element
    for value, pile in zip(vonMises, instances):
        if value > max_values[pile]:
            max_values[pile] = value

    print(f"Maximum value for {pile}: {max_values[pile]}")

        #Writing out max stresses for elements for current itteration
    if it == 0:
        file = open(csv_path_max, 'w', newline='')
    else:
        file = open(csv_path_max, 'a', newline='')

    writer = csv.writer(file, delimiter=';')
```

```
    if it == 0:
        writer.writerow(['Job name', f'Max stress {S_unit}', 'Element nr'])

    for pile, value in max_values.items():
        writer.writerow([ name_job, value, pile ])

    print(f"Maximum stress in each pile for job {name_job} saved to {csv_path_max}")

    #Max stress for model
    max_stress = max(vonMises)
    max_U1 = max(Displ1)
    min_U1 = min(Displ1)

    #Storing data for writing to Excel

        #Mises stress and instances
    rows = len(vonMises)
    col = 4
    data = np.zeros((rows,col), dtype = list)

        #Displacement vector
    rows2 = len(Displ1)
    col2 = 4
    data2 = np.zeros((rows2,col2), dtype = list)

    #Storing the data for stresses etc

    data[:,0] = name_job
    data[:,1] = D*unit
    data[:,2] = instances
    data[:,3] = vonMises

    data2[:,0] = name_job
    data2[:,1] = D*unit
    data2[:,2] = Instance_displ
    data2[:,3] = Displ1


    #Writing output data to Excel

    if it == 0:
        file = open(csv_path, mode='w', newline='')
    else:
        file = open(csv_path, mode='a', newline='')

    writer = csv.writer(file, delimiter=';')
    writer.writerow(['Job Name', f'Pile Diameter {D_unit}', 'Pile', f'Smises {S_unit}'])

    writer.writerows(data)
    writer.writerow('')
    writer.writerow(['Job Name', f'Pile Diameter {D_unit}', 'Pile', f'U1 {D_unit}'])
    writer.writerows(data2)
```

```
    writer.writerow('')
    #writer.writerow(['Max vonMises:', max_stress, 'Max U1:', max_U1, '
    Min U1:', min_U1, 'Diameter:', D*unit, 'dz:', dz])
    #writer.writerow('')

    # Append data to the CSV file
    print(f"Abaqus Outputdata for {name_job} saved to {csv_path}")

    odb.close() # Closing ODB-file

#Closing csv-file after exporting results from all itterations

file.close()

print(f'{num_rows} analysis completed.' )

print(f'min / max dz interval: {min(dz_it)} / {max(dz_it)}')

print(f'min / max diameter interval: {min(D_it)} / {max(D_it)}')
```

H Case study 2 - Intermediate steps, Grasshopper

The following figures shows the intermediate steps to obtain the reduced wedges for the Gothenburg straight case.

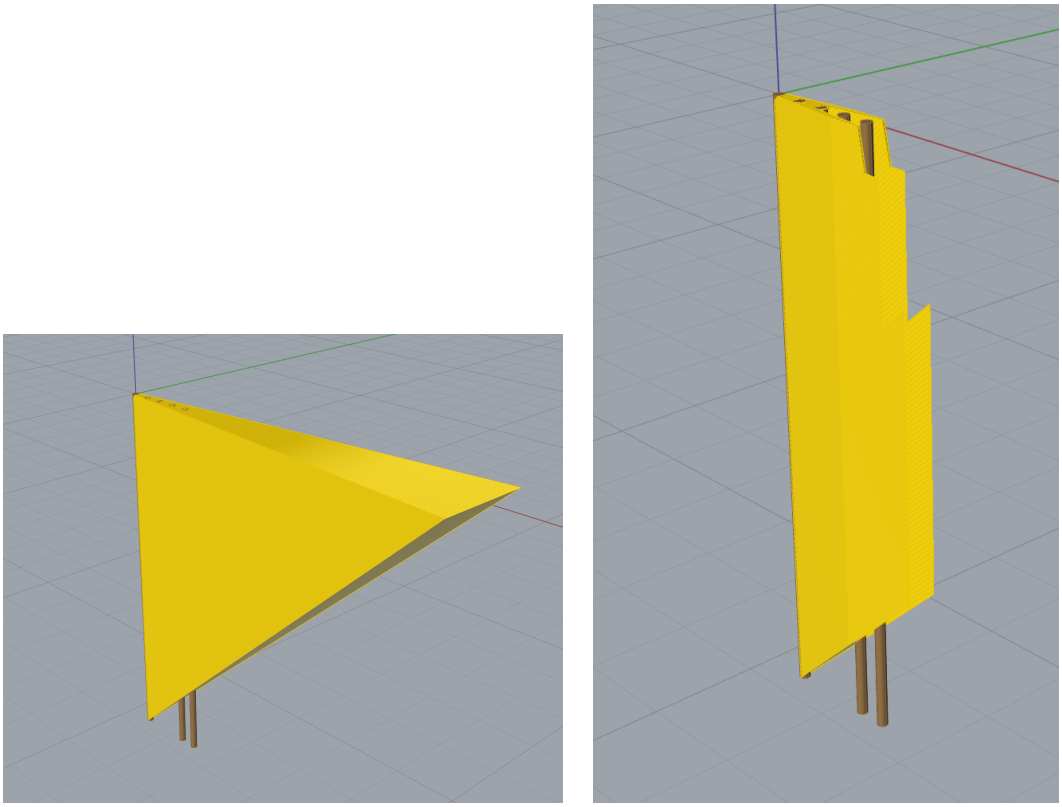


Figure H.1: Comparison of soil wedges reduced and unreduced for pile 1

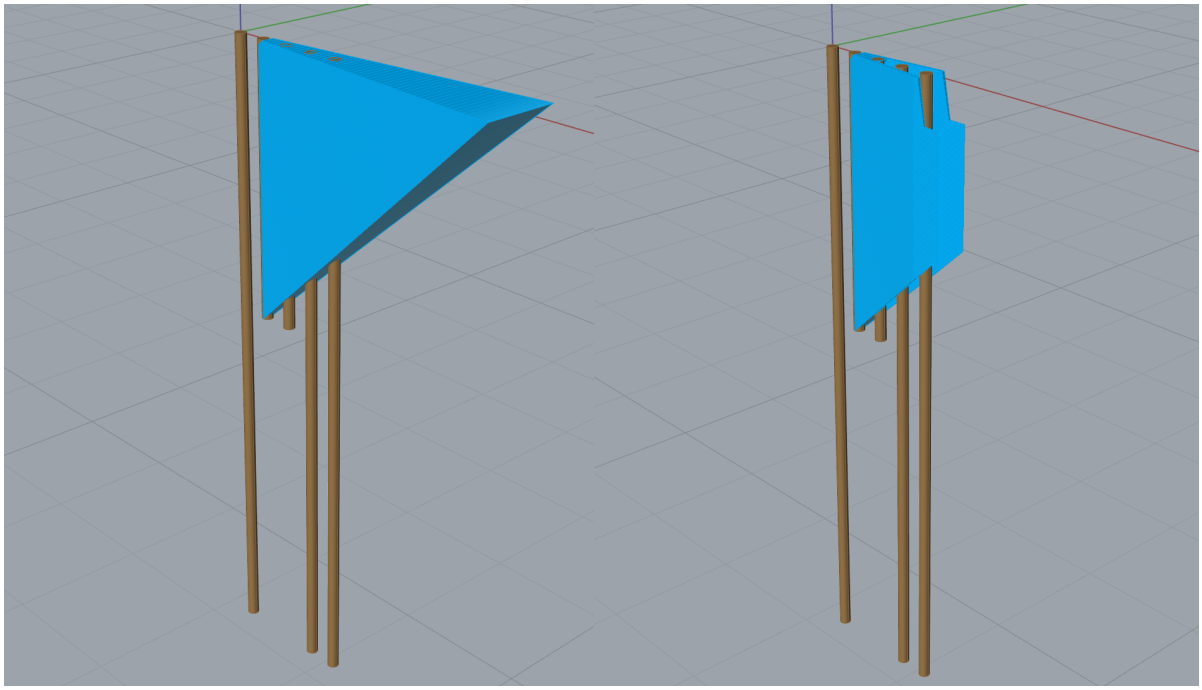


Figure H.2: Comparison of soil wedges reduced and unreduced for pile 2

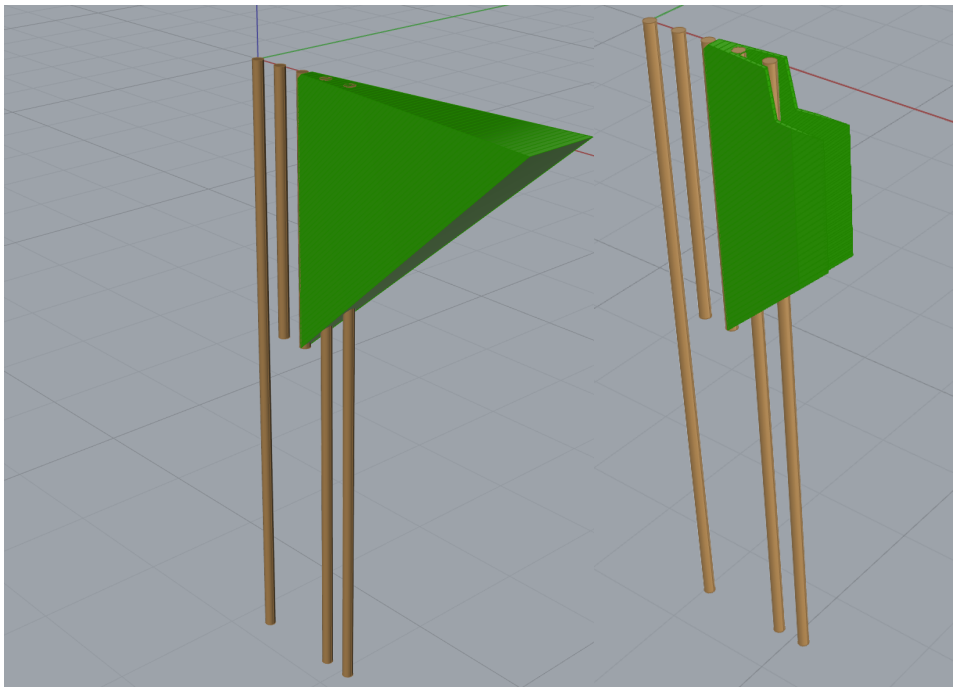


Figure H.3: Comparison of soil wedges reduced and unreduced for pile 3

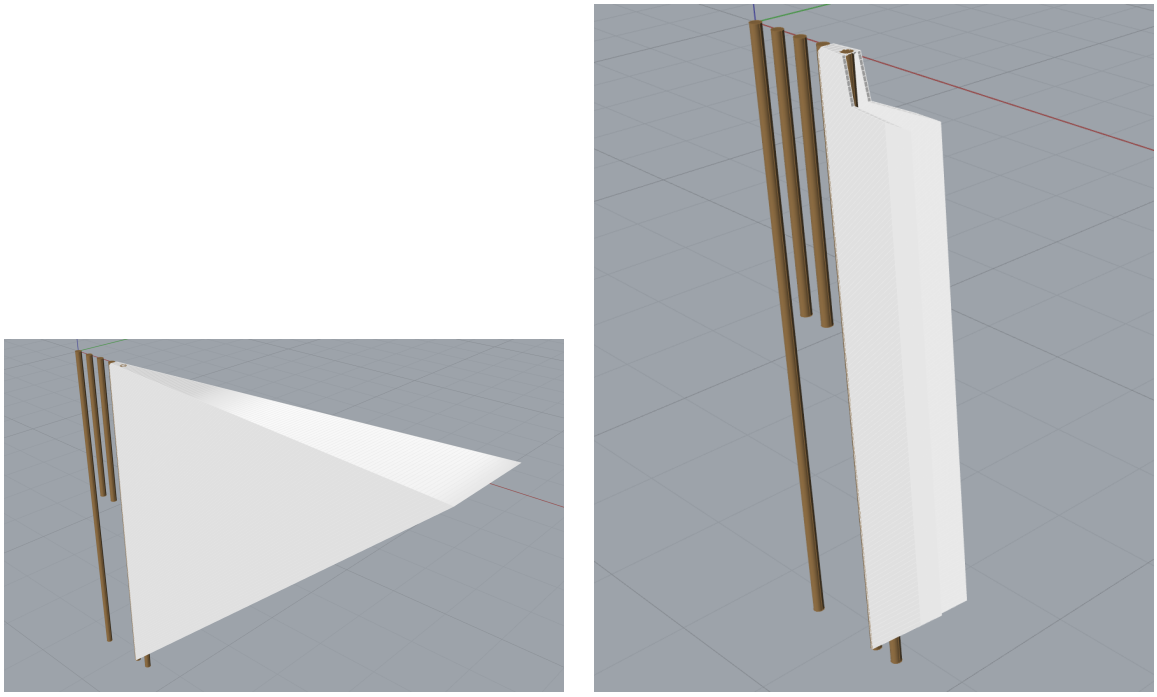


Figure H.4: Comparison of soil wedges reduced and unreduced for pile 4

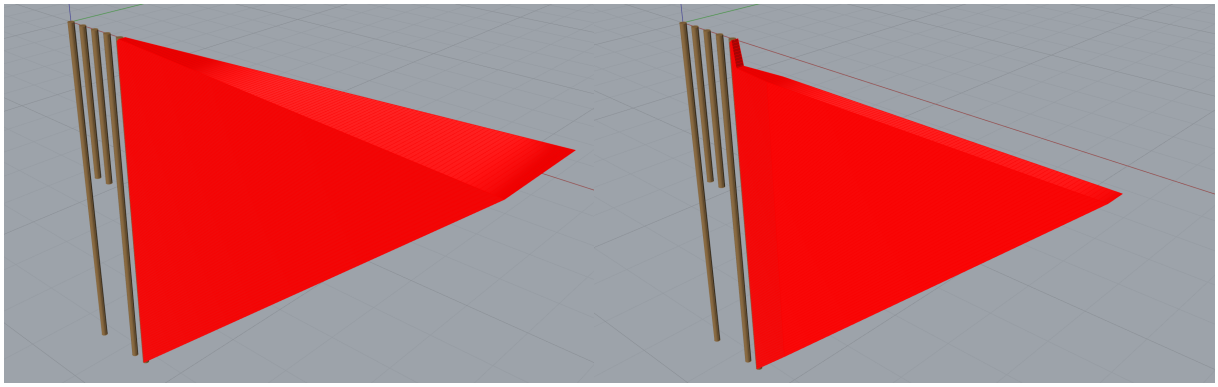


Figure H.5: Comparison of soil wedges reduced and unreduced for pile 5

I Case study 2 - Python codes for Grasshopper

The python code utilised to obtain the reduction factor for the Gothenburg Straight model is shown below. Is good to mention, that this code is written in Python prompt built in Grasshopper.

```
# -*- coding: utf-8 -*-
"""
Created on Wed May 21 11:49:06 2025

@author: SE1G4E
"""

"""Grasshopper Script (With Numpy)"""
# requirements: numpy

import numpy as np
import ghpythonlib.treehelpers as th
import csv
import os

V_Fill_P1_UR = np.array(V_Fill_P1_UR)
V_Fill_P2_UR = np.array(V_Fill_P2_UR)
V_Fill_P3_UR = np.array(V_Fill_P3_UR)
V_Fill_P4_UR = np.array(V_Fill_P4_UR)
V_Fill_P5_UR = np.array(V_Fill_P5_UR)

V_Clay1_P1_UR = np.array(V_Clay1_P1_UR)
V_Clay1_P2_UR = np.array(V_Clay1_P2_UR)
V_Clay1_P3_UR = np.array(V_Clay1_P3_UR)
V_Clay1_P4_UR = np.array(V_Clay1_P4_UR)
V_Clay1_P5_UR = np.array(V_Clay1_P5_UR)

V_Clay2_P1_UR = np.array(V_Clay2_P1_UR)
V_Clay2_P2_UR = np.array(V_Clay2_P2_UR)
V_Clay2_P3_UR = np.array(V_Clay2_P3_UR)
V_Clay2_P4_UR = np.array(V_Clay2_P4_UR)
V_Clay2_P5_UR = np.array(V_Clay2_P5_UR)

A_Fill_P1_UR = np.array(A_Fill_P1_UR)
A_Fill_P2_UR = np.array(A_Fill_P2_UR)
A_Fill_P3_UR = np.array(A_Fill_P3_UR)
A_Fill_P4_UR = np.array(A_Fill_P4_UR)
A_Fill_P5_UR = np.array(A_Fill_P5_UR)

A_Clay1_P1_UR = np.array(A_Clay1_P1_UR)
A_Clay1_P2_UR = np.array(A_Clay1_P2_UR)
A_Clay1_P3_UR = np.array(A_Clay1_P3_UR)
A_Clay1_P4_UR = np.array(A_Clay1_P4_UR)
A_Clay1_P5_UR = np.array(A_Clay1_P5_UR)

A_Clay2_P1_UR = np.array(A_Clay2_P1_UR)
A_Clay2_P2_UR = np.array(A_Clay2_P2_UR)
A_Clay2_P3_UR = np.array(A_Clay2_P3_UR)
A_Clay2_P4_UR = np.array(A_Clay2_P4_UR)
A_Clay2_P5_UR = np.array(A_Clay2_P5_UR)
```

```
51
52 V_Fill_P1_R = np.array(V_Fill_P1_R)
53 V_Fill_P2_R = np.array(V_Fill_P2_R)
54 V_Fill_P3_R = np.array(V_Fill_P3_R)
55 V_Fill_P4_R = np.array(V_Fill_P4_R)
56 V_Fill_P5_R = np.array(V_Fill_P5_R)
57
58 V_Clay1_P1_R = np.array(V_Clay1_P1_R)
59 V_Clay1_P2_R = np.array(V_Clay1_P2_R)
60 V_Clay1_P3_R = np.array(V_Clay1_P3_R)
61 V_Clay1_P4_R = np.array(V_Clay1_P4_R)
62 V_Clay1_P5_R = np.array(V_Clay1_P5_R)
63
64 V_Clay2_P1_R = np.array(V_Clay2_P1_R)
65 V_Clay2_P2_R = np.array(V_Clay2_P2_R)
66 V_Clay2_P3_R = np.array(V_Clay2_P3_R)
67 V_Clay2_P4_R = np.array(V_Clay2_P4_R)
68 V_Clay2_P5_R = np.array(V_Clay2_P5_R)
69
70 A_Fill_P1_R = np.array(A_Fill_P1_R)
71 A_Fill_P2_R = np.array(A_Fill_P2_R)
72 A_Fill_P3_R = np.array(A_Fill_P3_R)
73 A_Fill_P4_R = np.array(A_Fill_P4_R)
74 A_Fill_P5_R = np.array(A_Fill_P5_R)
75
76 A_Clay1_P1_R = np.array(A_Clay1_P1_R)
77 A_Clay1_P2_R = np.array(A_Clay1_P2_R)
78 A_Clay1_P3_R = np.array(A_Clay1_P3_R)
79 A_Clay1_P4_R = np.array(A_Clay1_P4_R)
80 A_Clay1_P5_R = np.array(A_Clay1_P5_R)
81
82 A_Clay2_P1_R = np.array(A_Clay2_P1_R)
83 A_Clay2_P2_R = np.array(A_Clay2_P2_R)
84 A_Clay2_P3_R = np.array(A_Clay2_P3_R)
85 A_Clay2_P4_R = np.array(A_Clay2_P4_R)
86 A_Clay2_P5_R = np.array(A_Clay2_P5_R)
87
88 #Centroides
89 Depth_C2_P1 = np.array(Depth_C2_P1) * -1
90 Depth_C1_P1 = np.array(Depth_C1_P1) * -1
91 Depth_C2_P1_Reduced = np.array(Depth_C2_P1_Reduced) * -1
92 Depth_C1_P1_Reduced = np.array(Depth_C1_P1_Reduced) * -1
93 Depth_C2_P2 = np.array(Depth_C2_P2) * -1
94 Depth_C1_P2 = np.array(Depth_C1_P2) * -1
95 Depth_C2_P2_Reduced = np.array(Depth_C2_P2_Reduced) * -1
96 Depth_C1_P2_Reduced = np.array(Depth_C1_P2_Reduced) * -1
97 Depth_C2_P3 = np.array(Depth_C2_P3) * -1
98 Depth_C1_P3 = np.array(Depth_C1_P3) * -1
99 Depth_C2_P3_Reduced = np.array(Depth_C2_P3_Reduced) * -1
100 Depth_C1_P3_Reduced = np.array(Depth_C1_P3_Reduced) * -1
101 Depth_C2_P4 = np.array(Depth_C2_P4) * -1
102 Depth_C1_P4 = np.array(Depth_C1_P4) * -1
103 Depth_C2_P4_Reduced = np.array(Depth_C2_P4_Reduced) * -1
104 Depth_C1_P4_Reduced = np.array(Depth_C1_P4_Reduced) * -1
105 Depth_C2_P5 = np.array(Depth_C2_P5) * -1
106 Depth_C1_P5 = np.array(Depth_C1_P5) * -1
107 Depth_C2_P5_Reduced = np.array(Depth_C2_P5_Reduced) * -1
```

```
Depth_C1_P5_Reduced = np.array(Depth_C1_P5_Reduced) * -1

"Soil Properties"

gamma_filling = 10
gamma_clay = 6
def c_clay1 (z):
    c = 23 + 0.7*z
    return c
def c_clay2 (z):
    c = 32 + 0.7*z
    return c

" Fixing the unregularities from data "
"_________________Pile 1_________________"

" For Clay 2 Reduced Volumes "

V_Clay2_P1_R = V_Clay2_P1_R[::-1] #Fliping it (Top cell closser to
    surface)

" For Clay 2 Reduced Areas "

A_Clay2_P1_R = A_Clay2_P1_R[::-1] #Fliping it (Top cell closser to
    surface)

" For Clay 1 Reduced Volumes "

Clay1R = V_Clay1_P1_R
V_Clay1_P1_R = np.zeros([67-28,1]) #New size, after merging the splited
    wedges into one

j = 0

for i in range (0, 56, 2):

    Suma = Clay1R[i] + Clay1R[i+1]
    Suma = Suma.item()

    V_Clay1_P1_R [j] = Suma

    j = j + 1

V_Clay1_P1_R[28:,0] = Clay1R[56:]
V_Clay1_P1_R = V_Clay1_P1_R[::-1] #Fliping it back again (Top cell
    closser to surface)

" For Clay 1 Reduced Areas "

Clay1R = A_Clay1_P1_R
A_Clay1_P1_R = np.zeros([66-28,1]) #New size, after merging the splited
    wedges into one
Points = Depth_C1_P1_Reduced
Depth_C1_P1_Reduced = np.zeros_like(A_Clay1_P1_R)

j = 0

```

```
for i in range (0, 56, 2):

    Suma = Clay1R[i] + Clay1R[i+1]
    Suma = Suma.item()

    A_Clay1_P1_R [j] = Suma
    Depth_C1_P1_Reduced[j] = Points[i]

    j = j + 1

A_Clay1_P1_R[28:,0] = Clay1R[56:]
Depth_C1_P1_Reduced[28:,0] = Points[56:]
A_Clay1_P1_R = A_Clay1_P1_R[::-1] #Fliping it back again (Top cell
    closser to surface)
Depth_C1_P1_Reduced = Depth_C1_P1_Reduced[::-1]

" For Fill Reduced Volumes "

FillR = V_Fill_P1_R
V_Fill_P1_R = np.zeros([97-28,1]) #New size, after merging the splited
    wedges into one

j = 0

for i in range (0, 56, 2):

    Suma = FillR[i] + FillR[i+1]
    Suma = Suma.item()

    V_Fill_P1_R [j] = Suma

    j = j + 1

V_Fill_P1_R[28:,0] = FillR[56:]
V_Fill_P1_R = V_Fill_P1_R[::-1] #Fliping it back again (Top cell closser
    to surface)


" For Fill Reduced Areas "

FillR = A_Fill_P1_R
A_Fill_P1_R = np.zeros([97-28,1]) #New size, after merging the splited
    wedges into one

j = 0

for i in range (0, 56, 2):

    Suma = FillR[i] + FillR[i+1]
    Suma = Suma.item()

    A_Fill_P1_R [j] = Suma

    j = j + 1

A_Fill_P1_R[28:,0] = FillR[56:]
```

```
A_Fill_P1_R = A_Fill_P1_R[::-1] #Fliping it back again (Top cell closser to surface)


"_________________Pile 2_________________"

" For Clay 2 Reduced Volumes "

V_Clay2_P2_R = V_Clay2_P2_R[::-1] #Fliping it (Top cell closser to surface)

" For Clay 2 Reduced Areas "

A_Clay2_P2_R = A_Clay2_P2_R[::-1] #Fliping it (Top cell closser to surface)

" For Clay 1 Reduced Volumes "

Clay1R = V_Clay1_P2_R
V_Clay1_P2_R = np.zeros([19-4,1]) #New size, after merging the splited wedges into one


j = 0

for i in range (0, 8, 2):

    Suma = Clay1R[i] + Clay1R[i+1]
    Suma = Suma.item()

    V_Clay1_P1_R [j] = Suma

    j = j + 1

V_Clay1_P2_R[4:,0] = Clay1R[8:]
V_Clay1_P2_R = V_Clay1_P2_R[::-1] #Fliping it back again (Top cell closser to surface)

" For Clay 1 Reduced Areas "

Clay1R = A_Clay1_P2_R
A_Clay1_P2_R = np.zeros([20-5,1]) #New size, after merging the splited wedges into one
Points = Depth_C1_P2_Reduced
Depth_C1_P2_Reduced = np.zeros_like(A_Clay1_P2_R)

j = 0

for i in range (0, 10, 2):

    Suma = Clay1R[i] + Clay1R[i+1]
    Suma = Suma.item()

    A_Clay1_P2_R [j] = Suma
    Depth_C1_P2_Reduced[j] = Points[i]

    j = j + 1
```

```
263
264 A_Clay1_P2_R[5:,0] = Clay1R[10:]
265 Depth_C1_P2_Reduced[5:,0] = Points[10:]
266 A_Clay1_P2_R = A_Clay1_P2_R[::-1] #Fliping it back again (Top cell
    closser to surface)
267 Depth_C1_P2_Reduced = Depth_C1_P2_Reduced[::-1]
268
269 " For Fill Reduced Volumes "
270
271 FillR = V_Fill_P2_R
272 V_Fill_P2_R = np.zeros([61-10,1]) #New size, after merging the splited
    wedges into one
273
274 j = 0
275
276 for i in range (0, 20, 2):
277
278     Suma = FillR[i] + FillR[i+1]
279     Suma = Suma.item()
280
281     V_Fill_P2_R [j] = Suma
282
283     j = j + 1
284
285 V_Fill_P2_R[10:,0] = FillR[20:]
286 V_Fill_P2_R = V_Fill_P2_R[::-1] #Fliping it back again (Top cell closser
    to surface)
287
288 " For Fill Reduced Areas "
289
290 FillR = A_Fill_P2_R
291 A_Fill_P2_R = np.zeros([63-11,1]) #New size, after merging the splited
    wedges into one
292
293 j = 0
294
295 for i in range (0, 22, 2):
296
297     Suma = FillR[i] + FillR[i+1]
298     Suma = Suma.item()
299
300     A_Fill_P2_R [j] = Suma
301
302     j = j + 1
303
304 A_Fill_P2_R[11:,0] = FillR[22:]
305 A_Fill_P2_R = A_Fill_P2_R[::-1] #Fliping it back again (Top cell closser
    to surface)
306
307 "_________________Pile 3_________________"
308
309 " For Clay 2 Reduced Volumes "
310
311 V_Clay2_P3_R = V_Clay2_P3_R[::-1] #Fliping it (Top cell closser to
    surface)
312
313 " For Clay 2 Reduced Areas "
```

```
314
315 A_Clay2_P3_R = A_Clay2_P3_R[::-1] #Fliping it (Top cell closser to
    surface)
316
317 " For Clay 1 Reduced Volumes "
318
319 Clay1R = V_Clay1_P3_R
320 V_Clay1_P3_R = np.zeros([19-4,1]) #New size, after merging the splited
    wedges into one
321
322 j = 0
323
324 for i in range (0, 8, 2):
325
326     Suma = Clay1R[i] + Clay1R[i+1]
327     Suma = Suma.item()
328
329     V_Clay1_P1_R [j] = Suma
330
331     j = j + 1
332
333 V_Clay1_P3_R[4:,0] = Clay1R[8:]
334 V_Clay1_P3_R = V_Clay1_P3_R[::-1] #Fliping it back again (Top cell
    closser to surface)
335
336 " For Clay 1 Reduced Areas "
337
338 Clay1R = A_Clay1_P3_R
339 A_Clay1_P3_R = np.zeros([20-5,1]) #New size, after merging the splited
    wedges into one
340 Points = Depth_C1_P3_Reduced
341 Depth_C1_P3_Reduced = np.zeros_like(A_Clay1_P3_R)
342
343
344 j = 0
345
346 for i in range (0, 10, 2):
347
348     Suma = Clay1R[i] + Clay1R[i+1]
349     Suma = Suma.item()
350
351     A_Clay1_P3_R [j] = Suma
352     Depth_C1_P3_Reduced[j] = Points[i]
353
354     j = j + 1
355
356 A_Clay1_P3_R[5:,0] = Clay1R[10:]
357 Depth_C1_P3_Reduced[5:,0] = Points[10:]
358 A_Clay1_P3_R = A_Clay1_P3_R[::-1] #Fliping it back again (Top cell
    closser to surface)
359 Depth_C1_P3_Reduced = Depth_C1_P3_Reduced[::-1]
360
361 " For Fill Reduced Volumes "
362
363 FillR = V_Fill_P3_R
364 V_Fill_P3_R = np.zeros([61-10,1]) #New size, after merging the splited
    wedges into one
```

```
365
366 j = 0
367
368 for i in range (0, 20, 2):
369
370     Suma = FillR[i] + FillR[i+1]
371     Suma = Suma.item()
372
373     V_Fill_P3_R [j] = Suma
374
375     j = j + 1
376
377 V_Fill_P3_R[10:,0] = FillR[20:]
378 V_Fill_P3_R = V_Fill_P3_R[::-1] #Fliping it back again (Top cell closser
    to surface)
379
380 " For Fill Reduced Areas "
381
382 FillR = A_Fill_P3_R
383 A_Fill_P3_R = np.zeros([63-11,1]) #New size, after merging the splited
    wedges into one
384
385 j = 0
386
387 for i in range (1, 22, 2):
388     Suma = FillR[i] + FillR[i+1]
389     Suma = Suma.item()
390
391     A_Fill_P3_R [j] = Suma
392
393     j = j + 1
394
395 A_Fill_P3_R[11:50,0] = FillR[21:60]
396 A_Fill_P3_R[50,0] = FillR[60]
397 A_Fill_P3_R[51,0] = FillR[0]
398 A_Fill_P3_R = A_Fill_P3_R[::-1] #Fliping it back again (Top cell closser
    to surface)
399
400 "_________________Pile 4_________________"
401
402 " For Clay 2 Reduced Volumes "
403
404 V_Clay2_P4_R = V_Clay2_P4_R[::-1] #Fliping it (Top cell closser to
    surface)
405
406 " For Clay 2 Reduced Areas "
407
408 A_Clay2_P4_R = A_Clay2_P4_R[::-1] #Fliping it (Top cell closser to
    surface)
409
410 " For Clay 1 Reduced Volumes "
411
412 Clay1R = V_Clay1_P4_R
413 V_Clay1_P4_R = np.zeros([44-17,1]) #New size, after merging the splited
    wedges into one
414
415 j = 0
```

```
416
417 for i in range (0, 34, 2):
418
419     Suma = Clay1R[i] + Clay1R[i+1]
420     Suma = Suma.item()
421
422     V_Clay1_P4_R [j] = Suma
423
424     j = j + 1
425
426 V_Clay1_P4_R[17:,0] = Clay1R[34:]
427 V_Clay1_P4_R = V_Clay1_P4_R[::-1] #Fliping it back again (Top cell
    closser to surface)
428
429 " For Clay 1 Reduced Areas "
430
431 Clay1R = A_Clay1_P4_R
432 A_Clay1_P4_R = np.zeros([44-17,1]) #New size, after merging the splited
    wedges into one
433 Points = Depth_C1_P4_Reduced
434 Depth_C1_P4_Reduced = np.zeros_like(A_Clay1_P4_R)
435
436 j = 0
437
438 for i in range (0, 34, 2):
439
440     Suma = Clay1R[i] + Clay1R[i+1]
441     Suma = Suma.item()
442
443     A_Clay1_P4_R [j] = Suma
444     Depth_C1_P4_Reduced[j] = Points[i]
445
446     j = j + 1
447
448 A_Clay1_P4_R[17:,0] = Clay1R[34:]
449 Depth_C1_P4_Reduced[17:,0] = Points[34:]
450 A_Clay1_P4_R = A_Clay1_P4_R[::-1] #Fliping it back again (Top cell
    closser to surface)
451 Depth_C1_P4_Reduced = Depth_C1_P4_Reduced[::-1]
452
453 " For Fill Reduced Volumes "
454
455 FillR = V_Fill_P4_R
456 V_Fill_P4_R = np.zeros([75-17,1]) #New size, after merging the splited
    wedges into one
457
458 j = 0
459
460 for i in range (0, 34, 2):
461
462     Suma = FillR[i] + FillR[i+1]
463     Suma = Suma.item()
464
465     V_Fill_P4_R [j] = Suma
466
467     j = j + 1
468
```

```
V_Fill_P4_R[17:,0] = FillR[34:]
V_Fill_P4_R = V_Fill_P4_R[::-1] #Fliping it back again (Top cell closser to surface)

" For Fill Reduced Areas "

FillR = A_Fill_P4_R
A_Fill_P4_R = np.zeros([74-17,1]) #New size, after merging the splited wedges into one

j = 0

for i in range (1, 34, 2):
    Suma = FillR[i] + FillR[i+1]
    Suma = Suma.item()

    A_Fill_P4_R [j] = Suma

    j = j + 1

A_Fill_P4_R[17:55,0] = FillR[34:72]
A_Fill_P4_R[56,0] = FillR[0]
A_Fill_P4_R[55,0] = FillR[72]
A_Fill_P4_R = A_Fill_P4_R[::-1] #Fliping it back again (Top cell closser to surface)

"______Obtaining reduction factors______"

" For un-reducted Pile 1  "

ndz = 119
Phi_w_P1 = np.zeros([ndz,1])
Phi_c_P1 = np.zeros([ndz,1])

WP1_UR = np.zeros([ndz,1])
AP1_UR = np.zeros([ndz,1])

#Clay 2
WP1_UR[(ndz - len(V_Clay2_P1_UR )):,0] = V_Clay2_P1_UR[::-1] * gamma_clay
AP1_UR[(ndz - len(A_Clay2_P1_UR )):,0] = A_Clay2_P1_UR[::-1] * c_clay2(Depth_C2_P1[::-1])

#Clay 1
WP1_UR[(ndz - len(V_Clay1_P1_UR )):,0] += V_Clay1_P1_UR[::-1] * gamma_clay
AP1_UR[(ndz - len(A_Clay1_P1_UR )):,0] += A_Clay1_P1_UR[::-1] * c_clay1(Depth_C1_P1[::-1])

#Filling
WP1_UR[(ndz - len(V_Fill_P1_UR )):,0] += V_Fill_P1_UR[::-1] * gamma_filling


" For reducted Pile 1  "

ndz = 119
WP1_R = np.zeros([ndz,1])
```

```
AP1_R = np.zeros([ndz,1])

#Clay 2
WP1_R[(ndz - len(V_Clay2_P1_R )):,0] = V_Clay2_P1_R * gamma_clay
AP1_R[(ndz - len(A_Clay2_P1_R )):,0] = A_Clay2_P1_R * c_clay2(Depth_C2_P1
    [::-1])

#Clay 1
Clay1R = np.zeros_like( V_Clay1_P1_UR )
Clay1R[:len( V_Clay1_P1_R )] = V_Clay1_P1_R [:,0]

WP1_R[(ndz - len(Clay1R)):,0] += Clay1R[:] * gamma_clay

Clay1R = np.zeros_like( A_Clay1_P1_UR )

Clay1 = A_Clay1_P1_R * c_clay1(Depth_C1_P1[::-1])
Clay1R[:len( A_Clay1_P1_R )] = Clay1[:,0]

AP1_R[(ndz - len(Clay1R )):,0] += Clay1R[:]

#Filling

Filling = np.zeros_like( V_Fill_P1_UR)
Filling[:len( V_Fill_P1_R)] = V_Fill_P1_R [:,0]
WP1_R[(ndz - len(Filling )):,0] += Filling[:]  * gamma_filling

for i in range(ndz):
    Phi_w_P1[i,0] = WP1_R[i] / WP1_UR[i]
    Phi_c_P1[i,0] = AP1_R[i] / AP1_UR[i]
    if np.isnan(Phi_c_P1[i,0]):
        Phi_c_P1[i,0] = 0
    if np.isnan(Phi_w_P1[i,0]):
        Phi_w_P1[i,0] = 0

" For un-reducted Pile 2  "

ndz = 51
Phi_w_P2 = np.zeros([ndz,1])
Phi_c_P2 = np.zeros([ndz,1])

WP2_UR = np.zeros([ndz,1])
AP2_UR = np.zeros([ndz,1])

#Clay 2
WP2_UR[(ndz - len(V_Clay2_P2_UR )):,0] = V_Clay2_P2_UR[::-1] * gamma_clay
AP2_UR[(ndz - len(A_Clay2_P2_UR )):,0] = A_Clay2_P2_UR[::-1] * c_clay2(
    Depth_C2_P2[::-1])

#Clay 1
WP2_UR[(ndz - len(V_Clay1_P2_UR )):,0] += V_Clay1_P2_UR[::-1] *
    gamma_clay
AP2_UR[(ndz - len(A_Clay1_P2_UR )):,0] += A_Clay1_P2_UR[::-1] * c_clay1(
    Depth_C1_P2[::-1])

#Filling
WP2_UR[(ndz - len(V_Fill_P2_UR )):,0] += V_Fill_P2_UR[::-1] *
    gamma_filling
```

```
571
572 " For reducted Pile 2  "
573
574 ndz = 51
575
576 WP2_R = np.zeros([ndz,1])
577 AP2_R = np.zeros([ndz,1])
578
579 #Clay 2
580 WP2_R[(ndz - len(V_Clay2_P2_R )):,0] = V_Clay2_P2_R * gamma_clay
581 AP2_R[(ndz - len(A_Clay2_P2_R )):,0] = A_Clay2_P2_R * c_clay2(
    Depth_C2_P2_Reduced[::-1])
582
583 #Clay 1
584 Clay1R = np.zeros_like( V_Clay1_P2_UR )
585 Clay1R[:len( V_Clay1_P2_R )] = V_Clay1_P2_R [:,0]
586
587 WP2_R[(ndz - len(Clay1R)):,0] += Clay1R[:] * gamma_clay
588
589 Clay1R = np.zeros_like( A_Clay1_P2_UR )
590
591 Clay1 = A_Clay1_P2_R * c_clay1(Depth_C1_P2_Reduced[::-1])
592 Clay1R[:len( A_Clay1_P2_R )] = Clay1[:,0]
593
594 AP2_R[(ndz - len(Clay1R )):,0] += Clay1R[:]
595
596 #Filling
597
598 Filling = np.zeros_like( V_Fill_P2_UR)
599 Filling[:len( V_Fill_P2_R)] = V_Fill_P2_R [:,0]
600 WP2_R[(ndz - len(Filling )):,0] += Filling[:]  * gamma_filling
601
602 for i in range(ndz):
603     Phi_w_P2[i,0] = WP2_R[i] / WP2_UR[i]
604     Phi_c_P2[i,0] = AP2_R[i] / AP2_UR[i]
605     if np.isnan(Phi_c_P2[i,0]):
606         Phi_c_P2[i,0] = 0
607     if np.isnan(Phi_w_P2[i,0]):
608         Phi_w_P2[i,0] = 0
609
610 " For un-reducted Pile 3  "
611
612 ndz = 51
613 Phi_w_P3 = np.zeros([ndz,1])
614 Phi_c_P3 = np.zeros([ndz,1])
615
616 WP3_UR = np.zeros([ndz,1])
617 AP3_UR = np.zeros([ndz,1])
618
619 #Clay 2
620 WP3_UR[(ndz - len(V_Clay2_P3_UR )):,0] = V_Clay2_P3_UR[::-1] * gamma_clay
621 AP3_UR[(ndz - len(A_Clay2_P3_UR )):,0] = A_Clay2_P3_UR[::-1] * c_clay2(
    Depth_C2_P3[::-1])
622
623 #Clay 1
624 WP3_UR[(ndz - len(V_Clay1_P3_UR )):,0] += V_Clay1_P3_UR[::-1] *
    gamma_clay
```

```
AP3_UR[(ndz - len(A_Clay1_P3_UR )):,0] += A_Clay1_P3_UR[::-1] * c_clay1(
    Depth_C1_P3[::-1])

#Filling
WP3_UR[(ndz - len(V_Fill_P3_UR )):,0] += V_Fill_P3_UR[::-1] *
    gamma_filling

" For reducted Pile 3  "

ndz = 51

WP3_R = np.zeros([ndz,1])
AP3_R = np.zeros([ndz,1])

#Clay 2
WP3_R[(ndz - len(V_Clay2_P3_R )):,0] = V_Clay2_P3_R * gamma_clay
AP3_R[(ndz - len(A_Clay2_P3_R )):,0] = A_Clay2_P3_R * c_clay2(
    Depth_C2_P3_Reduced[::-1])

#Clay 1

Clay1R = np.zeros_like( V_Clay1_P3_UR )
Clay1R[:len( V_Clay1_P3_R )] = V_Clay1_P3_R [:,0]

WP3_R[(ndz - len(Clay1R)):,0] += Clay1R[:] * gamma_clay

Clay1R = np.zeros_like( A_Clay1_P3_UR )

Clay1 = A_Clay1_P3_R * c_clay1(Depth_C1_P3_Reduced[::-1])
Clay1R[:len( A_Clay1_P3_R )] = Clay1[:,0]

AP3_R[(ndz - len(Clay1R )):,0] += Clay1R[:]

#Filling
Filling = np.zeros_like( V_Fill_P3_UR)
Filling[:len( V_Fill_P3_R)] = V_Fill_P3_R [:,0]
WP3_R[(ndz - len(Filling )):,0] += Filling[:]  * gamma_filling

for i in range(ndz):
    Phi_w_P3[i,0] = WP3_R[i] / WP3_UR[i]
    Phi_c_P3[i,0] = AP3_R[i] / AP3_UR[i]
    if np.isnan(Phi_c_P3[i,0]):
        Phi_c_P3[i,0] = 0
    if np.isnan(Phi_w_P3[i,0]):
        Phi_w_P3[i,0] = 0

" For un-reducted Pile 4  "

ndz = 119
Phi_w_P4 = np.zeros([ndz,1])
Phi_c_P4 = np.zeros([ndz,1])

WP4_UR = np.zeros([ndz,1])
AP4_UR = np.zeros([ndz,1])

#Clay 2
WP4_UR[(ndz - len(V_Clay2_P4_UR )):,0] = V_Clay2_P4_UR[::-1] * gamma_clay
```

```
AP4_UR[(ndz - len(A_Clay2_P4_UR )):,0] = A_Clay2_P4_UR[::-1] * c_clay2(
    Depth_C2_P4[::-1])

#Clay 1
WP4_UR[(ndz - len(V_Clay1_P4_UR )):,0] += V_Clay1_P4_UR[::-1] *
    gamma_clay
AP4_UR[(ndz - len(A_Clay1_P4_UR )):,0] += A_Clay1_P4_UR[::-1] * c_clay1(
    Depth_C1_P4[::-1])

#Filling
WP4_UR[(ndz - len(V_Fill_P4_UR )):,0] += V_Fill_P4_UR[::-1] *
    gamma_filling

" For reducted Pile 4  "

ndz = 119

WP4_R = np.zeros([ndz,1])
AP4_R = np.zeros([ndz,1])

#Clay 2
WP4_R[(ndz - len(V_Clay2_P4_R )):,0] = V_Clay2_P4_R * gamma_clay
AP4_R[(ndz - len(A_Clay2_P4_R )):,0] = A_Clay2_P4_R * c_clay2(
    Depth_C2_P4_Reduced[::-1])

#Clay 1
Clay1R = np.zeros_like( V_Clay1_P4_UR )
Clay1R[:len( V_Clay1_P4_R )] = V_Clay1_P4_R [:,0]

WP4_R[(ndz - len(Clay1R)):,0] += Clay1R[:] * gamma_clay

Clay1R = np.zeros_like( A_Clay1_P4_UR )

Clay1 = A_Clay1_P4_R * c_clay1(Depth_C1_P4_Reduced[::-1])
Clay1R[:len( A_Clay1_P4_R )] = Clay1[:,0]

AP4_R[(ndz - len(Clay1R )):,0] += Clay1R[:]

#Filling
Filling = np.zeros_like( V_Fill_P4_UR)
Filling[:len( V_Fill_P4_R)] = V_Fill_P4_R [:,0]
WP4_R[(ndz - len(Filling )):,0] += Filling[:]  * gamma_filling

for i in range(ndz):
    Phi_w_P4[i,0] = WP4_R[i] / WP4_UR[i]
    Phi_c_P4[i,0] = AP4_R[i] / AP4_UR[i]
    if np.isnan(Phi_c_P4[i,0]):
        Phi_c_P4[i,0] = 0
    if np.isnan(Phi_w_P4[i,0]):
        Phi_w_P4[i,0] = 0

" For un-reducted Pile 5  "

ndz = 119
Phi_w_P5 = np.zeros([ndz,1])
Phi_c_P5 = np.zeros([ndz,1])

```

```
731 WP5_UR = np.zeros([ndz,1])
732 AP5_UR = np.zeros([ndz,1])
733
734 #Clay 2
735 WP5_UR[(ndz - len(V_Clay2_P5_UR )):,0] = V_Clay2_P5_UR[::-1] * gamma_clay
736 AP5_UR[(ndz - len(A_Clay2_P5_UR )):,0] = A_Clay2_P5_UR[::-1] * c_clay2(
        Depth_C2_P5[::-1])
737
738 #Clay 1
739 WP5_UR[(ndz - len(V_Clay1_P5_UR )):,0] += V_Clay1_P5_UR[::-1] *
        gamma_clay
740 AP5_UR[(ndz - len(A_Clay1_P5_UR )):,0] += A_Clay1_P5_UR[::-1] * c_clay1(
        Depth_C1_P5[::-1])
741
742 #Filling
743 WP5_UR[(ndz - len(V_Fill_P5_UR )):,0] += V_Fill_P5_UR[::-1] *
        gamma_filling
744
745
746
747 " For reducted Pile 5  "
748
749 ndz = 119
750
751 WP5_R = np.zeros([ndz,1])
752 AP5_R = np.zeros([ndz,1])
753
754 #Clay 2
755 WP5_R[(ndz - len(V_Clay2_P5_R )):,0] = V_Clay2_P5_R[::-1] * gamma_clay
756 AP5_R[(ndz - len(A_Clay2_P5_R )):,0] = A_Clay2_P5_R[::-1] * c_clay2(
        Depth_C2_P5_Reduced[::-1])
757
758 #Clay 1
759 Clay1R = np.zeros_like( V_Clay1_P5_UR )
760 Clay1R[:len( V_Clay1_P5_R )] = V_Clay1_P5_R[::-1]
761
762 WP5_R[(ndz - len(Clay1R)):,0] += Clay1R * gamma_clay
763
764 Clay1R = np.zeros_like( A_Clay1_P5_UR )
765
766 Clay1 = A_Clay1_P5_R[::-1] * c_clay1(Depth_C1_P5_Reduced[::-1])
767 Clay1R[:len( A_Clay1_P5_R )] = Clay1
768
769 AP5_R[(ndz - len(Clay1R )):,0] += Clay1R[:]
770
771 #Filling
772 Filling = np.zeros_like( V_Fill_P5_UR)
773 Filling[:len( V_Fill_P5_R)] = V_Fill_P5_R [::-1]
774 WP5_R[(ndz - len(Filling )):,0] += Filling[:]  * gamma_filling
775
776
777 for i in range(ndz):
778     Phi_w_P5[i,0] = WP5_R[i] / WP5_UR[i]
779     Phi_c_P5[i,0] = AP5_R[i] / AP5_UR[i]
780     if np.isnan(Phi_c_P5[i,0]):
781         Phi_c_P5[i,0] = 0
782     if np.isnan(Phi_w_P5[i,0]):
```

```
        Phi_w_P5[i,0] = 0


"______TO EXPORT RESULTS______"

" For Pile 1"
# Correct file path
file_path = r"C:\Users\SE1G4E\OneDrive - Chalmers\MSC\Python\Gothenburg
    Reduction Factors\Phi_w_P1.csv"

# Make sure the folder exists
folder = os.path.dirname(file_path)
if not os.path.exists(folder):
    os.makedirs(folder)

# Write CSV
with open(file_path, 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    for row in Phi_w_P1:
        writer.writerow(row)

# Correct file path
file_path = r"C:\Users\SE1G4E\OneDrive - Chalmers\MSC\Python\Gothenburg
    Reduction Factors\Phi_c_P1.csv"

# Make sure the folder exists
folder = os.path.dirname(file_path)
if not os.path.exists(folder):
    os.makedirs(folder)

# Write CSV
with open(file_path, 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    for row in Phi_c_P1:
        writer.writerow(row)



" For Pile 2"
# Correct file path
file_path = r"C:\Users\SE1G4E\OneDrive - Chalmers\MSC\Python\Gothenburg
    Reduction Factors\Phi_w_P2.csv"

# Make sure the folder exists
folder = os.path.dirname(file_path)
if not os.path.exists(folder):
    os.makedirs(folder)

# Write CSV
with open(file_path, 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    for row in Phi_w_P2:
        writer.writerow(row)

# Correct file path
file_path = r"C:\Users\SE1G4E\OneDrive - Chalmers\MSC\Python\Gothenburg
    Reduction Factors\Phi_c_P2.csv"
```

```
836
837 # Make sure the folder exists
838 folder = os.path.dirname(file_path)
839 if not os.path.exists(folder):
840     os.makedirs(folder)
841
842 # Write CSV
843 with open(file_path, 'w', newline='') as csvfile:
844     writer = csv.writer(csvfile)
845     for row in Phi_c_P2:
846         writer.writerow(row)
847
848
849
850
851 " For Pile 3"
852 # Correct file path
853 file_path = r"C:\Users\SE1G4E\OneDrive - Chalmers\MSC\Python\Gothenburg
    Reduction Factors\Phi_w_P3.csv"
854
855 # Make sure the folder exists
856 folder = os.path.dirname(file_path)
857 if not os.path.exists(folder):
858     os.makedirs(folder)
859
860 # Write CSV
861 with open(file_path, 'w', newline='') as csvfile:
862     writer = csv.writer(csvfile)
863     for row in Phi_w_P3:
864         writer.writerow(row)
865
866 # Correct file path
867 file_path = r"C:\Users\SE1G4E\OneDrive - Chalmers\MSC\Python\Gothenburg
    Reduction Factors\Phi_c_P3.csv"
868
869 # Make sure the folder exists
870 folder = os.path.dirname(file_path)
871 if not os.path.exists(folder):
872     os.makedirs(folder)
873
874 # Write CSV
875 with open(file_path, 'w', newline='') as csvfile:
876     writer = csv.writer(csvfile)
877     for row in Phi_c_P3:
878         writer.writerow(row)
879
880
881
882 " For Pile 4"
883 # Correct file path
884 file_path = r"C:\Users\SE1G4E\OneDrive - Chalmers\MSC\Python\Gothenburg
    Reduction Factors\Phi_w_P4.csv"
885
886 # Make sure the folder exists
887 folder = os.path.dirname(file_path)
888 if not os.path.exists(folder):
889     os.makedirs(folder)
```

```
890
891 # Write CSV
892 with open(file_path, 'w', newline='') as csvfile:
893     writer = csv.writer(csvfile)
894     for row in Phi_w_P4:
895         writer.writerow(row)
896
897 # Correct file path
898 file_path = r"C:\Users\SE1G4E\OneDrive - Chalmers\MSC\Python\Gothenburg
        Reduction Factors\Phi_c_P4.csv"
899
900 # Make sure the folder exists
901 folder = os.path.dirname(file_path)
902 if not os.path.exists(folder):
903     os.makedirs(folder)
904
905 # Write CSV
906 with open(file_path, 'w', newline='') as csvfile:
907     writer = csv.writer(csvfile)
908     for row in Phi_c_P4:
909         writer.writerow(row)
910
911
912
913 " For Pile 5"
914 # Correct file path
915 file_path = r"C:\Users\SE1G4E\OneDrive - Chalmers\MSC\Python\Gothenburg
        Reduction Factors\Phi_w_P5.csv"
916
917 # Make sure the folder exists
918 folder = os.path.dirname(file_path)
919 if not os.path.exists(folder):
920     os.makedirs(folder)
921
922 # Write CSV
923 with open(file_path, 'w', newline='') as csvfile:
924     writer = csv.writer(csvfile)
925     for row in Phi_w_P5:
926         writer.writerow(row)
927
928 # Correct file path
929 file_path = r"C:\Users\SE1G4E\OneDrive - Chalmers\MSC\Python\Gothenburg
        Reduction Factors\Phi_c_P5.csv"
930
931 # Make sure the folder exists
932 folder = os.path.dirname(file_path)
933 if not os.path.exists(folder):
934     os.makedirs(folder)
935
936 # Write CSV
937 with open(file_path, 'w', newline='') as csvfile:
938     writer = csv.writer(csvfile)
939     for row in Phi_c_P5:
940         writer.writerow(row)
```

The python code utilized to obtain the reduction factor for the Gothenburg Inclined model is shown below. It is worth mentioning that this code is written in Python prompt built in Grasshopper.

```
# -*- coding: utf-8 -*-
"""
Created on Tue May 20 15:58:34 2025

@author: SE1G4E
"""

import numpy as np
import ghpythonlib.treehelpers as th
import csv
import os

V_Fill_P1_UR = np.array(V_Fill_P1_UR)
V_Fill_P2_UR = np.array(V_Fill_P2_UR)
V_Fill_P3_UR = np.array(V_Fill_P3_UR)
V_Fill_P4_UR = np.array(V_Fill_P4_UR)
V_Fill_P5_UR = np.array(V_Fill_P5_UR)

V_Clay1_P1_UR = np.array(V_Clay1_P1_UR)
V_Clay1_P2_UR = np.array(V_Clay1_P2_UR)
V_Clay1_P3_UR = np.array(V_Clay1_P3_UR)
V_Clay1_P4_UR = np.array(V_Clay1_P4_UR)
V_Clay1_P5_UR = np.array(V_Clay1_P5_UR)

V_Clay2_P1_UR = np.array(V_Clay2_P1_UR)
V_Clay2_P2_UR = np.array(V_Clay2_P2_UR)
V_Clay2_P3_UR = np.array(V_Clay2_P3_UR)
V_Clay2_P4_UR = np.array(V_Clay2_P4_UR)
V_Clay2_P5_UR = np.array(V_Clay2_P5_UR)

A_Fill_P1_UR = np.array(A_Fill_P1_UR)
A_Fill_P2_UR = np.array(A_Fill_P2_UR)
A_Fill_P3_UR = np.array(A_Fill_P3_UR)
A_Fill_P4_UR = np.array(A_Fill_P4_UR)
A_Fill_P5_UR = np.array(A_Fill_P5_UR)

A_Clay1_P1_UR = np.array(A_Clay1_P1_UR)
A_Clay1_P2_UR = np.array(A_Clay1_P2_UR)
A_Clay1_P3_UR = np.array(A_Clay1_P3_UR)
A_Clay1_P4_UR = np.array(A_Clay1_P4_UR)
A_Clay1_P5_UR = np.array(A_Clay1_P5_UR)

A_Clay2_P1_UR = np.array(A_Clay2_P1_UR)
A_Clay2_P2_UR = np.array(A_Clay2_P2_UR)
A_Clay2_P3_UR = np.array(A_Clay2_P3_UR)
A_Clay2_P4_UR = np.array(A_Clay2_P4_UR)
A_Clay2_P5_UR = np.array(A_Clay2_P5_UR)

V_Fill_P1_R = np.array(V_Fill_P1_R)
V_Fill_P2_R = np.array(V_Fill_P2_R)
V_Fill_P3_R = np.array(V_Fill_P3_R)
V_Fill_P4_R = np.array(V_Fill_P4_R)
```

```
V_Fill_P5_R = np.array(V_Fill_P5_R)

V_Clay1_P1_R = np.array(V_Clay1_P1_R)
V_Clay1_P2_R = np.array(V_Clay1_P2_R)
V_Clay1_P3_R = np.array(V_Clay1_P3_R)
V_Clay1_P4_R = np.array(V_Clay1_P4_R)
V_Clay1_P5_R = np.array(V_Clay1_P5_R)

V_Clay2_P1_R = np.array(V_Clay2_P1_R)
V_Clay2_P2_R = np.array(V_Clay2_P2_R)
V_Clay2_P3_R = np.array(V_Clay2_P3_R)
V_Clay2_P4_R = np.array(V_Clay2_P4_R)
V_Clay2_P5_R = np.array(V_Clay2_P5_R)

A_Fill_P1_R = np.array(A_Fill_P1_R)
A_Fill_P2_R = np.array(A_Fill_P2_R)
A_Fill_P3_R = np.array(A_Fill_P3_R)
A_Fill_P4_R = np.array(A_Fill_P4_R)
A_Fill_P5_R = np.array(A_Fill_P5_R)

A_Clay1_P1_R = np.array(A_Clay1_P1_R)
A_Clay1_P2_R = np.array(A_Clay1_P2_R)
A_Clay1_P3_R = np.array(A_Clay1_P3_R)
A_Clay1_P4_R = np.array(A_Clay1_P4_R)
A_Clay1_P5_R = np.array(A_Clay1_P5_R)

A_Clay2_P1_R = np.array(A_Clay2_P1_R)
A_Clay2_P2_R = np.array(A_Clay2_P2_R)
A_Clay2_P3_R = np.array(A_Clay2_P3_R)
A_Clay2_P4_R = np.array(A_Clay2_P4_R)
A_Clay2_P5_R = np.array(A_Clay2_P5_R)

#Centroides
Depth_C2_P1 = np.array(Depth_C2_P1) * -1
Depth_C1_P1 = np.array(Depth_C1_P1) * -1
Depth_C2_P1_Reduced = np.array(Depth_C2_P1_Reduced) * -1
Depth_C1_P1_Reduced = np.array(Depth_C1_P1_Reduced) * -1
Depth_C2_P2 = np.array(Depth_C2_P2) * -1
Depth_C1_P2 = np.array(Depth_C1_P2) * -1
Depth_C2_P2_Reduced = np.array(Depth_C2_P2_Reduced) * -1
Depth_C1_P2_Reduced = np.array(Depth_C1_P2_Reduced) * -1
Depth_C2_P3 = np.array(Depth_C2_P3) * -1
Depth_C1_P3 = np.array(Depth_C1_P3) * -1
Depth_C2_P3_Reduced = np.array(Depth_C2_P3_Reduced) * -1
Depth_C1_P3_Reduced = np.array(Depth_C1_P3_Reduced) * -1
Depth_C2_P4 = np.array(Depth_C2_P4) * -1
Depth_C1_P4 = np.array(Depth_C1_P4) * -1
Depth_C2_P4_Reduced = np.array(Depth_C2_P4_Reduced) * -1
Depth_C1_P4_Reduced = np.array(Depth_C1_P4_Reduced) * -1
Depth_C2_P5 = np.array(Depth_C2_P5) * -1
Depth_C1_P5 = np.array(Depth_C1_P5) * -1
Depth_C2_P5_Reduced = np.array(Depth_C2_P5_Reduced) * -1
Depth_C1_P5_Reduced = np.array(Depth_C1_P5_Reduced) * -1

"Soil Properties"

gamma_filling = 10
```

```
gamma_clay = 6
def c_clay1 (z):
    c = 23 * z/z
    return c
def c_clay2 (z):
    c = 23 + 0.7*z
    return c

" Fixing the unregularities from data "
"_________________Pile 1_________________"

" For Clay 2 Reduced Volumes "

V_Clay2_P1_R = V_Clay2_P1_R[::-1] #Fliping it (Top cell closser to
    surface)

" For Clay 2 Reduced Areas "

A_Clay2_P1_R = A_Clay2_P1_R[::-1] #Fliping it (Top cell closser to
    surface)

" For Clay 1 Reduced Volumes "

Clay1R = V_Clay1_P1_R
V_Clay1_P1_R = np.zeros([68-(11+13),1]) #New size, after merging the
    splited wedges into one

j = 0

for i in range (0, 22, 2):

    Suma = Clay1R[i] + Clay1R[i+1]
    Suma = Suma.item()

    V_Clay1_P1_R [j] = Suma

    j = j + 1

V_Clay1_P1_R[j:14,0] = Clay1R[22:25]
j = 15

for i in range (25, 52, 2):

    Suma = Clay1R[i] + Clay1R[i+1]
    Suma = Suma.item()

    V_Clay1_P1_R [j] = Suma

    j = j + 1

V_Clay1_P1_R[28:,0] = Clay1R[53:]
V_Clay1_P1_R = V_Clay1_P1_R[::-1] #Fliping it (Top cell closser to
    surface)

" For Clay 1 Reduced Areas "

Clay1R = A_Clay1_P1_R
```

```
A_Clay1_P1_R = np.zeros([69-26,1]) #New size, after merging the splited wedges into one
Points = Depth_C1_P1_Reduced
Depth_C1_P1_Reduced = np.zeros_like(A_Clay1_P1_R)

j = 0

for i in range (0, 22, 2):

    Suma = Clay1R[i] + Clay1R[i+1]
    Suma = Suma.item()

    A_Clay1_P1_R [j] = Suma
    Depth_C1_P1_Reduced[j] = Points[i]

    j = j + 1

A_Clay1_P1_R[j:13,0] = Clay1R[22:24]
Depth_C1_P1_Reduced[j:13,0] = Points[22:24]

j = 13
for i in range (24, 54, 2):

    Suma = Clay1R[i] + Clay1R[i+1]
    Suma = Suma.item()

    A_Clay1_P1_R [j] = Suma
    Depth_C1_P1_Reduced[j] = Points[i]

    j = j + 1

A_Clay1_P1_R[28:,0] = Clay1R[54:]
Depth_C1_P1_Reduced[28:,0] = Points[54:]
A_Clay1_P1_R = A_Clay1_P1_R[::-1] #Fliping it back again (Top cell closser to surface)
Depth_C1_P1_Reduced = Depth_C1_P1_Reduced[::-1]

" For Fill Reduced Volumes "

FillR = V_Fill_P1_R
V_Fill_P1_R = np.zeros([82-28,1]) #New size, after merging the splited wedges into one

j = 0

for i in range (0, 56, 2):

    Suma = FillR[i] + FillR[i+1]
    Suma = Suma.item()

    V_Fill_P1_R [j] = Suma

    j = j + 1

V_Fill_P1_R[28:,0] = FillR[56:]
V_Fill_P1_R = V_Fill_P1_R[::-1] #Fliping it back again (Top cell closser to surface)
```

```
216
217
218 " For Fill Reduced Areas "
219
220 FillR = A_Fill_P1_R
221 A_Fill_P1_R = np.zeros([78-28,1]) #New size, after merging the splited
    wedges into one
222
223 j = 0
224
225 for i in range (0, 56, 2):
226
227     Suma = FillR[i] + FillR[i+1]
228     Suma = Suma.item()
229
230     A_Fill_P1_R [j] = Suma
231
232     j = j + 1
233
234 A_Fill_P1_R[28:,0] = FillR[56:]
235 A_Fill_P1_R = A_Fill_P1_R[::-1] #Fliping it back again (Top cell closser
    to surface)
236
237
238 "_________________Pile 2_________________"
239
240 " For Clay 2 Reduced Volumes "
241
242 V_Clay2_P2_R = V_Clay2_P2_R[::-1] #Fliping it (Top cell closser to
    surface)
243
244 " For Clay 2 Reduced Areas "
245
246 A_Clay2_P2_R = A_Clay2_P2_R[::-1] #Fliping it (Top cell closser to
    surface)
247
248 " For Clay 1 Reduced Volumes "
249
250 Clay1R = V_Clay1_P2_R
251 V_Clay1_P2_R = np.zeros([24-4,1]) #New size, after merging the splited
    wedges into one
252
253
254 j = 0
255
256 for i in range (0, 8, 2):
257
258     Suma = Clay1R[i] + Clay1R[i+1]
259     Suma = Suma.item()
260
261     V_Clay1_P1_R [j] = Suma
262
263     j = j + 1
264
265 V_Clay1_P2_R[4:,0] = Clay1R[8:]
266 V_Clay1_P2_R = V_Clay1_P2_R[::-1] #Fliping it back again (Top cell
    closser to surface)
```

```
267
268 " For Clay 1 Reduced Areas "
269
270 Clay1R = A_Clay1_P2_R
271 A_Clay1_P2_R = np.zeros([25-5,1]) #New size, after merging the splited
    wedges into one
272 Points = Depth_C1_P2_Reduced
273 Depth_C1_P2_Reduced = np.zeros_like(A_Clay1_P2_R)
274
275 j = 0
276
277 for i in range (0, 10, 2):
278
279     Suma = Clay1R[i] + Clay1R[i+1]
280     Suma = Suma.item()
281
282     A_Clay1_P2_R [j] = Suma
283     Depth_C1_P2_Reduced[j] = Points[i]
284
285     j = j + 1
286
287 A_Clay1_P2_R[5:,0] = Clay1R[10:]
288 Depth_C1_P2_Reduced[5:,0] = Points[10:]
289 A_Clay1_P2_R = A_Clay1_P2_R[::-1] #Fliping it back again (Top cell
    closser to surface)
290 Depth_C1_P2_Reduced = Depth_C1_P2_Reduced[::-1]
291
292 " For Fill Reduced Volumes "
293
294 FillR = V_Fill_P2_R
295 V_Fill_P2_R = np.zeros([66-15,1]) #New size, after merging the splited
    wedges into one
296
297 j = 0
298
299 for i in range (0, 30, 2):
300
301     Suma = FillR[i] + FillR[i+1]
302     Suma = Suma.item()
303
304     V_Fill_P2_R [j] = Suma
305
306     j = j + 1
307
308 V_Fill_P2_R[15:,0] = FillR[30:]
309 V_Fill_P2_R = V_Fill_P2_R[::-1] #Fliping it back again (Top cell closser
    to surface)
310
311 " For Fill Reduced Areas "
312
313 FillR = A_Fill_P2_R
314 A_Fill_P2_R = np.zeros([68-11,1]) #New size, after merging the splited
    wedges into one
315
316 j = 0
317
318 for i in range (0, 22, 2):
```

```
319
320     Suma = FillR[i] + FillR[i+1]
321     Suma = Suma.item()
322
323     A_Fill_P2_R [j] = Suma
324
325     j = j + 1
326
327 A_Fill_P2_R[11:,0] = FillR[22:]
328 A_Fill_P2_R = A_Fill_P2_R[::-1] #Fliping it back again (Top cell closser
    to surface)
329
330 "_________________Pile 3_________________"
331
332 " For Clay 2 Reduced Volumes "
333
334 V_Clay2_P3_R = V_Clay2_P3_R[::-1] #Fliping it (Top cell closser to
    surface)
335
336 " For Clay 2 Reduced Areas "
337
338 A_Clay2_P3_R = A_Clay2_P3_R[::-1] #Fliping it (Top cell closser to
    surface)
339
340 " For Clay 1 Reduced Volumes "
341
342 V_Clay1_P3_R = V_Clay1_P3_R[::-1] #Fliping it back again (Top cell
    closser to surface)
343
344 " For Clay 1 Reduced Areas "
345
346 A_Clay1_P3_R = A_Clay1_P3_R[::-1] #Fliping it (Top cell closser to
    surface)
347 Depth_C1_P3_Reduced = Depth_C1_P3_Reduced[::-1]
348
349 " For Fill Reduced Volumes "
350
351 FillR = V_Fill_P3_R
352 V_Fill_P3_R = np.zeros([61-10,1]) #New size, after merging the splited
    wedges into one
353
354 j = 0
355
356 for i in range (0, 20, 2):
357
358     Suma = FillR[i] + FillR[i+1]
359     Suma = Suma.item()
360
361     V_Fill_P3_R [j] = Suma
362
363     j = j + 1
364
365 V_Fill_P3_R[10:,0] = FillR[20:]
366 V_Fill_P3_R = V_Fill_P3_R[::-1] #Fliping it back again (Top cell closser
    to surface)
367
368 " For Fill Reduced Areas "
```

```
369
370 FillR = A_Fill_P3_R
371 A_Fill_P3_R = np.zeros([62-5,1]) #New size, after merging the splited
    wedges into one
372
373 j = 0
374
375 for i in range (0, 10, 2):
376     Suma = FillR[i] + FillR[i+1]
377     Suma = Suma.item()
378
379     A_Fill_P3_R [j] = Suma
380
381     j = j + 1
382
383 A_Fill_P3_R[5:,0] = FillR[10:]
384 A_Fill_P3_R = A_Fill_P3_R[::-1] #Fliping it back again (Top cell closser
    to surface)
385
386 "_________________Pile 4_________________"
387
388
389 " For Clay 2 Reduced Volumes "
390
391 V_Clay2_P4_R = V_Clay2_P4_R[::-1] #Fliping it (Top cell closser to
    surface)
392
393 " For Clay 2 Reduced Areas "
394
395 Clay2 = A_Clay2_P4_R
396 A_Clay2_P4_R = np.zeros([76,1])
397 Points = Depth_C2_P4_Reduced
398 Depth_C2_P4_Reduced = np.zeros_like(A_Clay2_P4_R)
399
400
401 j = 0
402
403 for i in range (0, 152, 2):
404     Suma = Clay2[i] + Clay2[i+1]
405     Suma = Suma.item()
406
407     A_Clay2_P4_R [j] = Suma
408     Depth_C2_P4_Reduced[j] = np.mean([Points[i], Points[i+1]])
409
410     j = j + 1
411
412 A_Clay2_P4_R[77:, 0] = Clay2[152]
413 A_Clay2_P4_R = A_Clay2_P4_R[::-1] #Fliping it (Top cell closser to
    surface)
414
415
416
417 " For Clay 1 Reduced Volumes "
418
419 Clay1R = V_Clay1_P4_R
420 V_Clay1_P4_R = np.zeros([35-10,1]) #New size, after merging the splited
    wedges into one
```

```
421
422 j = 0
423
424 for i in range (0, 20, 2):
425
426     Suma = Clay1R[i] + Clay1R[i+1]
427     Suma = Suma.item()
428
429     V_Clay1_P4_R [j] = Suma
430
431     j = j + 1
432
433 V_Clay1_P4_R[10:,0] = Clay1R[20:]
434 V_Clay1_P4_R = V_Clay1_P4_R[::-1] #Fliping it back again (Top cell
    closser to surface)
435
436 " For Clay 1 Reduced Areas "
437
438 Clay1R = A_Clay1_P4_R
439 A_Clay1_P4_R = np.zeros([44-10-9,1]) #New size, after merging the splited
     wedges into one
440 Points = Depth_C1_P4_Reduced
441 Depth_C1_P4_Reduced = np.zeros_like(A_Clay1_P4_R)
442
443 j = 0
444
445 for i in range (0, 20, 2):
446
447     Suma = Clay1R[i] + Clay1R[i+1]
448     Suma = Suma.item()
449
450     A_Clay1_P4_R [j] = Suma
451     Depth_C1_P4_Reduced[j] = Points[i]
452
453     j = j + 1
454
455 A_Clay1_P4_R[10:13,0] = Clay1R[20:23]
456 Depth_C1_P4_Reduced[10:13,0] = Points[20:23]
457
458 j = 14
459
460 for i in range (24, 44, 2):
461
462     Suma = Clay1R[i] + Clay1R[i+1]
463     Suma = Suma.item()
464
465     A_Clay1_P4_R [j] = Suma
466     Depth_C1_P4_Reduced[j] = Points[i]
467
468     j = j + 1
469
470 A_Clay1_P4_R[24] = Clay1R[43]
471 Depth_C1_P4_Reduced[24] = Points[43]
472 A_Clay1_P4_R = A_Clay1_P4_R[::-1] #Fliping it back again (Top cell
    closser to surface)
473 Depth_C1_P4_Reduced = Depth_C1_P4_Reduced[::-1]
474
```

```
" For Fill Reduced Volumes "

FillR = V_Fill_P4_R
V_Fill_P4_R = np.zeros([57-11,1]) #New size, after merging the splited
    wedges into one

j = 0

for i in range (0, 22, 2):

    Suma = FillR[i] + FillR[i+1]
    Suma = Suma.item()

    V_Fill_P4_R [j] = Suma

    j = j + 1

V_Fill_P4_R[11:46,0] = FillR[22:57]
V_Fill_P4_R = V_Fill_P4_R[::-1] #Fliping it back again (Top cell closser
    to surface)

" For Fill Reduced Areas "

FillR = A_Fill_P4_R
A_Fill_P4_R = np.zeros([89-11-31,1]) #New size, after merging the splited
     wedges into one

j = 0

for i in range (1, 22, 2):
    Suma = FillR[i] + FillR[i+1]
    Suma = Suma.item()

    A_Fill_P4_R [j] = Suma

    j = j + 1

A_Fill_P4_R[11:14,0] = FillR[22:25]

j = 14

for i in range (25, 88, 2):
    Suma = FillR[i] + FillR[i+1]
    Suma = Suma.item()

    A_Fill_P4_R [j] = Suma

    j = j + 1

A_Fill_P4_R[46] = FillR[88]
A_Fill_P4_R = A_Fill_P4_R[::-1] #Fliping it back again (Top cell closser
    to surface)

"______Obtaining reduction factors______"

" For un-reducted Pile 1  "

```

```
ndz = 119
Phi_w_P1 = np.zeros([ndz,1])
Phi_c_P1 = np.zeros([ndz,1])

WP1_UR = np.zeros([ndz,1])
AP1_UR = np.zeros([ndz,1])

#Clay 2
WP1_UR[(ndz - len(V_Clay2_P1_UR )):,0] = V_Clay2_P1_UR[::-1] * gamma_clay
AP1_UR[(ndz - len(A_Clay2_P1_UR )):,0] = A_Clay2_P1_UR[::-1] * c_clay2(
    Depth_C2_P1[::-1])

#Clay 1
WP1_UR[(ndz - len(V_Clay1_P1_UR )):,0] += V_Clay1_P1_UR[::-1] *
    gamma_clay
AP1_UR[(ndz - len(A_Clay1_P1_UR )):,0] += A_Clay1_P1_UR[::-1] * c_clay1(
    Depth_C1_P1[::-1])

#Filling
WP1_UR[(ndz - len(V_Fill_P1_UR )):,0] += V_Fill_P1_UR[::-1] *
    gamma_filling


" For reducted Pile 1  "

ndz = 119
WP1_R = np.zeros([ndz,1])
AP1_R = np.zeros([ndz,1])

#Clay 2
WP1_R[(ndz - len(V_Clay2_P1_R )):,0] = V_Clay2_P1_R * gamma_clay
AP1_R[(ndz - len(A_Clay2_P1_R )):,0] = A_Clay2_P1_R * c_clay2(Depth_C2_P1
    [::-1])

#Clay 1
Clay1R = np.zeros_like( V_Clay1_P1_UR )
Clay1R[:len( V_Clay1_P1_R )] = V_Clay1_P1_R [:,0]

WP1_R[(ndz - len(Clay1R)):,0] += Clay1R[:] * gamma_clay

Clay1R = np.zeros_like( A_Clay1_P1_UR )

Clay1 = A_Clay1_P1_R * c_clay1(Depth_C1_P1[::-1])
Clay1R[:len( A_Clay1_P1_R )] = Clay1[:,0]

AP1_R[(ndz - len(Clay1R )):,0] += Clay1R[:]

#Filling

Filling = np.zeros_like( V_Fill_P1_UR)
Filling[:len( V_Fill_P1_R)] = V_Fill_P1_R [:,0]
WP1_R[(ndz - len(Filling )):,0] += Filling[:]  * gamma_filling

for i in range(ndz):
    Phi_w_P1[i,0] = WP1_R[i] / WP1_UR[i]
    Phi_c_P1[i,0] = AP1_R[i] / AP1_UR[i]
    if np.isnan(Phi_c_P1[i,0]):
```

```
        Phi_c_P1[i,0] = 0
    if np.isnan(Phi_w_P1[i,0]):
        Phi_w_P1[i,0] = 0

" For un-reducted Pile 2  "

ndz = 51
Phi_w_P2 = np.zeros([ndz,1])
Phi_c_P2 = np.zeros([ndz,1])

WP2_UR = np.zeros([ndz,1])
AP2_UR = np.zeros([ndz,1])

#Clay 2
WP2_UR[(ndz - len(V_Clay2_P2_UR )):,0] = V_Clay2_P2_UR[::-1] * gamma_clay
AP2_UR[(ndz - len(A_Clay2_P2_UR )):,0] = A_Clay2_P2_UR[::-1] * c_clay2(
    Depth_C2_P2[::-1])

#Clay 1
WP2_UR[(ndz - len(V_Clay1_P2_UR )):,0] += V_Clay1_P2_UR[::-1] *
    gamma_clay
AP2_UR[(ndz - len(A_Clay1_P2_UR )):,0] += A_Clay1_P2_UR[::-1] * c_clay1(
    Depth_C1_P2[::-1])

#Filling
WP2_UR[(ndz - len(V_Fill_P2_UR )):,0] += V_Fill_P2_UR[::-1] *
    gamma_filling

" For reducted Pile 2  "

ndz = 51

WP2_R = np.zeros([ndz,1])
AP2_R = np.zeros([ndz,1])

#Clay 2
WP2_R[(ndz - len(V_Clay2_P2_R )):,0] = V_Clay2_P2_R * gamma_clay
AP2_R[(ndz - len(A_Clay2_P2_R )):,0] = A_Clay2_P2_R * c_clay2(
    Depth_C2_P2_Reduced[::-1])

#Clay 1
Clay1R = np.zeros_like( V_Clay1_P2_UR )
Clay1R[:len( V_Clay1_P2_R )] = V_Clay1_P2_R [:,0]

WP2_R[(ndz - len(Clay1R)):,0] += Clay1R[:] * gamma_clay

Clay1R = np.zeros_like( A_Clay1_P2_UR )

Clay1 = A_Clay1_P2_R * c_clay1(Depth_C1_P2_Reduced[::-1])
Clay1R[:len( A_Clay1_P2_R )] = Clay1[:,0]

AP2_R[(ndz - len(Clay1R )):,0] += Clay1R[:]

#Filling

Filling = np.zeros_like( V_Fill_P2_UR)
Filling[:len( V_Fill_P2_R)] = V_Fill_P2_R [:,0]
```

```
WP2_R[(ndz - len(Filling )):,0] += Filling[:]  * gamma_filling

for i in range(ndz):
    Phi_w_P2[i,0] = WP2_R[i] / WP2_UR[i]
    Phi_c_P2[i,0] = AP2_R[i] / AP2_UR[i]
    if np.isnan(Phi_c_P2[i,0]):
        Phi_c_P2[i,0] = 0
    if np.isnan(Phi_w_P2[i,0]):
        Phi_w_P2[i,0] = 0

" For un-reducted Pile 3  "

ndz = 51
Phi_w_P3 = np.zeros([ndz,1])
Phi_c_P3 = np.zeros([ndz,1])

WP3_UR = np.zeros([ndz,1])
AP3_UR = np.zeros([ndz,1])

#Clay 2
WP3_UR[(ndz - len(V_Clay2_P3_UR )):,0] = V_Clay2_P3_UR[::-1] * gamma_clay
AP3_UR[(ndz - len(A_Clay2_P3_UR )):,0] = A_Clay2_P3_UR[::-1] * c_clay2(
    Depth_C2_P3[::-1])

#Clay 1
WP3_UR[(ndz - len(V_Clay1_P3_UR )):,0] += V_Clay1_P3_UR[::-1] *
    gamma_clay
AP3_UR[(ndz - len(A_Clay1_P3_UR )):,0] += A_Clay1_P3_UR[::-1] * c_clay1(
    Depth_C1_P3[::-1])

#Filling
WP3_UR[(ndz - len(V_Fill_P3_UR )):,0] += V_Fill_P3_UR[::-1] *
    gamma_filling

" For reducted Pile 3  "

ndz = 51

WP3_R = np.zeros([ndz,1])
AP3_R = np.zeros([ndz,1])

#Clay 2
WP3_R[(ndz - len(V_Clay2_P3_R )):,0] = V_Clay2_P3_R * gamma_clay
AP3_R[(ndz - len(A_Clay2_P3_R )):,0] = A_Clay2_P3_R * c_clay2(
    Depth_C2_P3_Reduced[::-1])

#Clay 1

Clay1R = np.zeros_like( V_Clay1_P3_UR )
Clay1R[:len( V_Clay1_P3_R )] = V_Clay1_P3_R

WP3_R[(ndz - len(Clay1R)):,0] += Clay1R[:] * gamma_clay

Clay1R = np.zeros_like( A_Clay1_P3_UR )

Clay1 = A_Clay1_P3_R * c_clay1(Depth_C1_P3_Reduced[::-1])
Clay1R[:len( A_Clay1_P3_R )] = Clay1
```

```
684
685 AP3_R[(ndz - len(Clay1R )):,0] += Clay1R[:]
686
687 #Filling
688 Filling = np.zeros_like( V_Fill_P3_UR)
689 Filling[:len( V_Fill_P3_R)] = V_Fill_P3_R [:,0]
690 WP3_R[(ndz - len(Filling )):,0] += Filling[:]  * gamma_filling
691
692 for i in range(ndz):
693     Phi_w_P3[i,0] = WP3_R[i] / WP3_UR[i]
694     Phi_c_P3[i,0] = AP3_R[i] / AP3_UR[i]
695     if np.isnan(Phi_c_P3[i,0]):
696         Phi_c_P3[i,0] = 0
697     if np.isnan(Phi_w_P3[i,0]):
698         Phi_w_P3[i,0] = 0
699
700 " For un-reducted Pile 4  "
701
702 ndz = 119
703 Phi_w_P4 = np.zeros([ndz,1])
704 Phi_c_P4 = np.zeros([ndz,1])
705
706 WP4_UR = np.zeros([ndz,1])
707 AP4_UR = np.zeros([ndz,1])
708
709 #Clay 2
710 WP4_UR[(ndz - len(V_Clay2_P4_UR )):,0] = V_Clay2_P4_UR[::-1] * gamma_clay
711 AP4_UR[(ndz - len(A_Clay2_P4_UR )):,0] = A_Clay2_P4_UR[::-1] * c_clay2(
    Depth_C2_P4[::-1])
712
713 #Clay 1
714 WP4_UR[(ndz - len(V_Clay1_P4_UR )):,0] += V_Clay1_P4_UR[::-1] *
    gamma_clay
715 AP4_UR[(ndz - len(A_Clay1_P4_UR )):,0] += A_Clay1_P4_UR[::-1] * c_clay1(
    Depth_C1_P4[::-1])
716
717 #Filling
718
719 V_Fill_P4_UR = V_Fill_P4_UR[:119]
720 WP4_UR[(ndz - len(V_Fill_P4_UR )):,0] += V_Fill_P4_UR[::-1] *
    gamma_filling
721
722 " For reducted Pile 4  "
723
724 ndz = 119
725
726 WP4_R = np.zeros([ndz,1])
727 AP4_R = np.zeros([ndz,1])
728
729
730 #Clay 2
731 WP4_R[(ndz - len(V_Clay2_P4_R )):,0] = V_Clay2_P4_R * gamma_clay
732 AP4_R[(ndz - len(A_Clay2_P4_R )):,0] = (A_Clay2_P4_R * c_clay2(
    Depth_C2_P4_Reduced[::-1])).flatten()
733
734
735 #Clay 1
```

```
Clay1R = np.zeros_like( V_Clay1_P4_UR )
Clay1R[:len( V_Clay1_P4_R )] = V_Clay1_P4_R [:,0]

WP4_R[(ndz - len(Clay1R)):,0] += Clay1R[:] * gamma_clay

Clay1R = np.zeros_like( A_Clay1_P4_UR )

Clay1 = A_Clay1_P4_R * c_clay1(Depth_C1_P4_Reduced[::-1])
Clay1R[:len( A_Clay1_P4_R )] = Clay1[:,0]

AP4_R[(ndz - len(Clay1R )):,0] += Clay1R[:]

#Filling
Filling = np.zeros_like( V_Fill_P4_UR)
Filling[:len( V_Fill_P4_R)] = V_Fill_P4_R [:,0]
WP4_R[(ndz - len(Filling )):,0] += Filling[:]  * gamma_filling

for i in range(ndz):
    Phi_w_P4[i,0] = WP4_R[i] / WP4_UR[i]
    Phi_c_P4[i,0] = AP4_R[i] / AP4_UR[i]
    if np.isnan(Phi_c_P4[i,0]):
        Phi_c_P4[i,0] = 0
    if np.isnan(Phi_w_P4[i,0]):
        Phi_w_P4[i,0] = 0

" For un-reducted Pile 5  "

ndz = 119
Phi_w_P5 = np.zeros([ndz,1])
Phi_c_P5 = np.zeros([ndz,1])

WP5_UR = np.zeros([ndz,1])
AP5_UR = np.zeros([ndz,1])

#Clay 2
WP5_UR[(ndz - len(V_Clay2_P5_UR )):,0] = V_Clay2_P5_UR[::-1] * gamma_clay
AP5_UR[(ndz - len(A_Clay2_P5_UR )):,0] = A_Clay2_P5_UR[::-1] * c_clay2(
    Depth_C2_P5[::-1])

#Clay 1
WP5_UR[(ndz - len(V_Clay1_P5_UR )):,0] += V_Clay1_P5_UR[::-1] *
    gamma_clay
AP5_UR[(ndz - len(A_Clay1_P5_UR )):,0] += A_Clay1_P5_UR[::-1] * c_clay1(
    Depth_C1_P5[::-1])

#Filling
WP5_UR[(ndz - len(V_Fill_P5_UR )):,0] += V_Fill_P5_UR[::-1] *
    gamma_filling



" For reducted Pile 5  "

ndz = 119

WP5_R = np.zeros([ndz,1])
AP5_R = np.zeros([ndz,1])
```

```
789
790 #Clay 2
791 WP5_R[(ndz - len(V_Clay2_P5_R )):,0] = V_Clay2_P5_R[::-1] * gamma_clay
792 AP5_R[(ndz - len(A_Clay2_P5_R )):,0] = A_Clay2_P5_R[::-1] * c_clay2(
    Depth_C2_P5_Reduced[::-1])
793
794 #Clay 1
795 Clay1R = np.zeros_like( V_Clay1_P5_UR )
796 Clay1R[:len( V_Clay1_P5_R )] = V_Clay1_P5_R[::-1]
797
798 WP5_R[(ndz - len(Clay1R)):,0] += Clay1R * gamma_clay
799
800 Clay1R = np.zeros_like( A_Clay1_P5_UR )
801
802 Clay1 = A_Clay1_P5_R[::-1] * c_clay1(Depth_C1_P5_Reduced[::-1])
803 Clay1R[:len( A_Clay1_P5_R )] = Clay1
804
805 AP5_R[(ndz - len(Clay1R )):,0] += Clay1R[:]
806
807 #Filling
808 Filling = np.zeros_like( V_Fill_P5_UR)
809 Filling[:len( V_Fill_P5_R)] = V_Fill_P5_R [::-1]
810 WP5_R[(ndz - len(Filling )):,0] += Filling[:]  * gamma_filling
811
812
813 for i in range(ndz):
814     Phi_w_P5[i,0] = WP5_R[i] / WP5_UR[i]
815     Phi_c_P5[i,0] = AP5_R[i] / AP5_UR[i]
816     if np.isnan(Phi_c_P5[i,0]):
817         Phi_c_P5[i,0] = 0
818     if np.isnan(Phi_w_P5[i,0]):
819         Phi_w_P5[i,0] = 0
820
821
822 "______TO EXPORT RESULTS______"
823
824 " For Pile 1"
825 # Correct file path
826 file_path = r"C:\Users\SE1G4E\OneDrive - Chalmers\MSC\Python\Gothenburg
    Reduction Factors\Inclined\Phi_w_P1.csv"
827
828 # Make sure the folder exists
829 folder = os.path.dirname(file_path)
830 if not os.path.exists(folder):
831     os.makedirs(folder)
832
833 # Write CSV
834 with open(file_path, 'w', newline='') as csvfile:
835     writer = csv.writer(csvfile)
836     for row in Phi_w_P1:
837         writer.writerow(row)
838
839 # Correct file path
840 file_path = r"C:\Users\SE1G4E\OneDrive - Chalmers\MSC\Python\Gothenburg
    Reduction Factors\Inclined\Phi_c_P1.csv"
841
842 # Make sure the folder exists
```

```
folder = os.path.dirname(file_path)
if not os.path.exists(folder):
    os.makedirs(folder)

# Write CSV
with open(file_path, 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    for row in Phi_c_P1:
        writer.writerow(row)



" For Pile 2"
# Correct file path
file_path = r"C:\Users\SE1G4E\OneDrive - Chalmers\MSC\Python\Gothenburg Reduction Factors\Inclined\Phi_w_P2.csv"

# Make sure the folder exists
folder = os.path.dirname(file_path)
if not os.path.exists(folder):
    os.makedirs(folder)

# Write CSV
with open(file_path, 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    for row in Phi_w_P2:
        writer.writerow(row)

# Correct file path
file_path = r"C:\Users\SE1G4E\OneDrive - Chalmers\MSC\Python\Gothenburg Reduction Factors\Inclined\Phi_c_P2.csv"

# Make sure the folder exists
folder = os.path.dirname(file_path)
if not os.path.exists(folder):
    os.makedirs(folder)

# Write CSV
with open(file_path, 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    for row in Phi_c_P2:
        writer.writerow(row)




" For Pile 3"
# Correct file path
file_path = r"C:\Users\SE1G4E\OneDrive - Chalmers\MSC\Python\Gothenburg Reduction Factors\Inclined\Phi_w_P3.csv"

# Make sure the folder exists
folder = os.path.dirname(file_path)
if not os.path.exists(folder):
    os.makedirs(folder)

# Write CSV
```

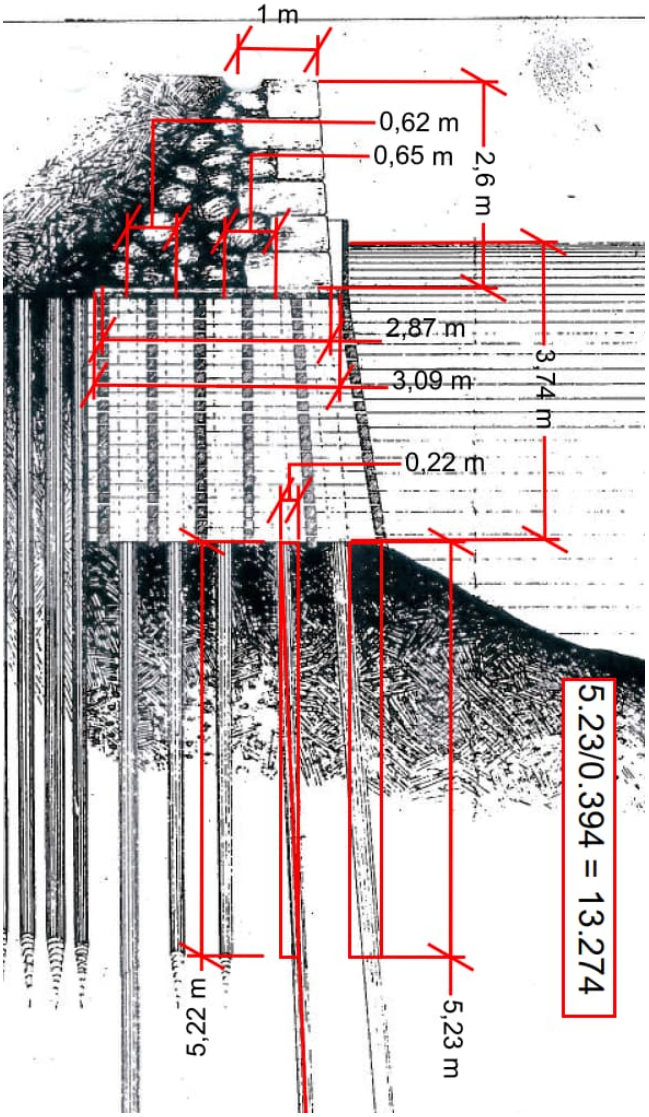
```
with open(file_path, 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    for row in Phi_w_P3:
        writer.writerow(row)

# Correct file path
file_path = r"C:\Users\SE1G4E\OneDrive - Chalmers\MSC\Python\Gothenburg Reduction Factors\Inclined\Phi_c_P3.csv"

# Make sure the folder exists
folder = os.path.dirname(file_path)
if not os.path.exists(folder):
    os.makedirs(folder)

# Write CSV
with open(file_path, 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    for row in Phi_c_P3:
        writer.writerow(row)



" For Pile 4"
# Correct file path
file_path = r"C:\Users\SE1G4E\OneDrive - Chalmers\MSC\Python\Gothenburg Reduction Factors\Inclined\Phi_w_P4.csv"

# Make sure the folder exists
folder = os.path.dirname(file_path)
if not os.path.exists(folder):
    os.makedirs(folder)

# Write CSV
with open(file_path, 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    for row in Phi_w_P4:
        writer.writerow(row)

# Correct file path
file_path = r"C:\Users\SE1G4E\OneDrive - Chalmers\MSC\Python\Gothenburg Reduction Factors\Inclined\Phi_c_P4.csv"

# Make sure the folder exists
folder = os.path.dirname(file_path)
if not os.path.exists(folder):
    os.makedirs(folder)

# Write CSV
with open(file_path, 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    for row in Phi_c_P4:
        writer.writerow(row)



" For Pile 5"
# Correct file path
```

```
file_path = r"C:\Users\SE1G4E\OneDrive - Chalmers\MSC\Python\Gothenburg Reduction Factors\Inclined\Phi_w_P5.csv"

# Make sure the folder exists
folder = os.path.dirname(file_path)
if not os.path.exists(folder):
    os.makedirs(folder)

# Write CSV
with open(file_path, 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    for row in Phi_w_P5:
        writer.writerow(row)

# Correct file path
file_path = r"C:\Users\SE1G4E\OneDrive - Chalmers\MSC\Python\Gothenburg Reduction Factors\Inclined\Phi_c_P5.csv"

# Make sure the folder exists
folder = os.path.dirname(file_path)
if not os.path.exists(folder):
    os.makedirs(folder)

# Write CSV
with open(file_path, 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    for row in Phi_c_P5:
        writer.writerow(row)
```

J Case study 2 - Input parameters

The input parameters used in the FE-models for Case study 2, in the sensitivity study of the Gothenburg walls, can be seen below:

GOTHENBURG INPUT-DATA TO FE-MODEL



$E := 7000\ MPa = \left(7 \cdot 10^{6}\right)\ kPa$

(Note typo in report as 7000 kN/ m² is mentioned)

$D_{min} := 0.2\ m$

$D_{max} := 0.25\ m$

$I_{min} := \frac{D_{min}^{4} \cdot \pi}{64} = \left(7.854 \cdot 10^{3}\right)\ cm^{4}$

$I_{max} := \frac{D_{max}^{4} \cdot \pi}{64} = \left(1.917 \cdot 10^{4}\right)\ cm^{4}$

$W_{min} := \frac{D_{min}^{3} \cdot \pi}{32} = 785.398\ cm^{3}$

$W_{max} := \frac{D_{max}^{3} \cdot \pi}{32} = \left(1.534 \cdot 10^{3}\right)\ cm^{3}$

$E \cdot I_{min} = 549.779\ kN \cdot m^{2}$

$E \cdot I_{max} = 1342\ kN \cdot m^{2}$

$t_{sheet} := 60\ mm$

$L_{sheet} := 3.95\ m$

Interface between pile cap and timber deck / beams assumed as hinged.

$h_{wall} := 2.6\ m$ - Wall height

$\gamma_{masonry} := 27\ \frac{kN}{m^{3}}$ - Density of masonry wall

$B_{wall} := 2.6\ m$ - Base width of wall, assumed based on pile spacing

$B_{wall.top} := 1\ m$ - Top width of wall

$e_{pile} := \frac{D_{min}}{2} = 0.1\ m$ - Distance from pile mid to edge of masonry wall

$N_{pile.top} := h_{wall} = 2.6\ m$ - Level of pile-head

$B_{wall.pile} := B_{wall} - 2 \cdot e_{pile} = 2.4\ m$ - Width of timber deck in Abaqus

$b_{deck} := 0.6\ m$ $h_{deck} := 0.06\ m$ - Geometry of timber deck

$I_{deck} := \frac{b_{deck} \cdot {h_{deck}}^3}{12} = \left(1.08 \cdot 10^{-5}\right)\ m^4$ - Moment of inertia, timber deck

According to the historical drawings of the Gothenburg quay wall the timber deck is not, unlike the reference-case, supported on headstocks. The bottom layer of beams, oriented longitudinally in the direction of the wall have a reduced stiffness for transversal bending.

Since the stiffness is lower, perpendicular to the grains for timber a weighted average of the stiffness is calculated.

$E_{0.mean} := 11000\ MPa$ Young's modulus, parallell to grain

$E_{90.mean} := 370\ MPa$ Young's modulus, perpendicular to grain

$E_{ratio} := \frac{E_{90.mean}}{E_{0.mean}} = 0.034$ Ratio, perpendicular / parallell

$I_{beam} := I_{deck} \cdot E_{ratio} + I_{deck} = \left(1.116 \cdot 10^{-5}\right)\ m^4$ Weighted moment of inertia of timber deck

$h_{deck.eq} := \sqrt[3]{\frac{I_{beam} \cdot 12}{b_{deck}}} = 0.061\ m$ - Equivalent height of deck in Abaqus

Soil stresses

Vertical at pile deck

$\gamma_{fill.dry} := 18\ \frac{kN}{m^3}$ $h_{dry} := 1.8\ m$ - Fill over GW-level

$\sigma_{v'.lay1} := \gamma_{fill.dry} \cdot h_{dry} = 32.4\ kPa$ - Vertical stress, first layer

$\gamma_{fill.wet} := 10\ \frac{kN}{m^3}$ $h_{wet} := h_{wall} - h_{dry} = 0.8\ m$ - Filling below GW to timber deck

$\sigma_{v'.lay2} := \sigma_{v'.lay1} + \gamma_{fill.wet} \cdot h_{wet} = 40.4\ kPa$ - Vertical stress, bot sand layer

$\sigma_{v'.deck} := \sigma_{v'.lay2} = 40.4\ kPa$ - Vertical stress on timber deck

$\varphi := 33\ deg$ - Friction angle at wall

$K_{A.wall} := \left(\tan\left(45\ deg - \frac{\varphi}{2}\right)\right)^2$ - Active soil pressure coefficient on wall

$K_0 := 1 - \sin(\varphi) = 0.455$ - Eearth pressure at rest

Input to Abaqus:

$\sigma_{v'.deck} = 40.4\ kPa$ - Vertical stress on top of timber deck from filling material

Wall forces, calculated for active and at rest soil conditions

$$\beta := \operatorname{atan}\left(\frac{B_{wall} - B_{wall.top}}{h_{wall}}\right) = 31.608\ \mathit{deg}$$

Vertical stress

$$\sigma_{v'.lay1} = 32.4\ \mathit{kPa}$$

$$\sigma_{v'.deck} = 40.4\ \mathit{kPa}$$

Height of soil:

$$h_{dry} = 1.8\ \mathit{m}$$

$$h_{wet} = 0.8\ \mathit{m}$$

Surcharge load:

$$q_k := 5\ \mathit{kPa}$$

Force components from soil stresses:

$$F_{V1} := \sigma_{v'.lay1} \cdot \frac{h_{dry}}{2} = 29.16\ \frac{\mathit{kN}}{\mathit{m}}$$

$$F_{V2} := \sigma_{v'.lay1} \cdot h_{wet} = 25.92\ \frac{\mathit{kN}}{\mathit{m}}$$

$$F_{V3} := \left(\sigma_{v'.deck} - \sigma_{v'.lay1}\right) \cdot \frac{h_{wet}}{2} = 3.2\ \frac{\mathit{kN}}{\mathit{m}}$$

Lever arm of forces:

$$e_{H1} := h_{wet} + \frac{h_{dry}}{3} = 1.4\ \mathit{m}$$

$$e_{H2} := \frac{h_{wet}}{2} = 0.4\ \mathit{m}$$

$$e_{H3} := \frac{h_{wet}}{3} = 0.267\ \mathit{m}$$

Force components from surcharge load:

$$F_{V4} := q_k \cdot h_{wall} = 13\ \frac{\mathit{kN}}{\mathit{m}}$$

$$e_{H4} := \frac{h_{wall}}{2}$$

Overturning moment:

$$M_{OT.A} := K_{A.wall} \cdot \left(F_{V1} \cdot e_{H1} + F_{V2} \cdot e_{H2} + F_{V3} \cdot e_{H3} + F_{V4} \cdot e_{H4}\right) \cdot \cos\left(\beta\right) = 17.31\ \frac{\mathit{kN} \cdot \mathit{m}}{\mathit{m}}$$

$$M_{OT.0} := \frac{M_{OT.A}}{K_{A.wall}} \cdot K_0 = 26.738\ \frac{\mathit{kN} \cdot \mathit{m}}{\mathit{m}}$$

Vertical force from wall:

$B_{wall} = 2.6\ m$ $h_{wall} = 2.6\ m$ $\gamma_{masonry} = 27\ \frac{kN}{m^3}$ $B_{wall.top} = 1\ m$

$B_{wall.dry} := B_{wall.top} + h_{dry} \cdot \sin(\beta) = 1.943\ m$ - Wall width at transition from above to below GW-level

$F_{v.wall.dry} := B_{wall.top} \cdot h_{dry} \cdot \gamma_{masonry} + \frac{(B_{wall.dry} - B_{wall.top}) \cdot h_{dry} \cdot \gamma_{masonry}}{2} = 71.524\ \frac{kN}{m}$

$h_{wet} = 0.8\ m$ $\gamma_{masonry.w} := \gamma_{masonry} - 10\ \frac{kN}{m^3} = 17\ \frac{kN}{m^3}$

$F_{v.wall.wet} := B_{wall.dry} \cdot h_{wet} \cdot \gamma_{masonry.w} + \frac{(B_{wall} - B_{wall.dry}) \cdot h_{wet} \cdot \gamma_{masonry.w}}{2} = 30.895\ \frac{kN}{m}$

$F_{v.wall} := F_{v.wall.dry} + F_{v.wall.wet} = 102.419\ \frac{kN}{m}$

Excentricity for active and at rest soil pressure:

$e_{Fv.A} := \frac{M_{OT.A}}{F_{v.wall}} = 0.169\ m$ $B_{eff.A} := B_{wall} - 2 \cdot e_{Fv.A} = 2.262\ m$ $\frac{B_{eff.A}}{B_{wall}} = 0.87$

$e_{Fv.0} := \frac{M_{OT.0}}{F_{v.wall}} = 0.261\ m$ $B_{eff.0} := B_{wall} - 2 \cdot e_{Fv.0} = 2.078\ m$ $\frac{B_{eff.0}}{B_{wall}} = 0.799$

<u>Ground pressure at Active earth pressure condition</u>

$lift_A :=$ if $e_{Fv.A} < \frac{B_{wall}}{6}$ return "No lifting" else if $\frac{B_{wall}}{6} < e_{Fv.A} < \frac{B_{wall}}{3}$ return "Lifting, calculate B_effective" else return "Unstable" $=$ "No lifting"

$\frac{F_{v.wall}}{B_{wall}} + \frac{6 \cdot M_{OT.A}}{B_{wall}^2} = 54.756\ kPa$ - Maximum stress when e < B/6

$\frac{4 \cdot F_{v.wall}}{3} \cdot \frac{1}{(B_{wall} - 2 \cdot e_{Fv.A})} = 60.371\ kPa$ - Maximum stress when B/6 < e < B/3

$$\sigma_{max.A} := \begin{cases} \left\| \dfrac{F_{v.wall}}{B_{wall}} + \dfrac{6 \cdot M_{OT.A}}{B_{wall}^{\,2}} \right. & \text{if } e_{Fv.A} < \dfrac{B_{wall}}{6} \\ \left\| \dfrac{4 \cdot F_{v.wall}}{3} \cdot \dfrac{1}{\left(B_{wall} - 2 \cdot e_{Fv.A}\right)} \right. & \text{else if } \dfrac{B_{wall}}{6} < e_{Fv.A} < \dfrac{B_{wall}}{3} \end{cases} = 54.756 \ \textbf{kPa}$$

- Maximum stress

$$\sigma_{min.A} := \begin{cases} \left\| \dfrac{F_{v.wall}}{B_{wall}} - \dfrac{6 \cdot M_{OT.A}}{B_{wall}^{\,2}} \right. & \text{if } e_{Fv.A} < \dfrac{B_{wall}}{6} \\ \left\| 0 \ \textbf{kPa} \right. & \text{else if } \dfrac{B_{wall}}{6} < e_{Fv.A} < \dfrac{B_{wall}}{3} \end{cases} = 24.028 \ \textbf{kPa}$$

- Minimum stress

Load input beneath masonry wall in Abaqus for Active soil pressure:

$B_{eff.A} = 2.262\ m$ - Distribution width of vertical load

$lift_A = \text{“No lifting”}$ - If "No liftning", pressure can be applied on the whole timber deck

$\sigma_{max.A} = 54.756\ kPa$ - Maximum pressure at wall front

$\sigma_{min.A} = 24.028\ kPa$ - Minimum pressure at back of wall

$\Delta\sigma_A := \sigma_{max.A} - \sigma_{min.A} = 30.728\ kPa$ - Difference between max and min pressure

$e_{pile} = 0.1\ m$

Input to Abaqus - Vertical pressure from wall, Active earth pressure

$\sigma_{p.max.A} := \left(\sigma_{max.A} - \Delta\sigma_A \cdot \frac{e_{pile}}{B_{wall}}\right) b_{deck} = 32.144\ \frac{kN}{m}$ - Max pressure at pile

$F_{p.max.A} := \frac{\sigma_{max.A} \cdot b_{deck} + \sigma_{p.max.A}}{2} \cdot e_{pile} = 3.25\ kN$ - Added force at pile from max pressure

$\sigma_{p.min.A} := \left(\sigma_{min.A} + \Delta\sigma_A \cdot \frac{e_{pile}}{B_{wall}}\right) b_{deck} = 15.126\ \frac{kN}{m}$ - Min pressure at pile

$F_{p.min.A} := \frac{\sigma_{min.A} \cdot b_{deck} + \sigma_{p.min.A}}{2} \cdot e_{pile} = 1.477\ kN$ - Added force at pile from min pressure

$\Delta\sigma_{A.p} := \sigma_{p.max.A} - \sigma_{p.min.A} = 17.019\ \frac{kN}{m}$ - Pressure difference

Equation for analytical pressure on timber deck to Abaqus:

$\sigma_A(x) := \sigma_{p.max.A} - \frac{x \cdot \Delta\sigma_{A.p}}{B_{wall.pile}}$

Ground pressure at Rest earth pressure condition

$$lift_0 := \text{if } e_{Fv.0} < \frac{B_{wall}}{6} \text{ return "No lifting"} \text{ else if } \frac{B_{wall}}{6} < e_{Fv.0} < \frac{B_{wall}}{3} \text{ return "Lifting, calculate B_effective"} \text{ else return "Unstable"} = \text{"No lifting"}$$

$$\frac{F_{v.wall}}{B_{wall}} + \frac{6 \cdot M_{OT.0}}{B_{wall}^{2}} = 63.124 \ kPa$$ - Maximum stress when e < B/6

$$\frac{4 \cdot F_{v.wall}}{3} \cdot \frac{1}{\left(B_{wall} - 2 \cdot e_{Fv.0}\right)} = 65.72 \ kPa$$ - Maximum stress when B/6 < e < B/3

$$\sigma_{max.0} := \text{if } e_{Fv.0} < \frac{B_{wall}}{6} \left\| \frac{F_{v.wall}}{B_{wall}} + \frac{6 \cdot M_{OT.0}}{B_{wall}^{2}} \right. \text{ else if } \frac{B_{wall}}{6} < e_{Fv.0} < \frac{B_{wall}}{3} \left\| \frac{4 \cdot F_{v.wall}}{3} \cdot \frac{1}{\left(B_{wall} - 2 \cdot e_{Fv.0}\right)} \right. = 63.124 \ kPa$$ - Maximum stress

$$\sigma_{min.0} := \text{if } e_{Fv.0} < \frac{B_{wall}}{6} \left\| \frac{F_{v.wall}}{B_{wall}} - \frac{6 \cdot M_{OT.0}}{B_{wall}^{2}} \right. \text{ else if } \frac{B_{wall}}{6} < e_{Fv.0} < \frac{B_{wall}}{3} \left\| 0 \ kPa \right. = 15.66 \ kPa$$ - Minimum stress

Load input beneath masonry wall in Abaqus for Active soil pressure:

$B_{eff.0} = 2.078\ m$ - Distribution width of load

$lift_0 = \text{“No lifting”}$ - If "No liftning", pressure can be applied on the whole timber deck

$\sigma_{max.0} = 63.124\ kPa$ - Maximum pressure at wall front

$\sigma_{min.0} = 15.66\ kPa$ - Minimum pressure at back of wall

$\Delta\sigma_0 := \sigma_{max.0} - \sigma_{min.0} = 47.464\ kPa$

$e_{pile} = 0.1\ m$

Input to Abaqus - Vertical pressure from wall, Earth pressure at rest

$\sigma_{p.max.0} := \left(\sigma_{max.0} - \Delta\sigma_0 \cdot \frac{e_{pile}}{B_{wall}}\right) b_{deck} = 36.779\ \frac{kN}{m}$ - Max pressure at pile

$F_{p.max.0} := \frac{\sigma_{max.0} \cdot b_{deck} + \sigma_{p.max.0}}{2} \cdot e_{pile} = 3.733\ kN$ - Added force at pile from max pressure

$\sigma_{p.min.0} := \left(\sigma_{min.0} + \Delta\sigma_0 \cdot \frac{e_{pile}}{B_{wall}}\right) b_{deck} = 10.491\ \frac{kN}{m}$ - Min pressure at pile

$F_{p.min.0} := \frac{\sigma_{min.0} \cdot b_{deck} + \sigma_{p.min.0}}{2} \cdot e_{pile} = 0.994\ kN$ - Added force at pile from min pressure

$\Delta\sigma_{0.p} := \sigma_{p.max.0} - \sigma_{p.min.0} = 26.288\ \frac{kN}{m}$ - Pressure difference

Equation for analytical pressure on timber deck to Abaqus:

$\sigma_0(x) := \sigma_{p.max.0} - \frac{x \cdot \Delta\sigma_{0.p}}{B_{wall.pile}}$

Resulting horizontal force from wall, Active soil pressure

$F_{H.A} := K_{A.wall} \cdot \left(F_{V1} + F_{V2} + F_{V3} + F_{V4}\right) \cdot \cos\left(\beta\right) = 17.9 \ \frac{kN}{m}$ - Horizontal force on wall

$q_{h.wall.A} := \frac{F_{H.A}}{B_{wall.pile}} \cdot b_{deck} = 4.47 \ \frac{kN}{m}$ - Horizontal line-load from wall

Resulting horizontal force from wall, Resting soil pressure

$F_{H.0} := \frac{F_{H.A}}{K_{A.wall}} \cdot K_0 = 27.64 \ \frac{kN}{m}$ - Horizontal force on wall

$q_{h.wall.0} := \frac{F_{H.0}}{B_{wall.pile}} \cdot b_{deck} = 6.91 \ \frac{kN}{m}$ - Horizontal line-load from wall

Input to Abaqus

- Horizontal line-load from wall, Active / at Rest earth pressure:

$q_{h.wall.A} = 4.474 \ \frac{kN}{m}$

$q_{h.wall.0} = 6.911 \ \frac{kN}{m}$

Self weight timber:

$w_t := 370 \ \frac{kg}{m^3}$ $g = 9.807 \ \frac{m}{s^2}$

$g_t := w_t \cdot g = 3.628 \ \frac{kN}{m^3}$

Geometrical input to assign springs in Abaqus

$$Depth := \begin{bmatrix} h_{dry} \\ h_{dry}+h_{wet} \\ 5.8\ m \\ 7.6\ m \\ 20\ m \end{bmatrix} = \begin{bmatrix} 1.8 \\ 2.6 \\ 5.8 \\ 7.6 \\ 20 \end{bmatrix} m \qquad Depth_{rl} := \begin{bmatrix} 2 \\ 0 \\ -0.6 \\ -3.9 \\ -7 \\ -12 \end{bmatrix}$$

$\Delta D_{piles} := 2.6\ m$ - Depth to different soil

$N_{pile.top} = 2.6\ m$ - Depth of pile top

$$Depth\_piles := Depth - \Delta D_{piles} = \begin{bmatrix} -0.8 \\ 0 \\ 3.2 \\ 5 \\ 17.4 \end{bmatrix} m$$

- Equivalent depth of soil layers counting top of pile as the 0 base

K Case study 2 - Python script for Abaqus

The Python script used to run the Abaqus analysis in Case study 2 is shown below:

```
# -*- coding: mbcs -*-
# Do not delete the following import lines
from abaqus import *
from abaqusConstants import *
import __main__

import section
import regionToolset
import displayGroupMdbToolset as dgm
import part
import material
import assembly
import step
import interaction
import load
import mesh
import optimization
import job
import sketch
import visualization
import xyPlot
import displayGroupOdbToolset as dgo
import connectorBehavior
import numpy as np
import math as ma

import os
import sys
import csv

sys.path.append(r"c:\program files\python312\lib\site-packages")

cwd = os.getcwd()

sys.path.append(f"{cwd}")


### IMPORTING REDUCTION FACTORS FOR SPRINGS ###


#OBSERVE, MAKE SURE THE CORRECT Psi_factors.csv FILE IS USED!

    #Reading csv-file and storing reduction factors
    #Note that csv files are ordered from back to front (Pile 5 is front
   row pile)
    #Numbering is reordered in Abaqus-script

    #Psi_c factors for all piles
with open('Phi_c_P1.csv', 'r' ) as csvfile:
    Psi_c_P5 = np.genfromtxt(csvfile, dtype=float, delimiter=';')
with open('Phi_c_P2.csv', 'r' ) as csvfile:
    Psi_c_P4 = np.genfromtxt(csvfile, dtype=float, delimiter=';')
```

```
with open('Phi_c_P3.csv', 'r' ) as csvfile:
    Psi_c_P3 = np.genfromtxt(csvfile, dtype=float, delimiter=';')
with open('Phi_c_P4.csv', 'r' ) as csvfile:
    Psi_c_P2 = np.genfromtxt(csvfile, dtype=float, delimiter=';')
with open('Phi_c_P5.csv', 'r' ) as csvfile:
    Psi_c_P1 = np.genfromtxt(csvfile, dtype=float, delimiter=';')

    #Psi_w factors for all piles
with open('Phi_w_P1.csv', 'r' ) as csvfile:
    Psi_w_P5 = np.genfromtxt(csvfile, dtype=float, delimiter=';')
with open('Phi_w_P2.csv', 'r' ) as csvfile:
    Psi_w_P4 = np.genfromtxt(csvfile, dtype=float, delimiter=';')
with open('Phi_w_P3.csv', 'r' ) as csvfile:
    Psi_w_P3 = np.genfromtxt(csvfile, dtype=float, delimiter=';')
with open('Phi_w_P4.csv', 'r' ) as csvfile:
    Psi_w_P2 = np.genfromtxt(csvfile, dtype=float, delimiter=';')
with open('Phi_w_P5.csv', 'r' ) as csvfile:
    Psi_w_P1 = np.genfromtxt(csvfile, dtype=float, delimiter=';')


include_red = 0     #Set to 1 to include soil-wedge reduction
NL = ON            #Set to ON or OFF to include or not include Non-linear
    geometry
include_OP = 0      #Set to 1 or 0 to include or not include Overburden
   pressure
include_fill = 1     #Set to 1 or 0 to include or not include springs in
   filling-layer

Name_Output = f'Red{include_red}_Fill{include_fill}'

### DEFINING ITTERATIONS TO SIMULATE ###


#Defining the amount of itterations of pile diameters
div_D = 3

min_D = 0.18 #Minimum diameter of piles to analyze
D_dif = 0.02 #Increment of diameter increase

#Defining the amount of itterations of varying inclination
div_inc = 2

# Number of itterations to check with active and at rest earth pressure
div_K = 2

#Number of itterations for including or not including front row pile
div_FP = 2

#Setting amount of pile rows

num_rows = div_D * div_inc * div_K * div_FP  #Set number of itterations

D_it = np.zeros(num_rows)
inc_it = np.zeros(num_rows)
K_it = np.zeros(num_rows)
FP_it = np.zeros(num_rows)

```

```
name_job_it = []

#Assigning values for the itterative analysis
for i in range(div_D):

    D_inp = min_D + i*D_dif
    #D_name = f'D{i+1}'
    D_name = f'{round(D_inp*100)}' #Rounded to ensure sucessful naming of job

    for j in range(div_inc):

        if j == 0:
            inc_name = 10
        else:
            inc_name = 6

        for k in range(div_K):

            if k == 1:
                red_name = 'A'
            else:
                red_name = k

            for l in range(div_FP):

                row_base = i*div_inc*div_K*div_FP + j*div_K*div_FP + k*div_FP
                row_nr = row_base + l

                D_it[row_nr] = D_inp
                inc_it[row_nr] = inc_name
                K_it[row_nr] = k
                FP_it[row_nr] = l

                name_job_it.append(f'{D_name}_K{red_name}_inc{inc_name}_FP{l}_{Name_Output}') #Name of the current job

Loop = len(D_it) #Number of itterations in the analysis

for it in range(Loop):

    #Units are imported as kN, m, kPa, kN/m^3

    Mdb()

    name_job = name_job_it[it]

    #Pile geometry
    #L = Abaqus_inp.L
    L = 12
    L_P2 = 5.2

    x_pile = np.array([0, 0.6, 1.2, 1.8, 2.4])

    pile_top = 0

```

```
    #Initial depth of pile to determine K-factors with or without OP
    if include_OP == 1:
        Depth_init = 2.6
    else:
        Depth_init = 0

    dx = x_pile[1]-x_pile[0]

    x_inc = ([1, 1, 0, 0, 0])       #Defining inclined piles with "1"
    x_short = ([0, 0, 1, 1, 0])     #Defining short piles with "1"

    inc_pile = inc_it[it]

    #Wall geometry
    B_wall = 2.4
    B_eff = B_wall

    #Partitioning division of deck-width

    if B_wall < dx:
        u_partition_deck = B_wall/dx
        print('Wall width less than pile spacing')
    else:
        u_partition_deck = 1
        print('Wall width larger than pile spacing')

    if B_eff < B_wall:
        u_partition_load = B_eff / B_wall
        print('Wall-edge liftup')
    else:
        u_partition_load = 1
        print('No lifting of wall')

    #Load from wall on Timber deck

    if K_it[it] == 0:
        sigma_wall_max = 36.779
        delta_sigma_wall = 26.288
        q_h_wall = -6.91
        F_p_max = -3.733
        F_p_min = -0.994
    else:
        sigma_wall_max = 32.144
        delta_sigma_wall = 17.019
        q_h_wall = -4.47
        F_p_max = -3.25
        F_p_min = -1.477

    #Increase of effective vertical stress due to overburden pressure

    if include_OP == 1:
        sigma_p = 40.4
    else:
        sigma_p = 0


    #BCs for pile base, SET = SUPPORTED, UNSET = UNSUPPORTED
```

```
    U1_PB = UNSET
    U2_PB = SET
    U3_PB = UNSET

    #Setting Mesh-properties

    #ndz = Abaqus_inp.ndz
    dz = 0.1
    ndz = ma.floor(L/dz)
    ndz_P2 = ma.floor(L_P2/dz)
    n_div = 1   #Number of divisions in mesh in between nodes

    #Depths
    Depth_div = [-3.2,-4.3,-20]

    z_depth = np.zeros(ndz)

    for i in range(ndz):
        z_depth[i] = -(dz/2+i*dz)

    #z_depth contains the z position of each spring

    #Meshsize
    meshsize_pile = L/(n_div*ndz)


    #ADDING ANALYSIS STEP


    Step_name = 'Step-1'

    mdb.models['Model-1'].StaticStep(name=Step_name, previous='Initial',
    nlgeom=NL)
    mdb.models['Model-1'].steps[Step_name].setValues(initialInc=0.01,
    maxNumInc=1000, maxInc=0.05)


    #MATERIAL DEFINITIONS


    Material = 'Timber'
    Youngs_mod = 7000000
    Density = 3.628         #In [kN/m^3]
    nu = 0.2

    #Pile section

    #D = Abaqus_inp.D
    D = D_it[it]
    Radius = D/2
    Section = 'Pile'
    name_profile=f'Section r={Radius}'

    #Timber deck section
    b_deck = 0.6      #Width of timber deck
    h_deck = 0.061  #Height of timber deck

```

```
    Section_Deck = 'Timber_Deck'
    name_profile_deck = f'Section b={b_deck}_h={h_deck}'


### CREATING MATERIAL AND SECTIONS ###


    mdb.models['Model-1'].Material(name=Material)
    mdb.models['Model-1'].materials[Material].Density(table=((Density, ),
    ))
    mdb.models['Model-1'].materials[Material].Elastic(table=((Youngs_mod,
    nu), ))

    print(f"Material {Material} created")

    mdb.models['Model-1'].CircularProfile(name=name_profile, r=Radius)
    mdb.models['Model-1'].BeamSection(name=Section, integration=
    DURING_ANALYSIS,
        poissonRatio=0.0, profile=name_profile, material=Material,
        temperatureVar=LINEAR, beamSectionOffset=(0.0, 0.0),
        consistentMassMatrix=False)

    print(f"Section {Section} created")

    mdb.models['Model-1'].RectangularProfile(name=name_profile_deck, a=
    b_deck, b=h_deck)
    mdb.models['Model-1'].BeamSection(name=Section_Deck,
        integration=DURING_ANALYSIS, poissonRatio=0.0, profile=
    name_profile_deck,
        material='Timber', temperatureVar=LINEAR, beamSectionOffset=(0.0,
    0.0),
        consistentMassMatrix=False)

    print(f"Section {Section} created")


### CALCULATION OF SPRING PROPERTIES FOR THE DIFFERENT DEPTHS ###

    #Assigning Psi-factors to a combined matrix
    Psi_w = np.zeros([ndz,len(x_pile)])
    Psi_c = np.zeros([ndz,len(x_pile)])

    Psi_w[:,0] = Psi_w_P1
    Psi_w[:,1] = Psi_w_P2
    Psi_w[0:ndz_P2,2] = Psi_w_P3
    Psi_w[0:ndz_P2,3] = Psi_w_P4
    Psi_w[:,4] = Psi_w_P5

    Psi_c[:,0] = Psi_c_P1
    Psi_c[:,1] = Psi_c_P2
    Psi_c[0:ndz_P2,2] = Psi_c_P3
    Psi_c[0:ndz_P2,3] = Psi_c_P4
    Psi_c[:,4] = Psi_c_P5

    for a in range(ndz):
        for b in range(len(x_pile)):
            if Psi_c[a,b] > 1:
```

```
                Psi_c[a,b] = 1

    #Calculation of plastic limit of springs

    "Data from Table 6.2 "

    Depth = np.array([
                       [0, 3.2],
                       [3.2, 4.3],
                       [4.3, 20]
                      ]) #Depth ranges in [m]

    gamma = np.array([ [10],
                   [6],
                   [6]
                   ]) # gamma'values in [kN/m^3]

    E_m = [1500, 50*13, 50*13]  #Assumed elasticity modulus, see Appendix

    c = np.array([ [0],
                   [13],
                   [1.9] #Incremented increase per m below -5
                 ]) #Cohesion [kN/m^2]

    fi = np.array([ [33],
                   [0],
                   [0]
                 ])

    sigmav = np.zeros([len(gamma)+1,1])

    " To obtain vertical stress: "

    def sigma_v(gamma, Depth_start, Depth_end, P0):

        sigma = P0 + gamma * ( Depth_end - Depth_start)

        return sigma

    for i in range (1,len (sigmav) ):

        if i == 1:
            sigmav[i,0] = sigma_v(gamma[i-1,0], Depth[i-1,0], Depth[i
    -1,1], 0)

        else:
            sigmav[i,0] = sigma_v(gamma[i-1,0], Depth[i-1,0], Depth[i
    -1,1], 0) + sigmav[i-1,0]

    sigmav = sigmav + sigma_p   #Adding overburden pressure to vertical
    stress

    R = D / 2  # Radius

    Deppts = np.zeros([len(Depth) + 1, 1])
    Deppts[1:, 0] = Depth[:, 1]
    Deppts = Deppts.flatten()
```

```
    sigmav = sigmav.flatten()

    Depth_dz = np.zeros([ndz,1])

    " To compute plastic stresses for Brom's extention method "

    sigmap_BE = np.zeros([ndz,len(x_pile)]) #Brom's extention method

    Depth_div_pos = [Depth[0,1], Depth[1,1], Depth[2,1]]

    sigmavp = np.zeros([ndz,1])

    p_lim_w = np.zeros([ndz,1])
    p_lim_c = np.zeros([ndz,1])

    for j in range(ndz):

        dzp = dz/2 + dz*j  # Ensures dzp increases predictably
        Depth_dz[j] = dzp

        sigmavp[j] = np.interp(dzp, Deppts, sigmav)

        # Ensure all variables are scalars
        #sigmavp[j] = sigmavp.item() if isinstance(sigmavp, np.ndarray)
    else sigmavp

        if Depth_dz[j] < Depth_div_pos[0]:

            p_lim_w[j] = sigmavp[j] * 3 * (np.tan(np.radians(45) + np.
    radians(fi[0]/2)))**2
            p_lim_c[j] = 0

        elif Depth_div_pos[0] < Depth_dz[j] < Depth_div_pos[1]:

            p_lim_w[j] = 0
            p_lim_c[j] = 9 * c[1]

        else:

            p_lim_w[j] = 0
            p_lim_c[j] = 9 * (c[1] + (Depth_dz[j] - Depth_div_pos[1]) * c
    [2])

        for k in range(len(x_pile)):

            if include_red == 1:
                sigmap_BE[j,k] = Psi_w[j,k]*p_lim_w[j] + Psi_c[j,k]*
    p_lim_c[j]
            else:
                sigmap_BE[j,k] = p_lim_w[j] + p_lim_c[j]

    #Calculation of elastic stiffness of springs

    #Values of Menard stiffness

    k_h = [0] * len(Depth_div)  #Initialize zero-matrix

```

```
    alpha_r_gravel = 1/4
    alpha_r_clay = 2/3

    #Rheological constant per layer
    alpha_r = [alpha_r_gravel, alpha_r_clay, alpha_r_clay]

    R_0 = 0.3

    for i in range(len(Depth_div)):
        if D < 0.3:
            k_h[i] = 1/ (1/(3*E_m[i]) * (1.3*R_0*(2.65*Radius/R_0)**
    alpha_r[i]+alpha_r[i]*Radius))

        elif D>= 0.3:
            k_h[i] = 1/ (2*Radius/E_m[i] * (4*2.65**alpha_r[i]+3*alpha_r[
    i])/18)


### CREATING GEOMETRY ####


    #Create pile geometry and partitions

    if FP_it[it] == 0:
        range_piles = range(1, len(x_pile))

    else:
        range_piles = range(len(x_pile))

    for i in range_piles:    #Looping over the amount of pile rows

        #Creating geometry for first pile row
        Pile = f'Pile_{(i+1)}'

        s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
    sheetSize=5.0)
        g, v, d, c = s1.geometry, s1.vertices, s1.dimensions, s1.
    constraints
        s1.setPrimaryObject(option=STANDALONE)

####################################

        #Checking if pile is inclined
        if x_inc[i] == 0:

            #Creating geometry for long piles
            if x_short[i] == 0:
                s1.Line(point1=(x_pile[i], pile_top), point2=(x_pile[i],-
    L))
            #Creating geometry for short piles
            else:
                s1.Line(point1=(x_pile[i], pile_top), point2=(x_pile[i],-
    L_P2))
        #Creating geometry for inclined piles
        elif x_inc[i] == 1:
            s1.Line(point1=(x_pile[i], pile_top), point2=(x_pile[i]-L/
    inc_pile,-L))
```

```
480
481         #s1.VerticalConstraint(entity=g[2], addUndoState=False)
482         p = mdb.models['Model-1'].Part(name=Pile, dimensionality=
    TWO_D_PLANAR,
483             type=DEFORMABLE_BODY)
484         p = mdb.models['Model-1'].parts[Pile]
485         p.BaseWire(sketch=s1)
486         s1.unsetPrimaryObject()
487         p = mdb.models['Model-1'].parts[Pile]
488
489         del mdb.models['Model-1'].sketches['__profile__']
490
491         print(f"Geometry for {Pile} created")
492
493         #Create datum point
494
495         if x_short[i] == 0:
496             for j in range(ndz):
497
498                 z = -(dz/2 + j*dz)     #z-location for itteration
499
500                 p = mdb.models['Model-1'].parts[Pile]
501
502 ################################
503                 if x_inc[i] == 0:
504                     p.DatumPointByCoordinate(coords=(x_pile[i], z, 0.0))
505                 elif x_inc[i] ==1:
506                     p.DatumPointByCoordinate(coords=(x_pile[i]+(z/
    inc_pile), z, 0.0))
507
508     #Partition the Pile and creating nodes for spring connection
509
510             for k in range(ndz):
511
512                 p = mdb.models['Model-1'].parts[Pile]
513                 e, v, d = p.edges, p.vertices, p.datums
514                 p.PartitionEdgeByPoint(edge=e[k], point=d[k+2])
515
516             print(f"Partitioning of long {Pile} successful")
517
518         if x_short[i] == 1:
519             for j in range(ndz_P2):
520
521                 z = -(dz/2 + j*dz)     #z-location for itteration
522
523                 p = mdb.models['Model-1'].parts[Pile]
524
525 ################################
526
527                 p.DatumPointByCoordinate(coords=(x_pile[i], z, 0.0))
528
529     #Partition the Pile and creating nodes for spring connection
530
531             for k in range(ndz_P2):
532
533                 p = mdb.models['Model-1'].parts[Pile]
534                 e, v, d = p.edges, p.vertices, p.datums
```

```
                p.PartitionEdgeByPoint(edge=e[k], point=d[k+2])

            print(f"Partitioning of short {Pile} successful")

    #Create set of the pile
        p = mdb.models['Model-1'].parts[Pile]
        e = p.edges
        all_edges = p.edges[:]
        all_nodes = p.nodes[:]
        name_pile_set = f'Set_{i+1}'
        set_name = name_pile_set
        p.Set(edges=all_edges, name=set_name)

        print(f"Sets created:{set_name}")

    #Assign material and set rotation
        p = mdb.models['Model-1'].parts[Pile]
        region = p.sets[set_name]
        p = mdb.models['Model-1'].parts[Pile]

        p.SectionAssignment(region=region, sectionName=Section, offset
    =0.0,
            offsetType=MIDDLE_SURFACE, offsetField='',
            thicknessAssignment=FROM_SECTION)

        p = mdb.models['Model-1'].parts[Pile]
        region=p.sets[set_name]
        p = mdb.models['Model-1'].parts[Pile]
        p.assignBeamSectionOrientation(region=region, method=N1_COSINES,
    n1=(x_pile[i], pile_top, -1.0))

        print(f"Section assigned for {Pile}")

        if FP_it[it] == 0:
            print(f"No front row pile")

    #Adding pile to assembly
        a = mdb.models['Model-1'].rootAssembly
        a.DatumCsysByDefault(CARTESIAN)
        p = mdb.models['Model-1'].parts[Pile]
        a.Instance(name=f'Pile_{i+1}', part=p, dependent=ON)

        print(f"{Pile} added to assembly")


    #Meshing

        elemType1 = mesh.ElemType(elemCode=B23, elemLibrary=STANDARD)
        p.setElementType(regions=region, elemTypes=(elemType1, ))
        p.seedPart(size=meshsize_pile, deviationFactor=0.1, minSizeFactor
    =0.1)
        p.generateMesh()

        print(f"Mesh created for {Pile}, size={meshsize_pile}")

        print(f"Datum points for {Pile} created")

```

```
589         #print(f"Node coordinates in {Pile}")
590         #for node in p.nodes:
591             #print(f"Node label: {node.label}, Coordinates: {node.
    coordinates}")
592 
593     #Creating fixed BC at base of pile
594 
595         v1 = a.instances[Pile].vertices
596 
597         if x_inc[i] == 0:
598             if x_short[i] == 0:
599                 Pile_base_vertice = v1.findAt(((x_pile[i], -L, 0.0),))
600             else:
601                 Pile_base_vertice = v1.findAt(((x_pile[i], -L_P2, 0.0),))
602         elif x_inc[i] ==1:
603             Pile_base_vertice = v1.findAt(((x_pile[i]-L/inc_pile, -L,
    0.0),))
604 
605         region = regionToolset.Region(vertices=Pile_base_vertice)
606         mdb.models['Model-1'].DisplacementBC(name=f'BC_Bot_P{i+1}',
    createStepName='Initial',
607             region=region, u1=U1_PB, u2=U2_PB, ur3=U3_PB, amplitude=UNSET
    ,
608             distributionType=UNIFORM, fieldName='', localCsys=None)
609 
610     #Creating dummy spring at first spring position for each pile
611         name_spring = f'Spring_Pile_{i+1}_Dummy'
612 
613         a = mdb.models['Model-1'].rootAssembly
614         v1 = a.instances[Pile].vertices
615 
616         if x_inc[i] == 0:
617             verts1 = v1.findAt(((x_pile[i], z_depth[1], 0.0), ))
618         elif x_inc[i] ==1:
619             verts1 = v1.findAt(((x_pile[i]+z_depth[1]/inc_pile, z_depth
    [1], 0.0), ))
620 
621         region=regionToolset.Region(vertices=verts1)
622         mdb.models['Model-1'].rootAssembly.engineeringFeatures.
    SpringDashpotToGround(
623         name=name_spring, region=region, orientation=None, dof=1,
624         springBehavior=ON, springStiffness=1, dashpotBehavior=OFF,
625         dashpotCoefficient=0.0)
626 
627 
628     #Create timber deck - connecting piles with a beam element
629 
630 
631     num_tie = 0 #Introducing first name of tie
632 
633     for i in range(len(x_pile)-1):
634 
635         Name_Deck = f'T_Deck_{i+1}'
636 
637         s1 = mdb.models['Model-1'].ConstrainedSketch(name='__profile_Deck
    ', sheetSize=5.0)
```

```
638         g, v, d, c = s1.geometry, s1.vertices, s1.dimensions, s1.
    constraints
639         s1.setPrimaryObject(option=STANDALONE)
640         s1.Line(point1=(x_pile[i], pile_top), point2=(x_pile[i+1],
    pile_top))
641         #s1.VerticalConstraint(entity=g[2], addUndoState=False)
642         p = mdb.models['Model-1'].Part(name=Name_Deck, dimensionality=
    TWO_D_PLANAR,
643             type=DEFORMABLE_BODY)
644         p = mdb.models['Model-1'].parts[Name_Deck]
645         p.BaseWire(sketch=s1)
646         s1.unsetPrimaryObject()
647         p = mdb.models['Model-1'].parts[Name_Deck]
648         del mdb.models['Model-1'].sketches['__profile_Deck']
649 
650     #Partition the Timber deck
651 
652         if i == 0:
653             if u_partition_deck < 1:
654                 e, v, d = p.edges, p.vertices, p.datums
655                 p.PartitionEdgeByParam(edges=e[0], parameter=
    u_partition_deck)
656                 print(f"Partitioning of {Name_Deck} successful")
657             else:
658                 print("Timber deck is not partitioned")
659 
660             if u_partition_load < 1:
661                 e, v, d = p.edges, p.vertices, p.datums
662                 p.PartitionEdgeByParam(edges=e[0], parameter=
    u_partition_load)
663                 print(f"Partitioning of {Name_Deck} for load application
    successful")
664             else:
665                 print(f"Timber deck is not partitioned for load
    application")
666 
667     #Create set of the Timber deck
668         p = mdb.models['Model-1'].parts[Name_Deck]
669         e = p.edges
670         all_edges = p.edges[:]
671         all_nodes = p.nodes[:]
672         set_name = f'Set_{i+1}'
673         p.Set(edges=all_edges, name=set_name)
674         print(f"Sets created:{set_name}")
675 
676     #Assign material and set orientation of beam
677         p = mdb.models['Model-1'].parts[Name_Deck]
678         region = p.sets[set_name]
679         p = mdb.models['Model-1'].parts[Name_Deck]
680         p.SectionAssignment(region=region, sectionName=Section_Deck,
    offset=0.0,
681             offsetType=MIDDLE_SURFACE, offsetField='',
682             thicknessAssignment=FROM_SECTION)
683         p = mdb.models['Model-1'].parts[Name_Deck]
684         region=p.sets[set_name]
685         p = mdb.models['Model-1'].parts[Name_Deck]
```

```
686         p.assignBeamSectionOrientation(region=region, method=N1_COSINES,
    n1=(0.0, 0.0, -1.0))
687         print(f"Section assigned for {Name_Deck}")
688 
689     #Adding Timber deck to assembly
690         a = mdb.models['Model-1'].rootAssembly
691         p = mdb.models['Model-1'].parts[Name_Deck]
692         a.Instance(name=Name_Deck, part=p, dependent=ON)
693 
694         print(f"{Name_Deck} added to assembly")
695 
696     #Meshing Timber deck
697         p.seedPart(size=meshsize_pile, deviationFactor=0.1, minSizeFactor
    =0.1)
698         p.generateMesh()
699 
700         print(f"Mesh created for {Name_Deck}, size={meshsize_pile}")
701 
702 
703 ### ADDING LOADS TO MODEL ###
704 
705 
706     #Adding lineload from wall pressure, vertical and horizontal
707 
708         name_load_v = f'Wall-pressure_{i+1}'
709         name_load_H = f'FH-Wall_{i+1}'
710 
711 
712         mdb.models['Model-1'].ExpressionField(name='AnalyticalField_wall
    ', localCsys=None,
713             description='Load from wall on timber deck', expression=f'{
    sigma_wall_max}-{delta_sigma_wall}*X/{B_eff}')
714         a = mdb.models['Model-1'].rootAssembly
715         s1 = a.instances[Name_Deck].edges
716         edge_wall = s1.findAt(((x_pile[i], pile_top, 0),))
717         region = regionToolset.Region(edges=edge_wall)
718         mdb.models['Model-1'].LineLoad(name=name_load_v, createStepName='
    Step-1',
719             region=region, comp1=0, comp2=-1.0, distributionType=FIELD,
    field='AnalyticalField_wall')
720 
721         mdb.models['Model-1'].LineLoad(name=name_load_H, createStepName='
    Step-1',
722             region=region, comp1=q_h_wall)
723 
724 ### CONNECTING TIMBER DECK TO PILES
725 
726 
727     #Creating tie between timber deck and pile
728 
729         Pile1 = f'Pile_{(i+1)}'
730         Pile2 = f'Pile_{(i+2)}'
731         num_tie = num_tie+1
732 
733         if i == 0 and FP_it[it] == 1:
734 
735             #Finding tie point for TD, first pile
```

```
            a = mdb.models['Model-1'].rootAssembly
            v1_Deck = a.instances[Name_Deck].vertices
            region1 = v1_Deck.findAt(((x_pile[i], pile_top, 0),))
            region_Deck=regionToolset.Region(vertices=region1)

            #Finding tie for first pile
            v1_Pile = a.instances[Pile1].vertices
            verts_Pile = v1_Pile.findAt(((x_pile[i], pile_top, 0),))
            region_Pile=regionToolset.Region(vertices=verts_Pile)

            #Creating tie for first pile to deck
            mdb.models['Model-1'].Tie(name=f'Tie-{num_tie}', main=
region_Deck, secondary=region_Pile,
                positionToleranceMethod=COMPUTED, adjust=ON, tieRotations
=OFF ,#SPECIFY HINGE HERE
                constraintRatioMethod=SPECIFIED, constraintRatio=0.0,
thickness=ON) #SET RATIO TO 0 FOR HINGE

            num_tie = num_tie+1

        if i == 0:

            #Finding tie for second pile
            v2_Pile = a.instances[Pile2].vertices
            verts_Pile2 = v2_Pile.findAt(((x_pile[i+1], pile_top, 0),))
            region_Pile2=regionToolset.Region(vertices=verts_Pile2)

            #Finding tie for TD, second pile
            v1_Deck = a.instances[Name_Deck].vertices
            region2 = v1_Deck.findAt(((x_pile[i+1], pile_top, 0),))
            region_Deck2=regionToolset.Region(vertices=region2)

            #Creating tie for second pile to deck
            mdb.models['Model-1'].Tie(name=f'Tie-{num_tie}', main=
region_Deck2, secondary=region_Pile2,
                positionToleranceMethod=COMPUTED, adjust=ON, tieRotations
=OFF ,#SPECIFY HINGE HERE
                constraintRatioMethod=SPECIFIED, constraintRatio=0.0,
thickness=ON) #SET RATIO TO 0 FOR HINGE


        else:
            name_Deck_prev = f'T_Deck_{i}'

            #Finding node for previous Deck
            v1_Deck_prev = a.instances[name_Deck_prev].vertices
            verts_Deck_prev = v1_Deck_prev.findAt(((x_pile[i], pile_top,
0),))
            region_Deck_prev = regionToolset.Region(vertices=
verts_Deck_prev)

            #Finding node for current Deck
            v1_Deck = a.instances[Name_Deck].vertices
            verts_Deck1 = v1_Deck.findAt(((x_pile[i], pile_top, 0),))
            region_Deck1 = verts_Deck1=regionToolset.Region(vertices=
verts_Deck1)

```

```
784             #Creating rigid connection between current and previous Deck
    instance
785             mdb.models['Model-1'].Tie(name=f'Tie-{num_tie}', main=
    region_Deck_prev, secondary=region_Deck1,
786                 positionToleranceMethod=COMPUTED, adjust=ON, tieRotations
    =ON ,#SPECIFIED RIGID CONNECTION HERE
787                 thickness=ON)
788 
789             num_tie = num_tie +1
790 
791             #Finding tie for second pile
792             v2_Pile = a.instances[Pile2].vertices
793             verts_Pile2 = v2_Pile.findAt(((x_pile[i+1], pile_top, 0),))
794             region_Pile2=regionToolset.Region(vertices=verts_Pile2)
795 
796             #Finding tie for TD, second pile
797             region2 = v1_Deck.findAt(((x_pile[i+1], pile_top, 0),))
798             region_Deck2=regionToolset.Region(vertices=region2)
799 
800             #Creating tie for second pile to deck
801             mdb.models['Model-1'].Tie(name=f'Tie-{num_tie}', main=
    region_Deck2, secondary=region_Pile2,
802                 positionToleranceMethod=COMPUTED, adjust=ON, tieRotations
    =OFF ,#SPECIFY HINGE HERE
803                 constraintRatioMethod=SPECIFIED, constraintRatio=0.0,
    thickness=ON) #SET RATIO TO 0 FOR HINGE
804 
805     #Adding vertical load on top of the front of timber deck and back
    pile
806 
807     a = mdb.models['Model-1'].rootAssembly
808     v1 = a.instances['T_Deck_1'].vertices
809     verts1 = v1.findAt(((0, pile_top, 0.0), ))
810     region = regionToolset.Region(vertices=verts1)
811     mdb.models['Model-1'].ConcentratedForce(name='FV_Pile-1',
    createStepName='Step-1',
812         region=region, cf2=F_p_max, distributionType=UNIFORM, field='',
813         localCsys=None)
814 
815     name_pile_end = f'Pile_{len(x_pile)}'
816 
817     v1 = a.instances[name_pile_end].vertices
818     verts1 = v1.findAt(((B_wall, pile_top, 0.0), ))
819     region = regionToolset.Region(vertices=verts1)
820     mdb.models['Model-1'].ConcentratedForce(name='FV_Pile-end',
    createStepName='Step-1',
821         region=region, cf2=F_p_min, distributionType=UNIFORM, field='',
822         localCsys=None)
823 
824 
825     #Creating gravity loading
826     mdb.models['Model-1'].Gravity(name='Gravity', createStepName='Step
    -1',
827         comp2=-1.0, distributionType=UNIFORM, field='')
828 
829 
830 ### CREATING JOB FOR SIMULATION
```

```
831
832
833     #Creating the job
834
835     mdb.Job(name=name_job, model='Model-1', description='', type=ANALYSIS
    ,
836             atTime=None, waitMinutes=0, waitHours=0, queue=None, memory
    =90,
837             memoryUnits=PERCENTAGE, getMemoryFromAnalysis=True,
838             explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE,
    echoPrint=ON,
839             modelPrint=ON, contactPrint=OFF, historyPrint=OFF,
    userSubroutine='',
840             scratch='', resultsFormat=ODB, numThreadsPerMpiProcess=1,
841             multiprocessingMode=DEFAULT, numCpus=1, numGPUs=0)
842
843     mdb.models['Model-1'].fieldOutputRequests['F-Output-1'].setValues(
    variables=(
844             'S','MISES','U','RF','UR','CF','SF','CSTRESS','CFORCE','GRAV
    '))
845
846     #Write the input file to insert spring stiffness to
847     mdb.jobs[name_job].writeInput(consistencyChecking=OFF)
848
849     #Create job running analysis from input file
850
851     name_job_inp = f'{name_job}_inp'
852
853     name_inp = f'{name_job}.inp'
854
855     mdb.JobFromInputFile(name=name_job_inp,
856             inputFileName=f'{cwd}\\{name_inp}', type=ANALYSIS,
857             atTime=None, waitMinutes=0, waitHours=0, queue=None, memory
    =90,
858             memoryUnits=PERCENTAGE, getMemoryFromAnalysis=True,
859             explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE,
860             userSubroutine='', scratch='', resultsFormat=ODB,
861             numThreadsPerMpiProcess=1, multiprocessingMode=DEFAULT,
    numCpus=1,
862             numGPUs=0)
863
864
865 ### CREATING SPRING PROPERTIES ###
866
867
868     #Create spring properties to be paste into the model
869
870     name_file = "Spring-properties.txt"
871
872     open(name_file, "w")
873
874     count = 0       #Initiating counting for total number of springs
875
876     sigma_Plim = sigmap_BE
877
878     Top_spring = int(-Depth_div[0]*10)-1
879
```

```
    if FP_it[it] == 0:
        sigma_Plim[:,0] = 0.0001

    for i in range_piles:

        if x_short[i] == 0 and include_fill == 1:
            inner_range = range(ndz)

        elif x_short[i] == 1 and include_fill == 1:
            inner_range = range(ndz_P2)

        elif x_short[i] == 0 and include_fill == 0:
            inner_range = range(Top_spring , ndz)

        elif x_short[i] == 1 and include_fill == 0:
            inner_range = range(Top_spring, ndz_P2)

        else:
            print("First spring is not defined correctly")

        for j in inner_range:

            for depth, stiffness in zip(Depth_div, k_h):
                if z_depth[j] >= depth:
                    Elastic = stiffness

                    break

            f = open(name_file, "a")

            count = count+1

            k = Elastic*dz*D

            P_lim = sigma_Plim[j,i]*dz*D  #Plastic limit per spring [kN]

            x = P_lim / k

            #Spring name at z_depth j
            name_spring = f'Spring_P{i+1}_{j+1}'

            #First row, Spring elset and name
            row_1 = f'*Spring, elset={name_spring}-spring, nonlinear\n'

            #Second input, DOF
            row_2 = '1\n'

            #Third input, NL properties
            row_3 = f'-{P_lim*1.001},-{x*1000000}\n'
            row_4 = f'-{P_lim},-{x}\n'
            row_5 = f'{P_lim},{x}\n'
            row_6 = f'{P_lim*1.001},{x*1000000}\n'

            #Fourth input, element assignment
            row_7 = f'*Element, type=Spring1, elset={name_spring}-spring\
n'

```

```
            #Input for last row in Spring properties section
            if i == len(x_pile)-1 and j == ndz-1:
                row_8 = f'{count}, Pile_{i+1}.{j+2}'
            else:
                row_8 = f'{count}, Pile_{i+1}.{j+2}\n'

            f.write(row_1)
            f.write(row_2)
            f.write(row_3)
            f.write(row_4)
            f.write(row_5)
            f.write(row_6)
            f.write(row_7)
            f.write(row_8)
            f.close()

        print(f'Spring stiffness for Pile_{i+1} added in {name_file}')

    # Inserting spring properties in the job-file

    with open(name_inp, 'r') as file:
        main_text = file.read()

    with open(name_file, 'r') as file:
        replacement_text = file.read()

    repl_1 = '*Spring, elset=Spring_Pile_1_Dummy-spring'
    repl_12 = '*Spring, elset=Spring_Pile_2_Dummy-spring'
    repl_13 = '*Spring, elset=Spring_Pile_3_Dummy-spring'
    repl_14 = '*Spring, elset=Spring_Pile_4_Dummy-spring'
    repl_15 = '*Spring, elset=Spring_Pile_5_Dummy-spring'

    repl_2 = "1"
    repl_3 = "1."

    repl_4 = '*Element, type=Spring1, elset=Spring_Pile_1_Dummy-spring'
    repl_42 = '*Element, type=Spring1, elset=Spring_Pile_2_Dummy-spring'
    repl_43 = '*Element, type=Spring1, elset=Spring_Pile_3_Dummy-spring'
    repl_44 = '*Element, type=Spring1, elset=Spring_Pile_4_Dummy-spring'
    repl_45 = '*Element, type=Spring1, elset=Spring_Pile_5_Dummy-spring'

    repl_5 = '1, Pile_1.3'
    repl_52 = '2, Pile_2.3'
    repl_53 = '3, Pile_3.3'
    repl_54 = '4, Pile_4.3'
    repl_55 = '5, Pile_5.3'

    repl_row1 = f'{repl_1}\n{repl_2}\n{repl_3}\n{repl_4}\n{repl_5}'
    repl_row2 = f'{repl_12}\n{repl_2}\n{repl_3}\n{repl_42}\n{repl_52}'
    repl_row3 = f'{repl_13}\n{repl_2}\n{repl_3}\n{repl_43}\n{repl_53}'
    repl_row4 = f'{repl_14}\n{repl_2}\n{repl_3}\n{repl_44}\n{repl_54}'
    repl_row5 = f'{repl_15}\n{repl_2}\n{repl_3}\n{repl_45}\n{repl_55}'

    if FP_it[it] == 1:

        text_to_replace = f'{repl_row1}\n{repl_row2}\n{repl_row3}\n{
repl_row4}\n{repl_row5}'
```

```
992
993     elif FP_it[it] == 0:
994
995         repl_52 = '1, Pile_2.3'
996         repl_53 = '2, Pile_3.3'
997         repl_54 = '3, Pile_4.3'
998         repl_55 = '4, Pile_5.3'
999
1000        repl_row2 = f'{repl_12}\n{repl_2}\n{repl_3}\n{repl_42}\n{repl_52
    }'
1001        repl_row3 = f'{repl_13}\n{repl_2}\n{repl_3}\n{repl_43}\n{repl_53
    }'
1002        repl_row4 = f'{repl_14}\n{repl_2}\n{repl_3}\n{repl_44}\n{repl_54
    }'
1003        repl_row5 = f'{repl_15}\n{repl_2}\n{repl_3}\n{repl_45}\n{repl_55
    }'
1004
1005        text_to_replace = f'{repl_row2}\n{repl_row3}\n{repl_row4}\n{
    repl_row5}'
1006
1007    updated_text = main_text.replace(text_to_replace, replacement_text)
1008
1009    with open(name_inp, 'w') as file:
1010        file.write(updated_text)
1011
1012    print(f"Text replacement completed successfully!\n {text_to_replace}\
    n is replaced with {name_file}.")
1013
1014    #Submit job for analysis
1015    mdb.jobs[name_job_inp].submit(consistencyChecking=OFF)
1016    mdb.jobs[name_job_inp].waitForCompletion()
1017
1018
1019 ### POST PROCESSING ###
1020
1021
1022    #Name of database
1023    odb_name = f'{name_job_inp}.odb'
1024
1025    #Open database
1026    odb_path = f'{cwd}/{odb_name}'
1027    odb = visualization.openOdb(path=odb_path)
1028
1029    #Plot von Mises stresses on Deformed shape
1030
1031    session_viewport = session.viewports['Viewport: 1']
1032
1033    #Setting viewport
1034    session_viewport.setValues(displayedObject=odb)
1035    session_viewport.view.fitView()
1036    session_viewport.odbDisplay.display.setValues(plotState=(
1037        CONTOURS_ON_DEF, ))
1038    session_viewport.odbDisplay.contourOptions.setValues(
1039        contourStyle=CONTINUOUS)
1040
1041    #Printing von-Mises stress
1042    name_image = f'S-Mises_{name_job}'
```

```
    session.printOptions.setValues(vpBackground=ON)
    session.printToFile(fileName=name_image, format=PNG, canvasObjects=(
        session.viewports['Viewport: 1'], ))

    #Plot u1-displacements on Deformed shape

    #Setting viewport
    session.viewports['Viewport: 1'].odbDisplay.setPrimaryVariable(
        variableLabel='U', outputPosition=NODAL, refinement=(COMPONENT, '
    U1'),
        )

    #Printing U1-displacements

    name_image_U1 = f'U1-Displacements_{name_job}'

    session.printOptions.setValues(vpBackground=ON)
    session.printToFile(fileName=name_image_U1, format=PNG, canvasObjects
    =(
        session.viewports['Viewport: 1'], ))

    #Saving maximum von mises stresses

        #Naming of csv-output file
    csv_path = os.path.join(cwd, 'Output_Abaqus.csv')
    csv_path_max = os.path.join(cwd, f'Abaqus_MaxStress_{Name_Output}.csv
    ')

    #Set current ODB
    odb = session.openOdb(name=odb_name)

    #Extracting stress field from model
    Stress_field = odb.steps['Step-1'].frames[-1].fieldOutputs['S']
    Displ_field = odb.steps['Step-1'].frames[-1].fieldOutputs['U']

    #Initializing list to save output data
    vonMises = []
    instances = []
    node_lab = []
    Displ1 = []
    Instance_displ = []

    #Initializing list to save output data for springs
    vonMises_springs = []

    unit = 1000 #1 for m and kPa, 1000 for mm and MPa

    #Storing stresses and corresponding instance
    for stress_value in Stress_field.values:
        if stress_value.instance is not None:
            vonMises.append(stress_value.mises)
            instances.append(stress_value.instance.name)
        else:
            vonMises_springs.append(stress_value.mises)

    #Storing displacements
```

```
    for displ_value in Displ_field.values:
        if displ_value.instance is not None:
            Displ1.append(displ_value.data[0])
            Instance_displ.append(displ_value.instance.name)

    #Changing units in data output
    Displ1 = [round((x * unit),2) for x in Displ1]
    vonMises = [round((x / unit),3) for x in vonMises]

    #Saving max stress for each pile

    if unit == 1:
        D_unit = '[m]'
        S_unit = '[kPa]'
    else:
        D_unit = '[mm]'
        S_unit = '[MPa]'

    unique_piles = set(instances)

    max_values = {pile: 0 for pile in unique_piles}

        #Storing maximum stress for each element
    for value, pile in zip(vonMises, instances):
        if value > max_values[pile]:
            max_values[pile] = value

    print(f"Maximum value for {pile}: {max_values[pile]}")

        #Writing out max stresses for elements for current itteration
    if it == 0:
        file = open(csv_path_max, 'w', newline='')
    else:
        file = open(csv_path_max, 'a', newline='')

    writer = csv.writer(file, delimiter=';')

    if it == 0:
        writer.writerow(['Job name', f'Pile diameter {D_unit}', f'Front pile [Yes/No]' , 'KA or K0' ,
                         'Inclination [x:1]', 'Plim Reduction [Yes/No]' , 'Springs in filling [Yes/No]' , f'Max stress {S_unit}', 'Element nr'])

    if K_it[it] == 0:
        K = 'K0'
    else:
        K = 'KA'

    if include_red == 0:
        red = 'No'
    else:
        red = 'Yes'

    if include_fill == 0:
        fill = 'No'
    else:
```

```
        fill = 'Yes'

    if FP_it[it] == 0:
        FP = 'No'
    else:
        FP = 'Yes'

    for pile, value in max_values.items():
        writer.writerow([ name_job, D*unit , FP , K ,
                          inc_it[it] , red , fill , value, pile ])

    print(f"Maximum stress in each pile for job {name_job} saved to {
    csv_path_max}")

    #Max stress for model
    max_stress = max(vonMises)
    max_U1 = max(Displ1)
    min_U1 = min(Displ1)

    #Storing data for writing to Excel

        #Mises stress and instances
    rows = len(vonMises)
    col = 4
    data = np.zeros((rows,col), dtype = list)

        #Displacement vector
    rows2 = len(Displ1)
    col2 = 4
    data2 = np.zeros((rows2,col2), dtype = list)

    #Storing the data for stresses etc

    data[:,0] = name_job
    data[:,1] = D*unit
    data[:,2] = instances
    data[:,3] = vonMises

    data2[:,0] = name_job
    data2[:,1] = D*unit
    data2[:,2] = Instance_displ
    data2[:,3] = Displ1


    #Writing output data to Excel

    if it == 0:
        file = open(csv_path, mode='w', newline='')
    else:
        file = open(csv_path, mode='a', newline='')

    writer = csv.writer(file, delimiter=';')
    writer.writerow(['Job Name', f'Pile Diameter {D_unit}', 'Pile', f'
    Smises {S_unit}'])

    writer.writerows(data)
    writer.writerow('')
```

```
    writer.writerow(['Job Name', f'Pile Diameter {D_unit}', 'Pile', f'U1 {D_unit}'])
    writer.writerows(data2)
    writer.writerow('')
    #writer.writerow(['Max vonMises:', max_stress, 'Max U1:', max_U1, 'Min U1:', min_U1, 'Diameter:', D*unit, 'dz:', dz])
    #writer.writerow('')

    # Append data to the CSV file
    print(f"Abaqus Outputdata for {name_job} saved to {csv_path}")

    odb.close() # Closing ODB-file

#Closing csv-file after exporting results from all itterations

file.close()

print(f'{num_rows} analysis completed.' )

print(f'min / max diameter interval: {min(D_it)} / {max(D_it)}')
```

L Case study 2 - Result tables

In this appendix all result tables from the Gothenburg Case study are presented below, for each condition, not considering, or considering the stress in the timber deck.

RNF condition stress results

Case	Max Stress [MPa]	Stress> 18MPa	5MPa <= Stress <= 18 MPa	Stress < 5 MPa
Stress_D180_FP_Yes_K0_I10	22.25	Yes	No	No
Stress_D180_FP_Yes_K0_I6	19.22	Yes	No	No
Stress_D180_FP_Yes_KA_I10	12.62	No	Yes	No
Stress_D180_FP_Yes_KA_I6	10.22	No	Yes	No
Stress_D180_FP_No_K0_I10	36.0	Yes	No	No
Stress_D180_FP_No_K0_I6	34.45	Yes	No	No
Stress_D180_FP_No_KA_I10	19.65	Yes	No	No
Stress_D180_FP_No_KA_I6	18.73	Yes	No	No
Stress_D200_FP_Yes_K0_I10	14.94	No	Yes	No
Stress_D200_FP_Yes_K0_I6	12.99	No	Yes	No
Stress_D200_FP_Yes_KA_I10	8.39	No	Yes	No
Stress_D200_FP_Yes_KA_I6	6.78	No	Yes	No
Stress_D200_FP_No_K0_I10	22.67	Yes	No	No
Stress_D200_FP_No_K0_I6	21.26	Yes	No	No
Stress_D200_FP_No_KA_I10	15.06	No	Yes	No
Stress_D200_FP_No_KA_I6	15.32	No	Yes	No
Stress_D220_FP_Yes_K0_I10	10.75	No	Yes	No
Stress_D220_FP_Yes_K0_I6	9.34	No	Yes	No
Stress_D220_FP_Yes_KA_I10	6.12	No	Yes	No
Stress_D220_FP_Yes_KA_I6	4.81	No	No	Yes
Stress_D220_FP_No_K0_I10	14.59	No	Yes	No
Stress_D220_FP_No_K0_I6	15.76	No	Yes	No
Stress_D220_FP_No_KA_I10	11.02	No	Yes	No
Stress_D220_FP_No_KA_I6	12.79	No	Yes	No
Counter		8	15	1
Compliance Global [%]		8,33	15,63	1,04
Compliance Conditon [%]		33,33	62,50	4,17

RNF Condition - Stress range including Timber deck

Case	Max Stress Piles [MPa]	Max Stress Timber deck [MPa]	Stress state Piles	Stress state Timber deck	Stress state Case
Stress_D180_FP_No_KA_I6	18.731	36.562	Failure	Failure	Failure
Stress_D200_FP_No_K0_I6	21.264	36.51	Failure	Failure	Failure
Stress_D180_FP_No_K0_I6	34.446	36.249	Failure	Failure	Failure
Stress_D200_FP_No_KA_I6	15.317	30.735	Intermediate	Failure	Failure
Stress_D220_FP_No_K0_I6	15.764	29.84	Intermediate	Failure	Failure
Stress_D180_FP_No_K0_I10	36.004	26.935	Failure	Failure	Failure
Stress_D180_FP_No_KA_I10	19.649	26.813	Failure	Failure	Failure
Stress_D220_FP_No_KA_I6	12.793	24.955	Intermediate	Failure	Failure
Stress_D200_FP_No_K0_I10	22.668	24.206	Failure	Failure	Failure
Stress_D220_FP_No_K0_I10	14.589	20.935	Intermediate	Failure	Failure
Stress_D200_FP_No_KA_I10	15.058	18.951	Intermediate	Failure	Failure
Stress_D220_FP_No_KA_I10	11.017	18.284	Intermediate	Failure	Failure
Stress_D180_FP_Yes_K0_I6	19.216	9.342	Failure	Intermediate	Failure
Stress_D180_FP_Yes_K0_I10	22.252	7.097	Failure	Intermediate	Failure
Stress_D200_FP_Yes_K0_I6	12.994	6.386	Intermediate	Intermediate	Intermediate
Stress_D180_FP_Yes_KA_I6	10.223	5.566	Intermediate	Intermediate	Intermediate
Stress_D200_FP_Yes_K0_I10	14.942	5.263	Intermediate	Intermediate	Intermediate
Stress_D220_FP_Yes_K0_I6	9.337	4.753	Intermediate	Low risk	Intermediate
Stress_D220_FP_Yes_K0_I10	10.748	4.645	Intermediate	Low risk	Intermediate
Stress_D200_FP_Yes_KA_I6	6.776	4.345	Intermediate	Low risk	Intermediate
Stress_D180_FP_Yes_KA_I10	12.624	4.127	Intermediate	Low risk	Intermediate
Stress_D220_FP_Yes_KA_I6	4.813	3.603	Low risk	Low risk	Low risk
Stress_D220_FP_Yes_KA_I10	6.12	3.136	Intermediate	Low risk	Intermediate
Stress_D200_FP_Yes_KA_I10	8.391	3.131	Intermediate	Low risk	Intermediate

NRNF condition stress results

Case	Max Stress [MPa]	Stress> 18MPa	5MPa <= Stress <= 18 MPa	Stress < 5 MPa
Stress_D180_FP_Yes_K0_I10	19.06	Yes	No	No
Stress_D180_FP_Yes_K0_I6	13.76	No	Yes	No
Stress_D180_FP_Yes_KA_I10	11.36	No	Yes	No
Stress_D180_FP_Yes_KA_I6	7.52	No	Yes	No
Stress_D180_FP_No_K0_I10	24.55	Yes	No	No
Stress_D180_FP_No_K0_I6	16.52	No	Yes	No
Stress_D180_FP_No_KA_I10	14.1	No	Yes	No
Stress_D180_FP_No_KA_I6	8.04	No	Yes	No
Stress_D200_FP_Yes_K0_I10	13.52	No	Yes	No
Stress_D200_FP_Yes_K0_I6	10.18	No	Yes	No
Stress_D200_FP_Yes_KA_I10	8.09	No	Yes	No
Stress_D200_FP_Yes_KA_I6	5.58	No	Yes	No
Stress_D200_FP_No_K0_I10	17.05	No	Yes	No
Stress_D200_FP_No_K0_I6	11.96	No	Yes	No
Stress_D200_FP_No_KA_I10	9.86	No	Yes	No
Stress_D200_FP_No_KA_I6	5.86	No	Yes	No
Stress_D220_FP_Yes_K0_I10	10.13	No	Yes	No
Stress_D220_FP_Yes_K0_I6	7.83	No	Yes	No
Stress_D220_FP_Yes_KA_I10	6.08	No	Yes	No
Stress_D220_FP_Yes_KA_I6	4.31	No	No	Yes
Stress_D220_FP_No_K0_I10	12.7	No	Yes	No
Stress_D220_FP_No_K0_I6	9.12	No	Yes	No
Stress_D220_FP_No_KA_I10	7.38	No	Yes	No
Stress_D220_FP_No_KA_I6	4.65	No	No	Yes
Counter		2	20	2
Compliance Global [%]		2,08	20,83	2,08
Compliance Condition [%]		8,33	83,33	8,33

NRNF Condition - Stress range including Timber deck

Case	Max Stress Piles [MPa]	Max Stress Timber deck [MPa]	Stress state Piles	Stress state Timber deck	Stress state Case
Stress_D180_FP_No_K0_I10	24.545	19.909	Failure	Failure	Failure
Stress_D180_FP_Yes_K0_I10	19.061	14.375	Failure	Intermediate	Failure
Stress_D200_FP_No_K0_I10	17.048	19.878	Intermediate	Failure	Failure
Stress_D180_FP_No_K0_I6	16.521	19.908	Intermediate	Failure	Failure
Stress_D220_FP_No_K0_I10	12.695	19.859	Intermediate	Failure	Failure
Stress_D200_FP_No_K0_I6	11.955	19.878	Intermediate	Failure	Failure
Stress_D220_FP_No_K0_I6	9.115	19.86	Intermediate	Failure	Failure
Stress_D180_FP_No_KA_I10	14.095	17.531	Intermediate	Intermediate	Intermediate
Stress_D180_FP_Yes_K0_I6	13.759	14.924	Intermediate	Intermediate	Intermediate
Stress_D200_FP_Yes_K0_I10	13.521	11.017	Intermediate	Intermediate	Intermediate
Stress_D180_FP_Yes_KA_I10	11.361	8.602	Intermediate	Intermediate	Intermediate
Stress_D200_FP_Yes_K0_I6	10.176	11.309	Intermediate	Intermediate	Intermediate
Stress_D220_FP_Yes_K0_I10	10.131	8.846	Intermediate	Intermediate	Intermediate
Stress_D200_FP_No_KA_I10	9.861	17.519	Intermediate	Intermediate	Intermediate
Stress_D200_FP_Yes_KA_I10	8.089	6.599	Intermediate	Intermediate	Intermediate
Stress_D180_FP_No_KA_I6	8.038	17.519	Intermediate	Intermediate	Intermediate
Stress_D220_FP_Yes_K0_I6	7.832	9.303	Intermediate	Intermediate	Intermediate
Stress_D180_FP_Yes_KA_I6	7.524	7.422	Intermediate	Intermediate	Intermediate
Stress_D220_FP_No_KA_I10	7.38	17.513	Intermediate	Intermediate	Intermediate
Stress_D220_FP_Yes_KA_I10	6.078	5.35	Intermediate	Intermediate	Intermediate
Stress_D200_FP_No_KA_I6	5.862	17.511	Intermediate	Intermediate	Intermediate
Stress_D200_FP_Yes_KA_I6	5.585	5.908	Intermediate	Intermediate	Intermediate
Stress_D220_FP_No_KA_I6	4.648	17.506	Low risk	Intermediate	Intermediate
Stress_D220_FP_Yes_KA_I6	4.308	4.89	Low risk	Low risk	Low risk

RF condition stress results

Case	Max Stress [MPa]	Stress> 18MPa	5MPa <= Stress <= 18 MPa	Stress < 5 MPa
Stress_D180_FP_Yes_K0_I10	3.37	No	No	Yes
Stress_D180_FP_Yes_K0_I6	3.79	No	No	Yes
Stress_D180_FP_Yes_KA_I10	2.96	No	No	Yes
Stress_D180_FP_Yes_KA_I6	3.43	No	No	Yes
Stress_D180_FP_No_K0_I10	17.26	No	Yes	No
Stress_D180_FP_No_K0_I6	15.47	No	Yes	No
Stress_D180_FP_No_KA_I10	15.31	No	Yes	No
Stress_D180_FP_No_KA_I6	14.16	No	Yes	No
Stress_D200_FP_Yes_K0_I10	2.64	No	No	Yes
Stress_D200_FP_Yes_K0_I6	3.25	No	No	Yes
Stress_D200_FP_Yes_KA_I10	2.31	No	No	Yes
Stress_D200_FP_Yes_KA_I6	2.94	No	No	Yes
Stress_D200_FP_No_K0_I10	14.25	No	Yes	No
Stress_D200_FP_No_K0_I6	13.84	No	Yes	No
Stress_D200_FP_No_KA_I10	12.2	No	Yes	No
Stress_D200_FP_No_KA_I6	12.54	No	Yes	No
Stress_D220_FP_Yes_K0_I10	2.08	No	No	Yes
Stress_D220_FP_Yes_K0_I6	2.8	No	No	Yes
Stress_D220_FP_Yes_KA_I10	1.82	No	No	Yes
Stress_D220_FP_Yes_KA_I6	2.52	No	No	Yes
Stress_D220_FP_No_K0_I10	11.12	No	Yes	No
Stress_D220_FP_No_K0_I6	12.26	No	Yes	No
Stress_D220_FP_No_KA_I10	9.18	No	Yes	No
Stress_D220_FP_No_KA_I6	10.97	No	Yes	No
Counter		0	12	12
Compliance Global [%]		0	12,5	12,5
Compliance Case [%]		0	50	50

RF condition - Stress range including Timber deck

Case	Max Stress Piles [MPa]	Max Stress Timber deck [MPa]	Stress state Piles	Stress state Timber deck	Stress state Case
Stress_D180_FP_No_K0_I10	17.257	32.768	Intermediate	Failure	Failure
Stress_D180_FP_No_K0_I6	15.469	43.57	Intermediate	Failure	Failure
Stress_D180_FP_No_KA_I10	15.311	26.073	Intermediate	Failure	Failure
Stress_D200_FP_No_K0_I10	14.248	23.818	Intermediate	Failure	Failure
Stress_D180_FP_No_KA_I6	14.157	37.287	Intermediate	Failure	Failure
Stress_D200_FP_No_K0_I6	13.836	37.902	Intermediate	Failure	Failure
Stress_D200_FP_No_KA_I6	12.536	31.885	Intermediate	Failure	Failure
Stress_D220_FP_No_K0_I6	12.256	32.019	Intermediate	Failure	Failure
Stress_D200_FP_No_KA_I10	12.197	19.001	Intermediate	Failure	Failure
Stress_D220_FP_No_K0_I10	11.121	21.076	Intermediate	Failure	Failure
Stress_D220_FP_No_KA_I6	10.969	26.403	Intermediate	Failure	Failure
Stress_D220_FP_No_KA_I10	9.18	18.438	Intermediate	Failure	Failure
Stress_D180_FP_Yes_K0_I6	3.787	5.909	Low risk	Intermediate	Intermediate
Stress_D180_FP_Yes_KA_I6	3.434	5.223	Low risk	Intermediate	Intermediate
Stress_D200_FP_Yes_K0_I6	3.25	5.377	Low risk	Intermediate	Intermediate
Stress_D180_FP_Yes_K0_I10	3.366	4.031	Low risk	Low risk	Low risk
Stress_D180_FP_Yes_KA_I10	2.957	3.545	Low risk	Low risk	Low risk
Stress_D200_FP_Yes_KA_I6	2.944	4.738	Low risk	Low risk	Low risk
Stress_D220_FP_Yes_K0_I6	2.798	4.883	Low risk	Low risk	Low risk
Stress_D200_FP_Yes_K0_I10	2.636	3.676	Low risk	Low risk	Low risk
Stress_D220_FP_Yes_KA_I6	2.524	4.293	Low risk	Low risk	Low risk
Stress_D200_FP_Yes_KA_I10	2.307	3.247	Low risk	Low risk	Low risk
Stress_D220_FP_Yes_K0_I10	2.079	3.421	Low risk	Low risk	Low risk
Stress_D220_FP_Yes_KA_I10	1.817	3.035	Low risk	Low risk	Low risk

NRF condition stress results

Case	Max Stress [MPa]	Stress> 18MPa	5MPa <= Stress <= 18 MPa	Stress < 5 MPa
Stress_D180_FP_Yes_K0_I10	2.69	No	No	Yes
Stress_D180_FP_Yes_K0_I6	2.81	No	No	Yes
Stress_D180_FP_Yes_KA_I10	1.9	No	No	Yes
Stress_D180_FP_Yes_KA_I6	2.53	No	No	Yes
Stress_D180_FP_No_K0_I10	5.19	No	Yes	No
Stress_D180_FP_No_K0_I6	7.26	No	Yes	No
Stress_D180_FP_No_KA_I10	4.62	No	No	Yes
Stress_D180_FP_No_KA_I6	6.46	No	Yes	No
Stress_D200_FP_Yes_K0_I10	2.03	No	No	Yes
Stress_D200_FP_Yes_K0_I6	2.32	No	No	Yes
Stress_D200_FP_Yes_KA_I10	1.57	No	No	Yes
Stress_D200_FP_Yes_KA_I6	2.1	No	No	Yes
Stress_D200_FP_No_K0_I10	4.18	No	No	Yes
Stress_D200_FP_No_K0_I6	5.86	No	Yes	No
Stress_D200_FP_No_KA_I10	3.74	No	No	Yes
Stress_D200_FP_No_KA_I6	5.22	No	Yes	No
Stress_D220_FP_Yes_K0_I10	1.6	No	No	Yes
Stress_D220_FP_Yes_K0_I6	1.95	No	No	Yes
Stress_D220_FP_Yes_KA_I10	1.32	No	No	Yes
Stress_D220_FP_Yes_KA_I6	1.77	No	No	Yes
Stress_D220_FP_No_K0_I10	3.45	No	No	Yes
Stress_D220_FP_No_K0_I6	4.83	No	No	Yes
Stress_D220_FP_No_KA_I10	3.09	No	No	Yes
Stress_D220_FP_No_KA_I6	4.32	No	No	Yes
Counter		0	5	19
Compliance Global [%]		0	5,21	19,79
Compliance Condition [%]		0	20,83	79,17

NRF Condition - Stress range including Timber deck

Case	Max Stress Piles [MPa]	Max Stress Timber deck [MPa]	Stress state Piles	Stress state Timber deck	Stress state Case
Stress_D180_FP_No_K0_I6	7.264	20.146	Intermediate	Failure	Failure
Stress_D200_FP_No_K0_I6	5.855	20.092	Intermediate	Failure	Failure
Stress_D180_FP_No_K0_I10	5.186	19.969	Intermediate	Failure	Failure
Stress_D220_FP_No_K0_I6	4.83	20.048	Low risk	Failure	Failure
Stress_D200_FP_No_K0_I10	4.182	19.926	Low risk	Failure	Failure
Stress_D220_FP_No_K0_I10	3.45	19.891	Low risk	Failure	Failure
Stress_D180_FP_No_KA_I6	6.462	17.836	Intermediate	Intermediate	Intermediate
Stress_D200_FP_No_KA_I6	5.224	17.782	Intermediate	Intermediate	Intermediate
Stress_D180_FP_No_KA_I10	4.621	17.671	Low risk	Intermediate	Intermediate
Stress_D220_FP_No_KA_I6	4.318	17.74	Low risk	Intermediate	Intermediate
Stress_D200_FP_No_KA_I10	3.736	17.628	Low risk	Intermediate	Intermediate
Stress_D220_FP_No_KA_I10	3.089	17.593	Low risk	Intermediate	Intermediate
Stress_D180_FP_Yes_K0_I6	2.812	3.473	Low risk	Low risk	Low risk
Stress_D180_FP_Yes_K0_I10	2.689	3.216	Low risk	Low risk	Low risk
Stress_D180_FP_Yes_KA_I6	2.53	3.175	Low risk	Low risk	Low risk
Stress_D200_FP_Yes_K0_I6	2.316	3.401	Low risk	Low risk	Low risk
Stress_D200_FP_Yes_KA_I6	2.097	3.095	Low risk	Low risk	Low risk
Stress_D200_FP_Yes_K0_I10	2.027	3.151	Low risk	Low risk	Low risk
Stress_D220_FP_Yes_K0_I6	1.95	3.337	Low risk	Low risk	Low risk
Stress_D180_FP_Yes_KA_I10	1.896	2.917	Low risk	Low risk	Low risk
Stress_D220_FP_Yes_KA_I6	1.773	3.027	Low risk	Low risk	Low risk
Stress_D220_FP_Yes_K0_I10	1.603	3.094	Low risk	Low risk	Low risk
Stress_D200_FP_Yes_KA_I10	1.567	2.846	Low risk	Low risk	Low risk
Stress_D220_FP_Yes_KA_I10	1.321	2.785	Low risk	Low risk	Low risk

DEPARTMENT OF ARCHITECTURE AND
CIVIL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY