AALBORG UNIVERSITY
DENMARK

DEPARTMENT OF CIVIL ENGINEERING

MASTER THESIS IN CIVIL & STRUCTURAL ENGINEERING

Implementation of Parallelization in Finite Element Analysis

By Daniel Dalgaard Cramer, B.Sc. in Civil & Structural Engineering, Aalborg University

Supervised by
Associate Professor Johan CLAUSEN

June 8, 2018

AALBORG UNIVERSITY
DENMARK

MSc. Structural & Civil Engineering
Thomas Manns vej 23
9220 Aalborg East

Title:

Implementation of Parallelization in Finite Element Analysis

Theme:

Numerical Finite Element Modelling

Project period:

September 1st 2017
to June 8th 2018

Supervisor(s):

Johan Clausen

Number of pages:

45

Number of pages in appendices:

16

Abstract:

In this project a program, programmed in C# , is developed and capable of determing material parameters of soil based on some target data, supplied in form of a non-linear load-displacement curve. For the purpose of the project, the data is created with know material parameter using the non-associated Mohr-Coulomb material model. The determination of the parameters can be classified as an optimisation problem, which are very computationally intensive. For this reason parallelization is built in to the program. Initially different parallel programming models are investigated, with one being chosen as the backbone of the program.
With the programming model established, some different implementation methods are tested and evaluated. The final version of the program is tested against different sets of target data, and with the use of established error functions, the relative difference between simulated data and target data are determined. Based on a set of requirements, the simulated data is either accepted or rejected. A final investigation of the attained speedup, following the parallel implementation, is performed.

Participant(s):

Daniel Dalgaard Cramer

Preface

This thesis documents the work which serves as a prerequisite for obtaining the degree of *M.Sc.* within the programme of *Civil & Structural Engineering* under the curriculum dated 21. of June 2010. The enclosed Appendix report also serves as part of the thesis. The thesis time frame spanned the 1st of September 2017 through to 8th of June 2018.

Prerequisites for reading the report is general knowledge of soil mechanics and more in depth knowledge of numerical finite element modelling.

The author would like to thank the supervisor, Johan Clausen, for pleasant meetings and discussions during the project period. His opinions were greatly appreciated.

Reading Guide

Various references are used throughout the report by the use of numbers, i.e. the references have the format [1] or if a specific page, table or figure is referenced the format will be [1, Pages 1-2], [1, Pages Table 2.4] or [1, Figure 3.5]. The references are collected in the bibliography at the very end of the report, preceding the Appendices.

Labelling of figures and tables are done in accordance to the corresponding chapter for which they are located.

Contents

Chapter 1

Introduction

Several different Civil Engineering sectors rely on the accurate prediction of how structures behave during loading for the safe estimation of how the structures should be constructed.

In the past, analytical solutions have been developed for many varieties of problems and loading's, but as structures and development project have grown in both complexity and size, these analytical solution comes short because of advanced geometries or because analytical solutions simply does not exists.

In the recent years a huge development of sophisticated software which has had it main focus in aiding designers and engineers, in solving the problem and challenges of tomorrow.

In particular within the field of geotechnical engineering, the use of computer based simulation of soil behaviour is of paramount importance, since even a seemingly simple geometry can amount to complex analytical equations, if they even exist, and can be impossible to solve. One example of the advances in geotechnical engineering can be seen on Figure 1.1 which is a very advanced problem with many facets.

The methodology and design is now based on well established design methodology and verification as opposed to the past where the design relied merely on the experience of the engineers. For this reason a large number of commercial software is developed to satisfy

FIGURE 1.1: Example of an advanced structure involving different engineering disciplines.

the need from engineers and designers, with big companies like PLAXIS, ABAQUS and COMSOL MULTIPHYSICS, offering solutions to a wide range of problems.

1.1 Finite Element Method

The software developed towards solving the increasingly difficult problems engineers face, is in most cases based upon the Finite Element Method, FEM, which is a computational technique in which an approximate solution of a boundary value problem is obtained. FEM is extremely versatile since the boundary value problem is described by differential equations, and depending on the branch of engineering, it being chemical-, structural-mechanical- or electrical engineering, it is just a matter of changing the governing differential equations to make it suit the needs of the user.

FEM is an approximation of the differential equations which is based on a spacial discretization of a domain of interest. This domain can have any form or shape, since it is approximated by elements. Simple shape can require a few elements to properly approximate the domain whereas complex geometry may require the use of different shapes and sizes of elements to properly approximate the domain. The discretization can the approximate the differential equations by means of numerical equations which finally can be solved by applying numerical methods.

The nature of FEA with the discretization of the domain implies that approximate solutions are found, when the numerical methods are applied. For this reason a refinement of the discretization will make the approximated solution converge towards a steady value. The refinement imposes, along with other factors, an overall increase in the necessary computational power which subsequently increases the time it takes to produce a result. For this reason, the development of routines with the intent to reduce the total computational time, is the focus of several research projects and companies.

1.2 Determination of material parameters

Within geotechnical engineering it is of great importance to be able deduce a soils material strength and deformation parameters. With these, load bearing capacity and settlement, which are important behaviour, can be calculated. During the years, many different methods have been developed with the aim of accurately determining the soils material parameters. Depending on the structure and the accuracy needed of the material parameters, the methods range from simple vane and CPT test, which are done in-situ, all the way up to full-scale on-site loading test and expensive laboratory experiment.

Especially the full-scale on-site loading test and laboratory experiments are of interest. Performing a full scale test, as illustrated on Figures 1.2 and 1.3, ensures that the on-site parameters are determined. In the laboratory great accuracy can be attained by carefully handling and conducting experiments to soil samples, by several different methods. One of which, the tri-axial apparatus provides excellent knowledge of the soil response during loading.

By using FEM with geotechnical engineering it is possible to model the behaviour by using one of many different material models. The choice for which, depends on the situation at hand and the engineer must be familiar with the benefits and drawback of different models to be able to apply them correctly. Many complex models exists which

FIGURE 1.2: Full scale loading test. [1]

FIGURE 1.3: Full scale loading test. [1]

are great at describing soil behaviour, but the complexity comes at a cost. A complex model requires a skilled and seasoned engineering behind the screen to understand in- and output from a model. This poses a problem for an engineering company, since it can be difficult and expensive to acquire an experienced engineer.

For the most part however, the more simple Mohr-Coulomb material model can be applied to a wide variety of soil modelling problems which will also be the material model used in this thesis.

1.3 Thesis Statement

It is within the scope of this thesis to apply some of the FEM routines which is developed for geotechnical engineering, on a problem which is extremely computational intensive. With this in mind, the main thesis statement is formulated as:

> *The focus will be on developing a Finite Element Method program which can utilise parallelization in the calculation phase.*
>
> *The thesis will put special emphasis on optimising, testing and implementing parallelization. In this regard different ways of parallelization will be investigated and compared utilising the Central Processing Unit, CPU, or the Graphics Processing Unit, GPU.*

Continuing in the geotechnical engineering field, it will be the task for the author to create a program which is able to determine material parameters for a soil body, by using the Mohr-Coulomb yield criterion. The program will be "given" data, in the form of load-displacement data, and it will then be up the program to calculate the material parameters which comes as close to the parameters for the soil body, as possible. This sort of problem can be characterised as an optimisation problem, since a large solution space is present and thus some form of evaluation of the calculated material parameters is introduced.

1.3.1 Project scope

It is within the scope of the project to apply a Mohr-Coulomb material model to a computational intensive problem. It is however not within the scope of the project to apply any other model or analytical ways of determining settlement or bearing capacity and thus the result are not compared to analytical methods. The project is purely focused on the implementation of well established theory such as the finite element method and the non-associated material model which goes into this.

Chapter 2

Parallelization

In recent years, the clock speed of new processors have stagnated as illustrated by Figure 2.1 and instead chip manufactures have made great efforts in increasing the number of transistors in the processor. [2] This increase is achieved by making the transistors smaller, and by doing do, more cores are added to the processor.

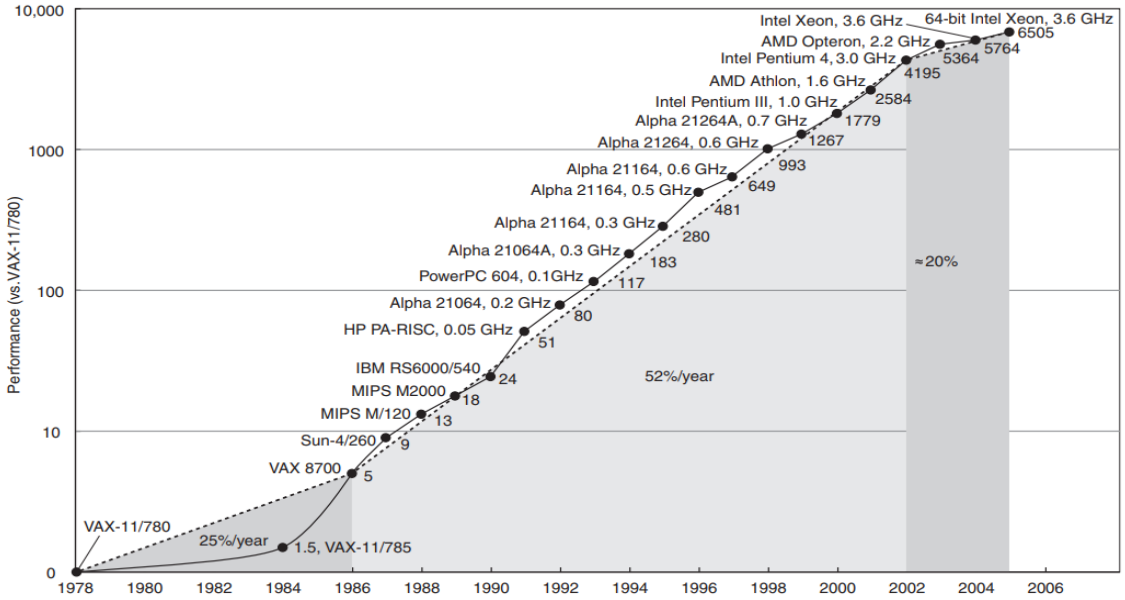

FIGURE 2.1: Evolution of core clock speeds over time. [3, Figure 1.16]

The effect of having more cores directly influence the computing power of the processor in a positive way and it must be the goal, as programmers and engineers, to take advantage of this increase in computing power. Usually programs are written in a sequential way, such that a process is only handled by one core in a multicore processing environment. This implies for example that one calculation in a loop, needs to finish before another can be started. However by taking advantage of the computational power in the other idle processors one could substantially decrease the time it would take to run through calculations. Therefore, changing the sequential loops to be performed as parallel loops, could offer a full utilisation of the computing power.

As one would find out this is however not always straight forward, since it is case dependent weather or not the implementation is actually achievable and as such a few requirements can be formulated for the operation to be performed in parallel.

Calculations must be independent Since parallelization does not guarantee any specific order of execution of loops some higher valued indices might be executed before some lower indices. This implies that each calculation must be independent of the previous.

The computer must have more than one core In order to achieve a speedup of the computation there must be more than one core available for the program to distribute tasks onto. However even if there are only one core, the parallel code will function since it operates by distributing workload to where resources are available. So even though only one core is available the parallel code is still able to run, but will do so with no apparent speedup, compared to sequential programmed loops.

The problem at hand should be "balanced" In order to utilise the full computational power of the processors, the computation should be as balanced as possible, meaning that when partitioning workload an even amount of expected work is distributed to each core. An example of uneven partitioning can be found in [4, Pages 9-12] with the associated consequences.

2.1 Parallel programming model

There are two fundamental ways to parallelize data, namely through a so called Fork / Join configuration or through loop distribution. [5]

2.1.1 Fork/Join

Using a Fork/Join parallelization, a master thread creates a so called thread pool, in which two or more threads are spawned, see Figure 2.2. This is called the fork, since it bears great resembles to the cutlery. Then the parallel pool executes the statements and commands which are enclosed in the parallel region. When all threads have finished the assigned task, a synchronisation happens at the join. Here the parallel pool is closed and the master thread which initiated to pool, continues its work.

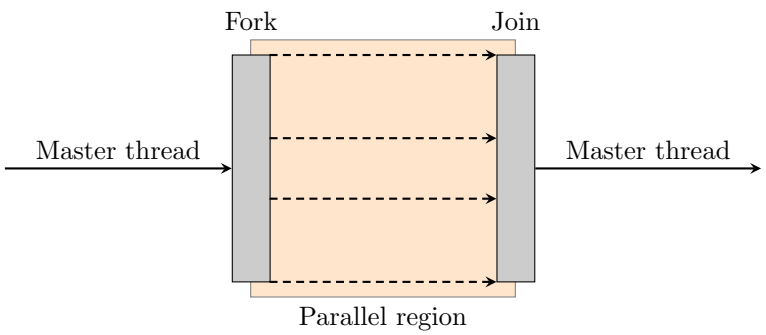

FIGURE 2.2: Balanced fork/join parallelization. The dashed lines represent work being done by a thread.

The fork/join method of parallelization is most effective when the compute load is balanced and distributed evenly. If this is not the case, and the compute load in unevenly distributed, then the whole process would need to wait until the slowest thread has finished, thus hurting compute time. An example of an unbalanced fork/join can be seen on Figure 2.3.

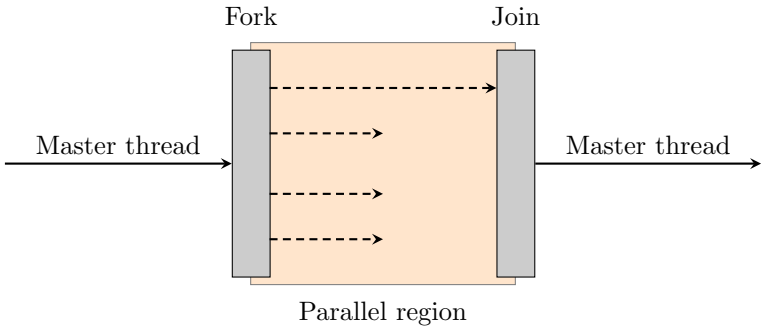

FIGURE 2.3: Unbalanced fork/join parallelization. The dashed lines represent work being done by a thread.

2.1.2 Loop distribution

When the compute load of a single iteration of a loop becomes significantly large for one thread, an option is the so called loop distribution. With this, the loop execution would be divided into two or more sections of code which would be handled by an independent thread. This is illustrated by Figure 2.4

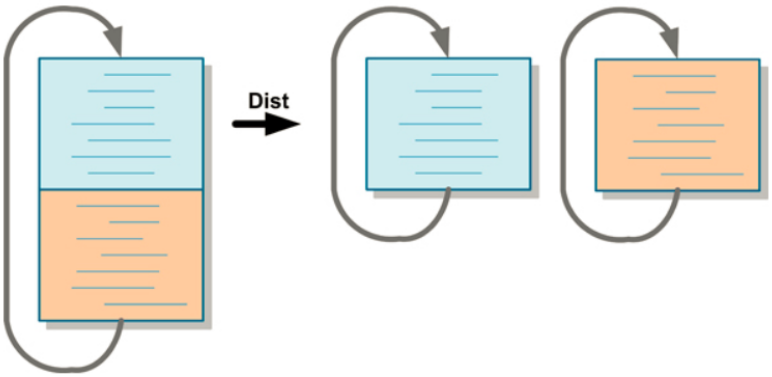

FIGURE 2.4: Loop distribution. [5, Figur 3]

For the case of loop distribution the second half of the original code can only execute when the first half of the code finished thus demanding it to wait until the first thread has finished. This is merely an issue in the first iterations, since in the second all threads will be busy doing work. When doing loop distribution, great care has to be taken as to where the loop is split. If data is produced in the first part of the code and used in the second part, this data much be communicated and synchronised to secure correct calculations.

For the project both options were considered but a final choice were made to go with the fork/join method of parallel programming.

2.2 Programming languages

Within programming many different languages exists and depending on what a program is intended to do, the languages should be chosen accordingly. Within the field of number crunching and application building a few of the many stands out and should be considered. Just to name a few, languages like PYTON and FORTRAN are highly recommended if the program is intended to do matrix and algebraic manipulations, whereas C , C++ and C# are general purpose, all-round languages used in everything from mobile apps to finite element software. It is especially the implementation of the languages in finite element software which is of importance for the author, since the main focus of this thesis is to develop a FEM program.

When dealing with programming languages, a reference to its "level" is often done, in the sense that a language can either be considered high, medium or low level. This has to do with the level of abstraction from machine code in which the languages deals with. It can be quite difficult to characterise a language to be specifically one level, since no clear guideline is established. However, by following a line of thought that lower level languages require code that resembles the architecture of the computer and higher level provides an abstraction, so that the code resembles more the structure of the task the program needs to solve, a crude distinction can be made.

2.2.1 Choice of language, C#

The choice of programming languages falls upon the C# language which is based upon a weighing of pros and cons.

Pros

C# can be considered a mid level programming language, meaning that it provides some abstraction from a computers instruction set architecture. This can be highly preferable for someone with little to no experience with programming languages other than MATLAB.

The language is strongly typed and uses a compiler, meaning that a compilation software is used to transform the code into machine code. The compiler has the additional feature that it checks and flags errors easing the development of the code. Further more it has highly optimised libraries developed specifically for implementing parallelization and adding multithreading to a program. These libraries, if not specified otherwise, handles the scaling of the degree of parallelization dynamically and can thus ensure the most efficient use of the available processors. These libraries also offer a great level of control as to what is happening within the loops and provides control of the degree of parallelization.

It has long been the authors wish to learn lower level languages, like C++ , and before doing so it is recommended to start with languages with some degree of abstraction.

One major benefit of C# is object-oriented style of programming which is the key to building programs in C# . The object is given some properties and it is then through the use of methods and actions that the object changes. The should apply well to FEM, since the soil body is the object of interest as well as the associated material parameters. Object-oriented programming offers great code reusability, since objects are defined from so called classes. It is these classes which contain all the methods used to manipulate an object.

Cons

It can be difficult to learn new material and one major disadvantage is that the author does not have much experience within medium and low level languages. For this reason it is expected that it will take some time getting use to the syntax and the programmatic structure of the language.

Wanting to work in parallel on a GPU can be facilitated through the NVIDIA CUDA software. Different versions, involving different languages has been made, but unfortunately a direct implementation into C# is not currently available which must be considered as a major drawback.

2.3 Pitfalls in parallel programming

The concepts of parallelization is simple in concept but actual implementation within a program must be done with high confidence and knowledge of the program. Without this, there is a high possibility that the implementation is done incorrectly, with the result that the program calculates incorrectly. The use of erroneous values in an engineering phase of a structure could, in the worst possible case, lead to loss of life as a result of structural failure, which would be catastrophic. With this in mind, care must be taken in implementation and the following must be considered. [6]

2.3.1 Asynchronous thread safety

When executing computations in parallel great care must be given to shared variables which takes in results handled in each thread. It is of paramount importance that the synchronisation between threads are dealt with in a way, that ensures that no two threads are trying to access the same variable at the same time.

One way this could be done is to simply lock access to a global variable when a thread is accessing it, meaning that only one thread can use and change the variable. This however introduces the problem that when the variable is locked all other concurrent tasks need to wait until the variable is free, thus affecting performance which is not desirable.

2.3.2 Cost of overhead

The initiation and termination of a parallel thread pool incurs a cost of computational time defined as overhead. This is the product of the master thread having to coordinate and distribute the work done by the parallel pool, and subsequently the termination in which synchronisation is done and the threads are shut down.

To investigate this cost, a test program has been made in Appendix A where an increase in the number of computations need to be done, is increased. The result for which can be seen on Figure 2.5.

2.3.3 Over-parallelization

Several different computation schemes involve the use of nested loops, meaning that one or more loops are enclosed within another loop. The programmer should, when trying to establish the best approach at dealing with nested loops, be careful not to invoke a new set of parallel threads within a parallel loop. However, if the nested loop is know to be computationally very expensive, or if the system for which the computations are being

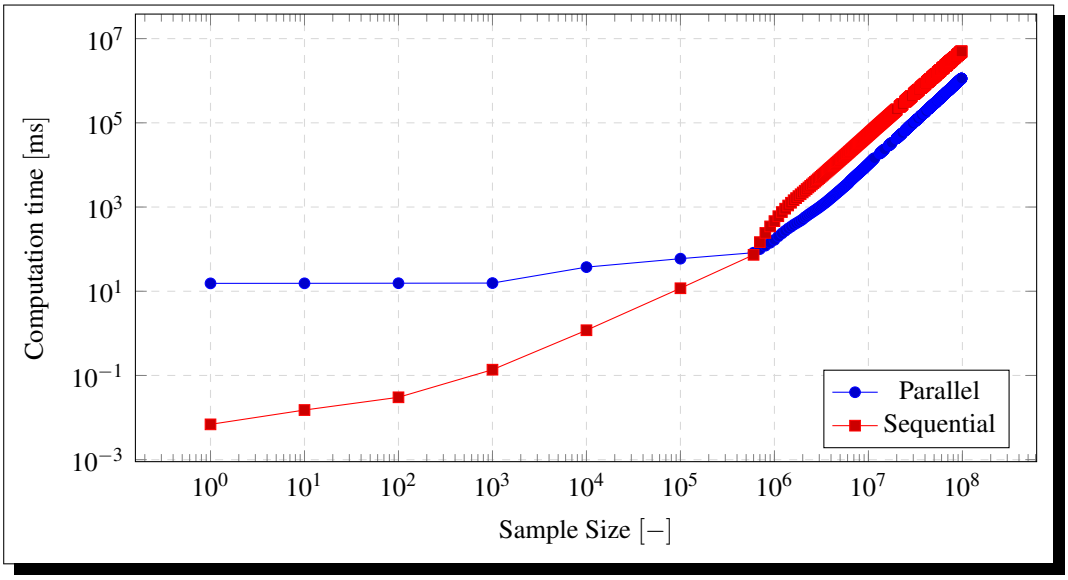

FIGURE 2.5: Study of computation times from the test example in Appendix A. Computations performed on a INTEL® CORE™ i7-4710MQ with 20 Gb of DDR3 RAM.

performed on, have enough threads to handle the nested initialisation of threads, it can be necessary to invoke nested parallel loops. It should be considered carefully, since invoking each parallel pool of threads incurs the overhead cost of starting up and partitioning work to each thread.

2.4 Theoretical speedup

The speed of a program is defined as the time that it takes to execute and provide a result. Adding more threads to a program should decrease the time it takes to execute and by doing so a speedup should be attained. The speedup must then be defined as the time it takes for the program to execute in serial, with one thread, divided by the time it take to execute in parallel.

Having a computer with a multicore processor and steadily increase the number of threads in the parallel region, cf. Figure 2.2, expecting to see an ever increasing speedup, will in many situations not satisfy expectations.

This can be explained by *Amdahl's law*, Equation (2.1), which takes into account the fraction of the code that can execute in parallel, P and the number of processors, N. [7]

$$\text{Speedup} = \frac{1}{\frac{P}{N} + (1 - P)} \tag{2.1}$$

With this, a preliminary prediction of the expected speedup can be made and further it can be used to explain results obtained with a program that has executed in parallel. FIG. 2.6 illustrates different fractions of parallel code and it can be observed that in order to obtain a relative high speedup, the fraction of code that should run in parallel should be close to, or higher than 95 %. With a lower percentage the curves flatten relatively quickly and adding more than 10 threads does not improve the speedup significantly. It can also be observed that a code that execute completely in parallel has a linear increase in speedup as more processors are added which is quite interesting.

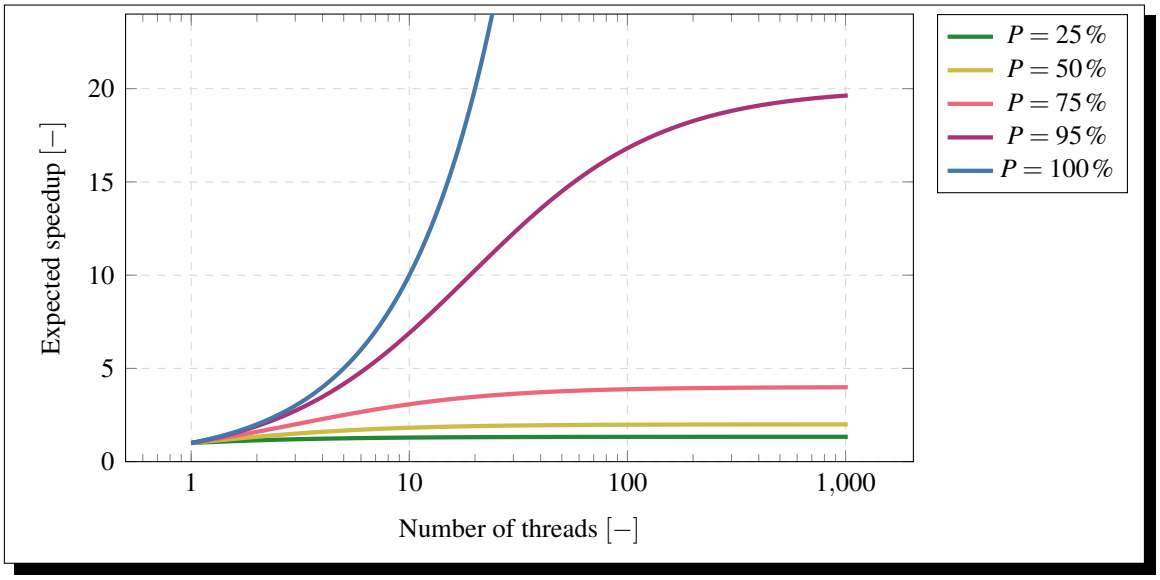

FIGURE 2.6: Expected speedup when considering the number of threads. Plot of EQ. (2.1).

2.4.1 Scalability

In EQ. (2.1) the fraction of the code that can run in parallel, P, can be expressed in total percentage of code or time. It can in many situations be an advantage to express the fraction in terms of time due to the fact that it is easy to measure the time spent in the parallel region. Another advantage of expressing the equation in terms of time, is that an increasing in the complexity of a certain problem, does not mean that written code needs changing, but it rather increases computational time.

As an example if solving numerical equations in a domain, which is approximated with x number of elements, can be done in 60 s parallel and 40 s serial time, meaning that 60 % of the time is spent in parallel. However if the same domain is approximated with a 1000 times more elements and solving the equations now takes 400 s parallel and 40 s serial time, meaning that the time spent in parallel is now approximately 93 %. This has the implication that according to EQ. (2.1) the same scaled problem is relatively more sensitive with respect to the speeding up, when adding more computational power.

Thus a guideline statement can be formulated as the following: *"Problems that increase the percentage of parallel time with their size are more scalable than problems with a fixed percentage of parallel time."* [8]

Chapter 3

Objective and Case

As mentioned in

As mentioned in Section 1.3 the main focus of the developed program will be to determine material parameters for a soil body using a non-associated Mohr-Coulomb material model which takes in 5 variables. These 5 variables define the elastic and plastic response, respectively. The variables related to elastic response are

E is the Young's modulus of the soil skeleton.

ν is Poisson's ratio of the soil skeleton.

And the 3 variables related to plastic response are

c is the cohesion of the soil.

ϕ is the friction angle of the soil

ψ is the dilation angle of the soil

For the purpose of simplification a choice were made to use the same value for the friction and dilation angle. The simplification comes from a significant decrease in the number of different combinations of the variables which needs to be tested, as well as the fact that this is common practice, when using the Mohr-Coulomb model. [9] The Mohr-Coulomb material model is further explained in Appendix D.

3.1 Data to analyse

For the program to determine the parameters for the input variables, some target test values are needed. Initially, it was thought that actual experimental data would be used, since the goal of the program was usability within an engineering company. In a company, experimental data would be provided by e.g. triaxial testing of soil and the program would determine the soil parameters from this data. However to ensure that correct values would be determined, this is not the approach which has been taken. The approach taken is that the data, for which variable parameters are to be determined, is provided by the program itself, created with a known set of parameters. This will ensure that, depending on the requirements and bounds for the fitted parameters, see Section 3.2, the program will find at least one solution of parameters which is exact and more which will deviate slightly.

For the sake of testing the program several sets of parameters are sought and listed in Table 3.1. This is done in order to investigate the robustness and efficiency of the

program, but also to see the influence different parameters of the variables have on the load-displacement curve, and to find variables which have a distinct elastic and plastic region.

Table 3.1: *Sets of parameters for input to the test program.*

	E [MPa]	ν [-]	c [kPa]	φ [deg]	ψ [deg]	Range increment
Set 1	$10-50$	$0,3$	30	20	20	10 [MPa]
Set 2	50	$0,1-0,5$	30	20	20	$0,1$ [-]
Set 3	50	$0,3$	$10-50$	20	20	10 [MPa]
Set 4	50	$0,3$	30	$20-40$	$20-40$	5 [deg]
Set 5	100	$0,1$	30	$20-40$	$20-40$	5 [deg]
Set 6	100	$0,2$	30	$20-40$	$20-40$	5 [deg]
Set 7	100	$0,3$	30	$20-40$	$20-40$	5 [deg]
Set 8	100	$0,4$	30	$20-40$	$20-40$	5 [deg]
Set 9	100	$0,1$	$10-50$	30	30	10 [MPa]
Set 10	100	$0,2$	$10-50$	30	30	10 [MPa]
Set 11	100	$0,3$	$10-50$	30	30	10 [MPa]
Set 12	100	$0,4$	$10-50$	30	30	10 [MPa]

Each of the sets of parameters, listed in Table 3.1, are plotted in Figures 3.2 to 3.13. The soil body is subjected to a deformation, which is calculated in 50 loads steps, resulting in a total vertical deformation of $u = 100\,\text{mm}$. The soil is assumed to have a total saturated weight of $\gamma_{total} = 20\,\text{kN/m}^3$.

Given the relatively simple geometry, a wide variate of different element type can be applied. For the present project, 6 noded triangular linear strain elements are employed which have two degrees of freedom pr. node. The mesh is created with a total number of degrees of freedom, $n_{dof} = 322$ and is illustrated on Figure 3.1.

3.1.1 Choice of data

As mentioned the goal is to analyse data which has both an elastic and plastic region. For the purpose of illustration in the report, pressure is taken as positive, whereas, in the program, pressure is taken as negative. Looking at the data, Figures 3.2 to 3.13, clear elastic and plastic response is evident in much of the data. However Figures 3.2 and 3.13 seems to be a bit better, since plastic response is evident in all ranges of the parameters. For this reason, Set 2 and Set 12 is chosen to be the input to the program.

3.2 Error estimation

Since the issue of determining the parameters of the constitutive model, can be considered as an optimisation problem, it is necessary to formulate some function which evaluate the response between the numerical prediction and the experimental results. For this reason the concept of error estimation is introduced and a few different methods of assessing the relative error between the calculated response and the data is introduced. The overall goal

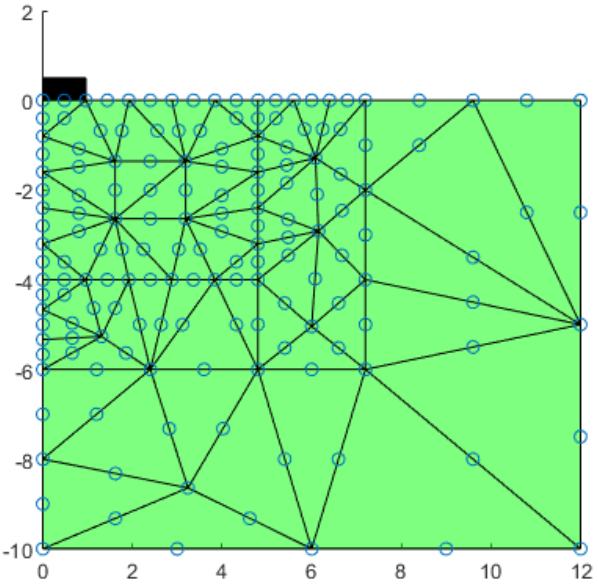

FIGURE 3.1: Initial mesh used for data analysis. Displacement is applied at the top left corner on the black box.

is to minimise the error such that

$$\text{Error}(\mathbf{x}) \rightarrow \min \tag{3.1}$$

for which $\mathbf{x}$ is the set of parameters to be optimised. It must be the goal to minimise these functions individually, and then be used as a basis for whether the obtained numerical prediction with one set of parameters are accepted or rejected based on a set of requirements and / or bound for which the error must lie within.

Function 1

The first and simplest of the error functions takes the following expression [10, Eq. (4)]

$$\text{Error}(\mathbf{x}) = \frac{1}{N}\left(\sum_{i=1}^{N}\left|U^i_{exp} - U^i_{num}\right|\right) \tag{3.2}$$

where N is the number of discrete points, U^i_{exp} is the value of the measurement i and U^i_{num} is the value of the calculation i. The function can be physically interpreted as the absolute distance between discrete points, which means it has the same units as the data.

Function 2

A further expansion of the error function, Equation (3.2), can be made [10, Eq. (5)]

$$\text{Error}(\mathbf{x}) = \frac{1}{N}\left(\sum_{i=1}^{N}\left(U^i_{exp} - U^i_{num}\right)^k\right)^{\frac{1}{k}} \tag{3.3}$$

where k is a non-zero positive value with $k = 1$ for the sum of error at every point and $k = 2$ for the least square function. The units of the estimated error, retains the units from the evaluated data.

Function 3

The least square method is the basis for the last formulation of an error function. It is adopted with modifications of 100 percentage and by adding weights to each calculation point, with a further modification of determining the average difference between the measured and calculated result [10, Eq. (6)]

$$\text{Error}(\mathbf{x}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N} w_i \left(\frac{U_{exp}^i - U_{num}^i}{U_{exp}^i}\right)^2} \times 100 \tag{3.4}$$

where w_i is the weight for the calculation at point i. This last method of error estimation can be considered particularly useful in practical engineering since it has the option of adding weight to some part of the curve. This could be relevant in two particular situations, namely the serviceability limit state and the ultimate limit state. For the serviceability limit state the first elastic part of the curve would be on interest, since it can be used to estimate the initial displacement of the soil body before yielding happens. For the ultimate limit state the last part, where plastic deformation occurs, can be used to estimate the ultimate bearing capacity of the soil.

The output from the function is unitless and thus by multiplying it with 100 the result is expressed as a percentage.

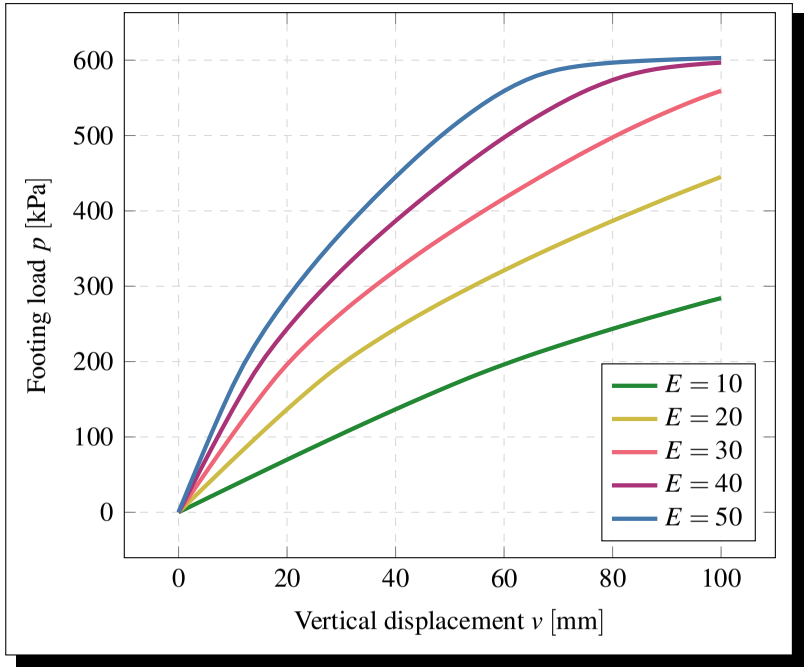

FIGURE 3.2: Set 1 with varying E and $\nu = 0.3$, $c = 30$[kPa], $\varphi = 20$[deg] and $\psi = 20$[deg]

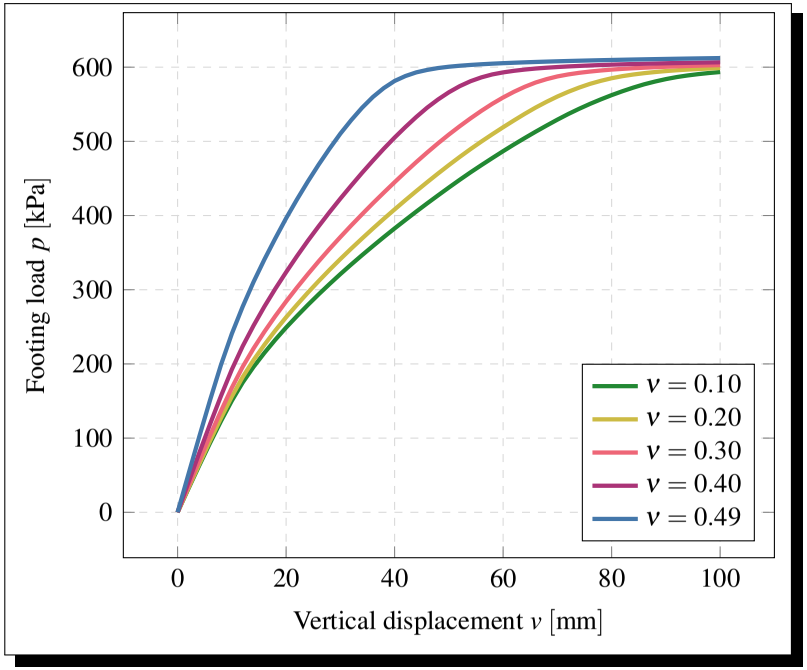

FIGURE 3.3: Set 2 with varying ν and $E = 50$[MPa], $c = 30$[kPa], $\varphi = 20$[deg] and $\psi = 20$[deg]

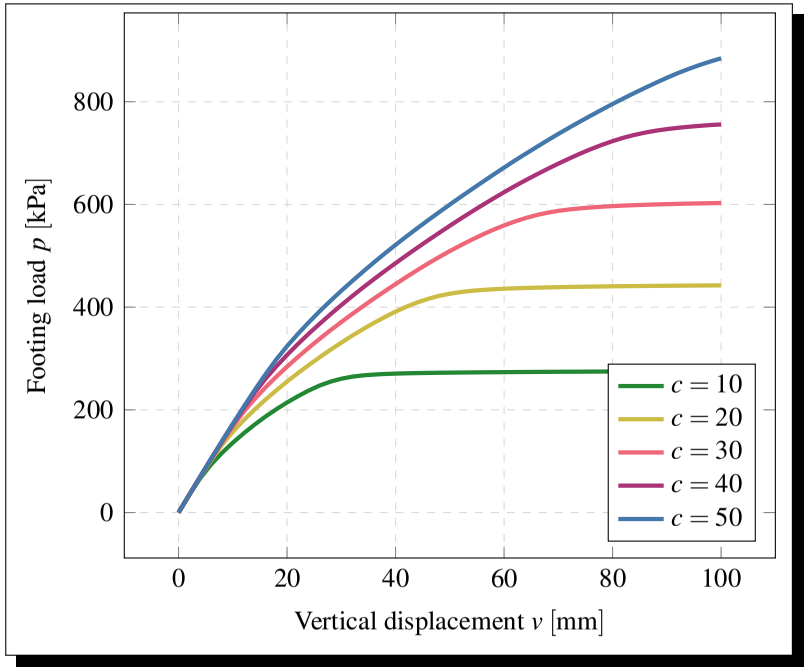

FIGURE 3.4: Set 3 with varying c and $E = 50$[MPa], $\nu = 0.3$, $\varphi = 20$[deg] and $\psi = 20$[deg]

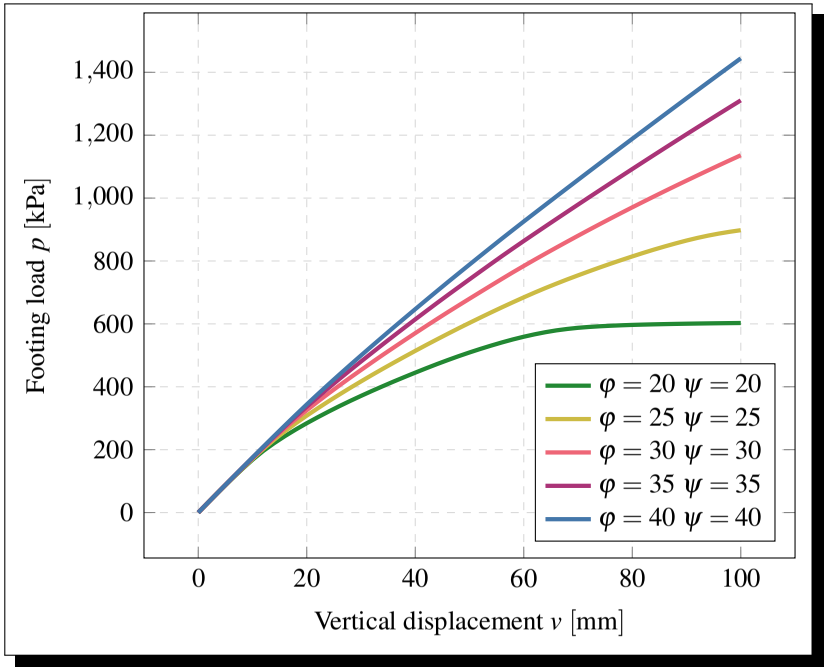

FIGURE 3.5: Set 4 with varying φ and ψ and $E = 50$[MPa], $\nu = 0.3$ and $c = 30$[kPa]

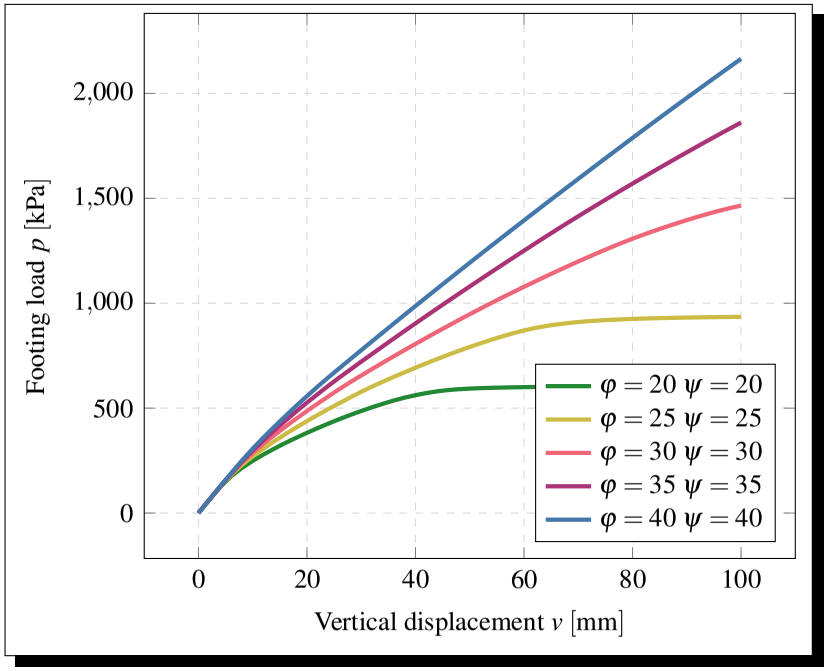

FIGURE 3.6: Set 5 with varying φ and ψ and $E = 100$[MPa], $\nu = 0.1$ and $c = 30$[kPa]

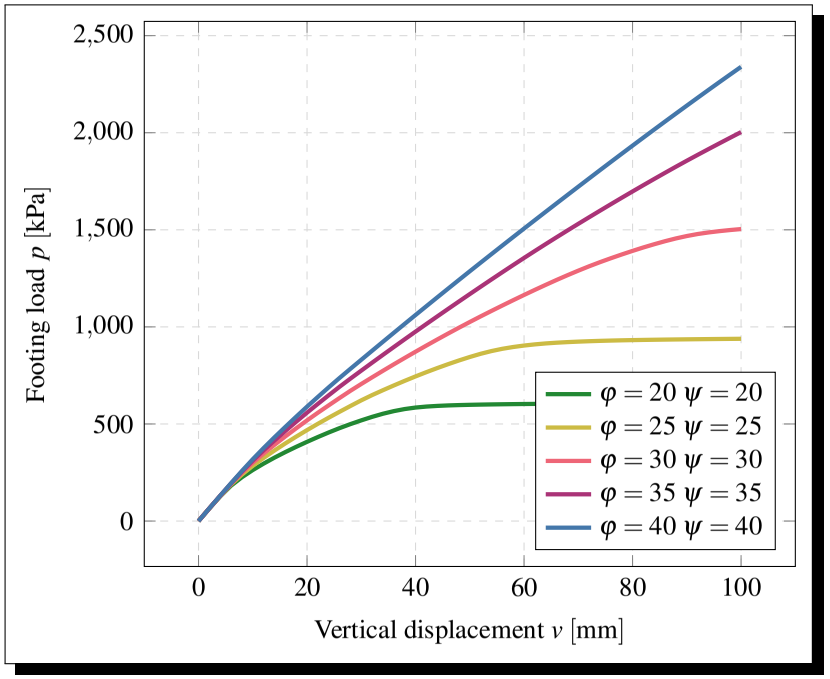

FIGURE 3.7: Set 6 with varying φ and ψ and $E = 100$[MPa], $\nu = 0.2$ and $c = 30$[kPa]

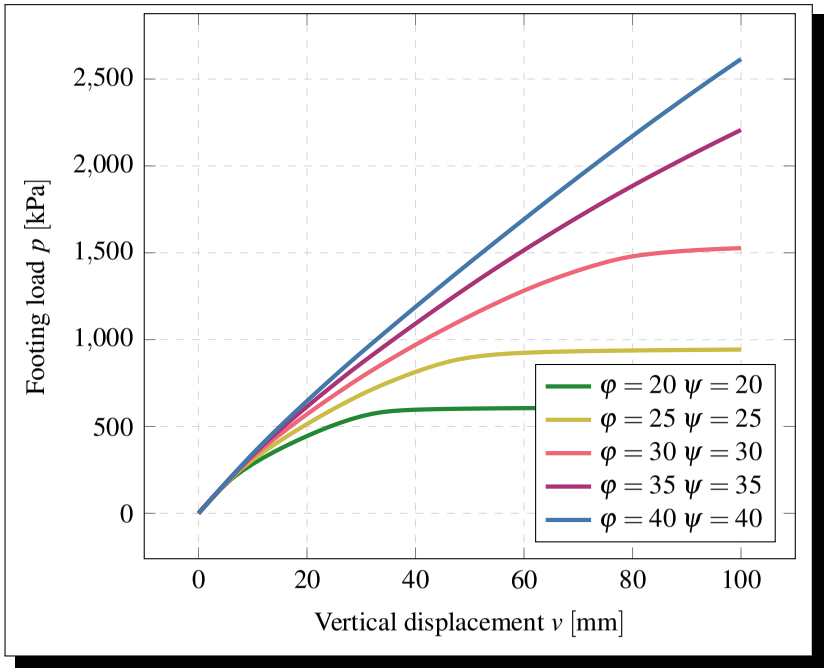

FIGURE 3.8: Set 7 with varying φ and ψ and $E = 100$[MPa], $\nu = 0.3$ and $c = 30$[kPa]

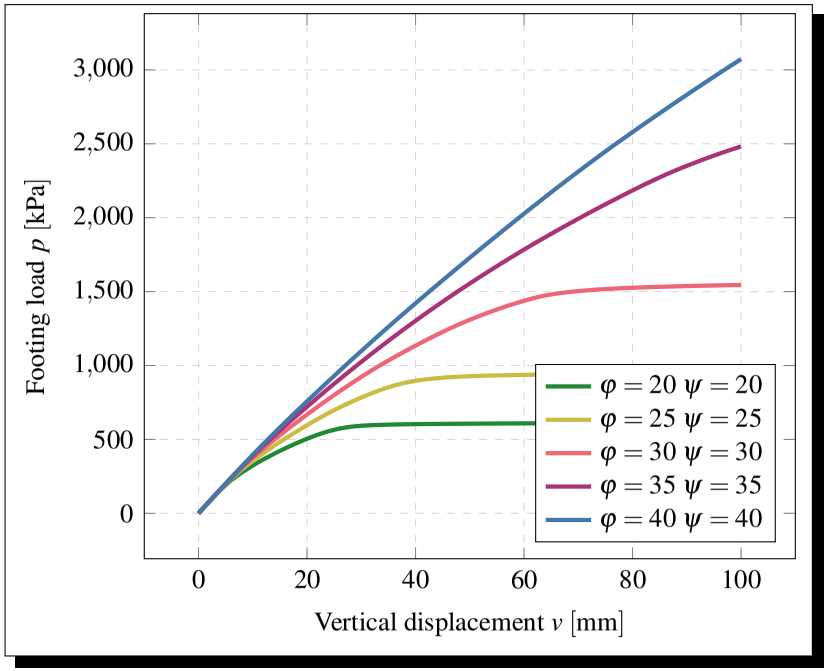

FIGURE 3.9: Set 8 with varying φ and ψ and $E = 100$[MPa], $\nu = 0.4$ and $c = 30$[kPa]

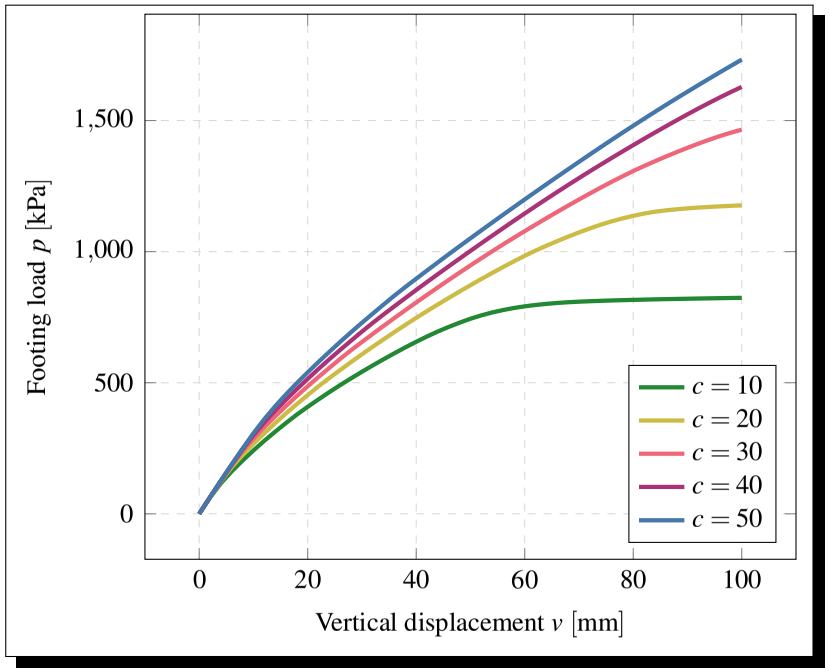

FIGURE 3.10: Set 9 with varying c and $E = 100$[MPa], $\nu = 0.1$, $\varphi = 30$[deg] and $\psi = 30$[deg]

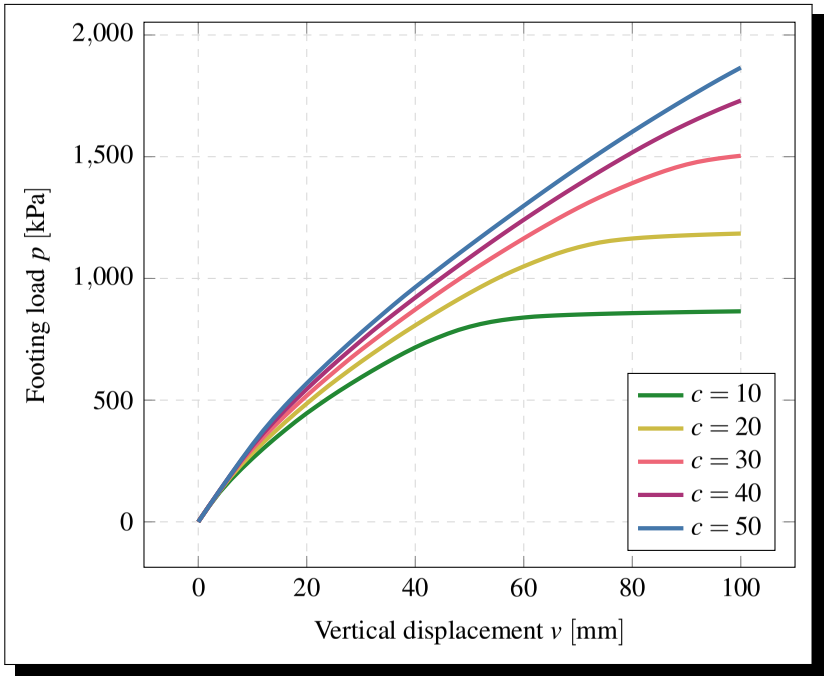

FIGURE 3.11: Set 10 with varying c and $E = 100$[MPa], $\nu = 0.2$, $\varphi = 30$[deg] and $\psi = 30$[deg]

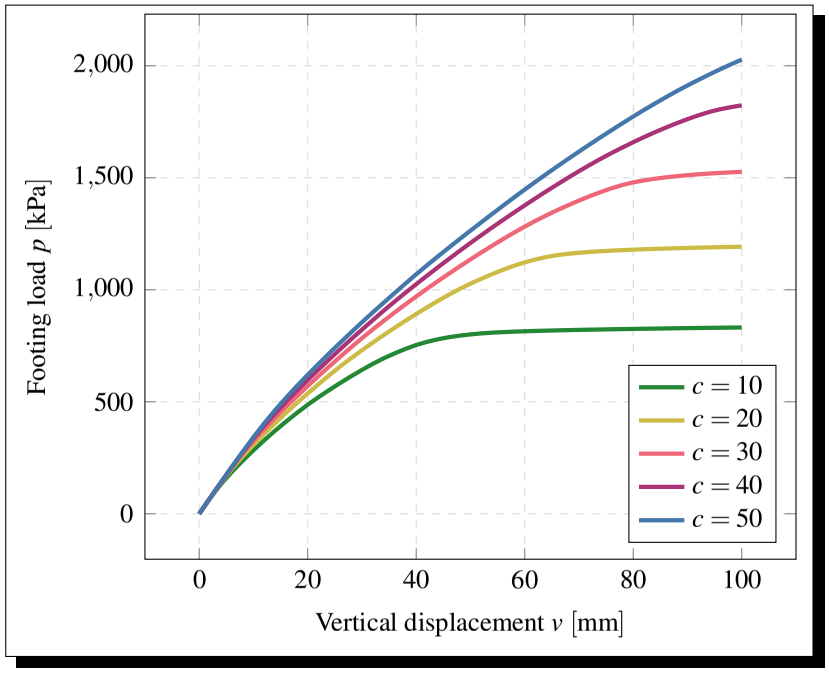

FIGURE 3.12: Set 11 with varying c and $E = 100$[MPa], $\nu = 0.3$, $\varphi = 30$[deg] and $\psi = 30$[deg]

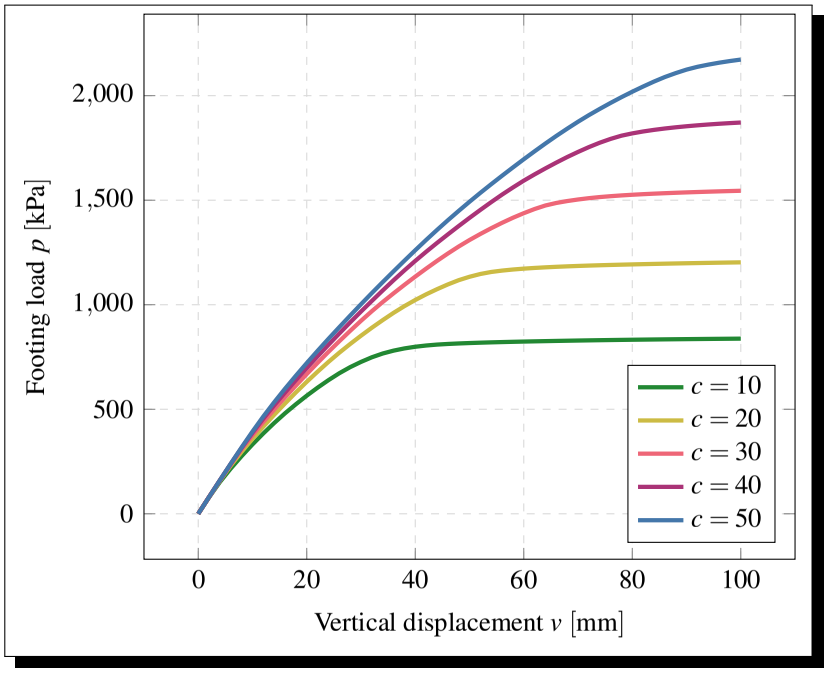

FIGURE 3.13: Set 12 with varying c and $E = 100$[MPa], $\nu = 0.4$, $\varphi = 30$[deg] and $\psi = 30$[deg]

Chapter 4

Developed Program

The developed program has its back end primarily based upon the program developed by Johan Clausen during his PHD thesis [9] which was written in MATLAB. For the reasons mentioned in Section 2.2, this original program has been translated into C# and the features and routines needed to determine material properties has been added by the author.

4.1 Structure of program

In order to give readers, who is not proficient in C# , a chance to fully understand the solution schemes implemented in the program, some form of schematic illustration is deemed necessary. For this reason, Algorithm 1 is used as illustration. It is a Newton-Raphson scheme in which a forced displacement is applied in each step k which is an increment of v_{final} divided by n_{ink} which is the maximum allowed number of steps. For each displacement increment, an equilibrium between the external and internal forces, a residual, are sought by evaluating and minimising the norm of the residual. This minimisation is done until it is smaller than a tolerance value or until the number of iterations done to reach an equilibrium exceeds a certain number n_{glo}. In each iteration the system stiffness matrix $\mathbf{K}$ is formed based on constitutive matrices, and finally the stresses and constitutive matrices are updated.

The domain for which the scheme of Algorithm 1 is applied can be seen on Figure 4.1. The displacement is applied at the nodes which are at the position where the footing is. The boundary is assumed to be roller supports which restricts movement perpendicular towards the supports but allows for movement parallel with the support. This is the type of boundary conditions which applies to all sides, except the top part of the soil body, which is free to move in any direction.

When modelling it is important to include a fairly large domain, relative to the applied deformations, such that the resulting deformations and stresses, which are of importance, are not lost. For these reasons the $Width$ and $Height$ are taken as $Radius \cdot 12$ and $Radius \cdot 10$, respectively.

When discretizing the domain, it is of importance to mesh the areas, where large deformation of stress gradients are present, with a large number of elements, relatively to the boundaries. For this reason, the domain is subdivided into 3 grades of element refinement; Fine, Medium and Coarse cf. Figure 4.2. The finest mesh refinement happens

Input:

$\mathbf{p}_k$; /* System load vector. */

1 $\Delta\mathbf{u}$; /* System displacement increment. */

2 **for** $k = 1$ **to** n_{ink} **do**

3 $j = 1$; /* Equilibrium iterations counter. */

4 **while** $\|\mathbf{r}_k\| > \varepsilon \|\mathbf{p}\|$ **do**

5 $\mathbf{K}_k^{epc,j} = \mathbf{K}_k^{epc,j}\left(\mathbf{D}_k^{epc,j}\right)$; /* Form the tangent stiffness matrix. */

6 $\mathbf{r}_k^j = \mathbf{p} - \mathbf{q}^j\left(\sigma_k^j\right)$; /* Determine force residual, $\mathbf{r}_k^j$ from $\mathbf{p}$ and internal forces $\mathbf{q}^j$. */

7 $\delta\mathbf{u}_k^j = \left(\mathbf{K}_k^j\right)^{-1}\mathbf{r}_k^j$; /* Solve the FEM equations. */

8 $\Delta\mathbf{u}_k^{j+1} = \Delta\mathbf{u} + \delta\mathbf{u}_k^j$; /* Update displacement increment. */

9 $\Delta\varepsilon_k^{j+1} = \mathbf{B}\,\Delta\mathbf{u}_k^{j+1}$; /* Calculate strain increment. */

10 $\Delta\sigma_k^{j+1} = \mathbf{D}\,\Delta\varepsilon_k^{j+1}$; /* Elastic stress increment. */

11 $\sigma_B = \sigma_k + \Delta\sigma_k^{j+1}$; /* Elastic predictor stress. */

12 $\sigma_k^{j+1}(\sigma_B)$; /* Plastic stress update. */

13 $\mathbf{D}_k^{epc,j+1}\left(\sigma_k^{j+1}\right)$; /* Constitutive matrix update. */

14 **if** $j > n_{glo}$ **then**

15 Break ;

16 $j = j + 1$; /* Iterations counter update. */

17 $\mathbf{u}_k = \mathbf{u}_k + \Delta\mathbf{u}_k^{j+1}$;

18 $p_{k,footing} = \mathbf{q}\left(DOF_{footing}\right) - \mathbf{p}\left(DOF_{footing}\right)$

Algorithm 1: Schematic principle of global Newton scheme for elasto-plastic FEM. [9, Table 1.1, Modified]

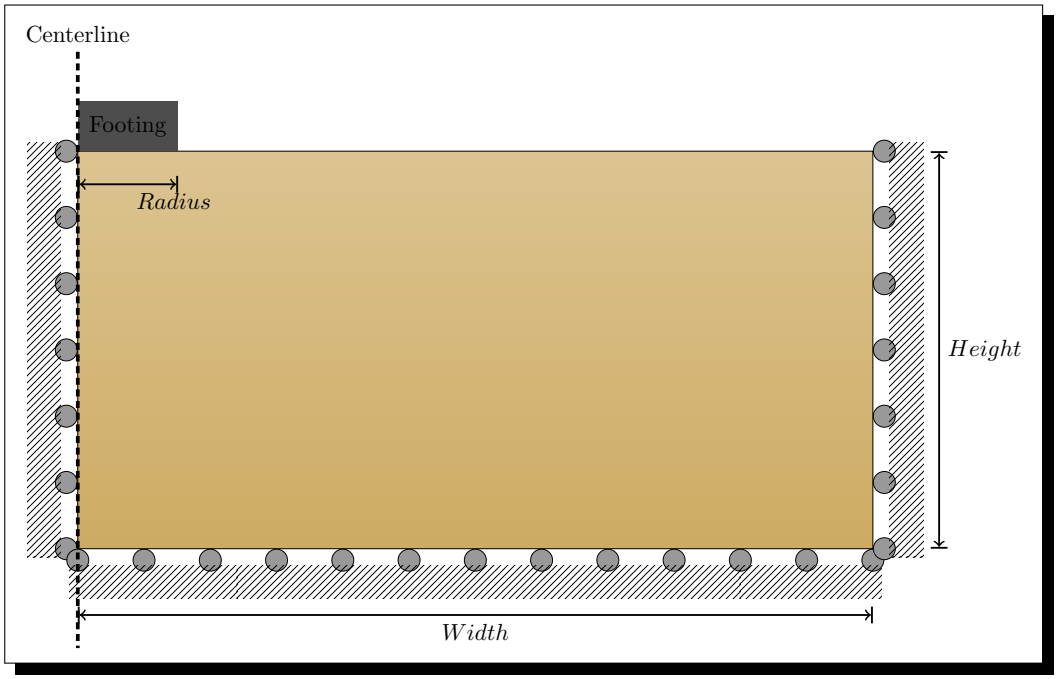

FIGURE 4.1: Geometry and boundary conditions for the computed domain.

at the top part of the domain in an area of $Radius \cdot 0.4 \times Radius \cdot 0.4$. The area is dependent on the size of the radius, but it could just as well be dependent on the final deformation, however having the area being dependent on the radius, the user faces no risks of being in a situation in which a wide footing is examined, where the footing would exceed the fine meshed area.

The input to define the meshed area, is interpreted as the number of elements along the boundaries. The user controls the mesh density of the coarse area, and the two other areas, the medium and fine, are dependent on the input to the coarse area. The relation between the number of coarse elements, $nCor$, and number of medium elements, $nMed$, are $nMed = nCor + 1$, and the relation between the number of fine elements, $nFine$, is $nFine = nMed + 1$.

4.2 Limitations

Simulating a real world scenario often brings along some choices which makes the model exactly what it is: A model. This means that choices are made to simplify the real world into something for which it is possible to calculate. For this reason the program suffers the following limitations

Stress type

The program calculates in plane strain. It was originally the idea to have both plane strain and axial symmetric stress types included. However due to some programming error, which did not get resolved, only plane strain was implemented correctly. This has the implication that the program is only capable of simulating strip footings and thus circular footings are not supported.

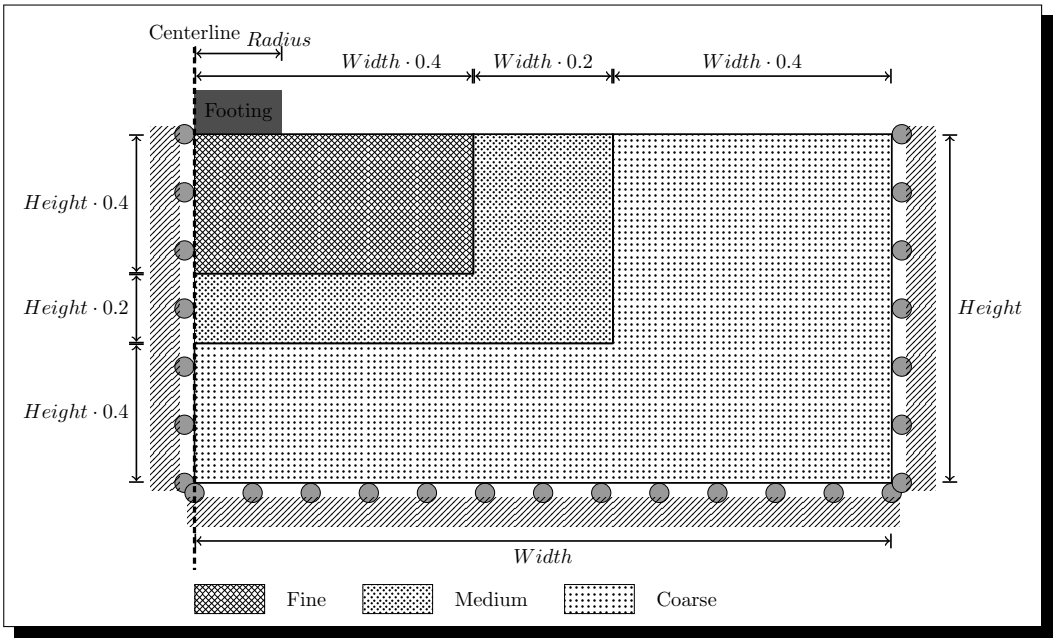

FIGURE 4.2: Illustration of the mesh density division.

Discretization of soil body

The domain is assumed as a homogeneous and uniform soil, meaning that the domain has the same properties throughout the entire body. This is again an assumption which maps poorly to a real world scenario, where a soil body can be a mix of different layers which can have very different properties.

Type of load applied

When performing test on soil, usually a load is applied and the subsequent displacement is recorded. In the program, only the option of specifying the final displacement is an option. The reason for not allowing the user to input actual data, is that the program is purely an experiment, and in order to correctly test and verify the output the, test data had to be made with precise and known parameters. If the program were to be used in some engineering company like COWI or ORSTED it would be relatively simple to program the ability to input actual data.

4.3 Input to program

The input is programmed in such a way that the user need to specify some program options, which is done through the use of three .TXT files. The user has access to the three files through the folder PROPERTIES/ and the files are

1_Material Through this file the user must specify the value of the variables E, ν, c, ϕ and ψ. It is necessary to specify the values of these because of two reasons. Firstly the input values are used to generate the target data for which the program does the optimisation and secondly, to make a convergence analysis based upon the specified values.

2_FEM Through this file the user sets the first set of settings specifically for the finite element analysis. The setting that must be specified are the *Radius* and it is possible

to manually set the $Width$ and the $Height$ of the domain, illustrated on Figure 4.1. If the $Width$ and $Height$ is specified as 0 the default values are used, as discussed in Section 4.1. Further more the user must specify the mesh density along the areas of the coarse edge. This is done through the *nCor* variable, and the program makes sure that the largest concentration of elements are near the footing and subsequently decreasing the number of elements as the distance to the footing increases. Care should be taken in specifying the mesh density, since the computational load is linked to the number of elements in the domain, and thus a large number of elements increases the computational load and subsequently the time each calculation takes, which is discussed later in Section 4.5. Last setting, $G0$ is the number of gauss points within each element and must be either 1, 3, 4 or 6.

3_Calculation The last file is used for the more specific FEM settings. Through this file the tolerance, which determines if the global equilibrium iterations has converged, are set through $tolfac$ and the number of global equilibrium iterations are specified through n_glo. The tolerance should ideally be in the order of 1×10^{-3} to 1×10^{-5}, and for the sake of speed the number of iterations should be between 10 and 20. If no number is specified to n_glo, and a very small tolerance is required, there is a possibility that the loop would not be able to converge. This is avoided by the use of restrictions on the amount of loop iterations.

The final deformation is specified through v_final along with the number of load increments n_ink. The number of load increments should be chosen while considering the final deformation since the calculation is non-linear. In order to be sure to avoid loosing information in between load steps, each load increments should be sufficiently small. however one should take care to not make each increment very small since the number of calculations needed until full deformation has occurred, is directly tied to the number of increments. This means that a compromise between the accuracy of the load-displacement curve and the computational time, must be made.

4.4 Implementation of parallelization

It is the purpose of this thesis to implement parallelization, in some form and thus some different approaches have been taken. First of all, the parallelization is implemented by taking advantage of the CPU and is not done using the GPU. To take advantage of the GPU, the NVIDIA CUDA software would be utilised, but since the author is not proficient in C++ this was not successful. The implementation was however successful in C# on the CPU so this is the case discussed hereafter.

4.4.1 Different ways to implement

Before the implementation the requirements for being able to to parallelize a program were checked and confirmed as they were descriped in the beginning of Chapter 2. As discussed in Section 2.1 the programming model were established as the fork/join method which form the basis for the implementation. The pitfalls, discussed in Section 2.3, were initially checked and remembered and with them in mind the following strategies have been implemented and tested.

System stiffness parallelization

The first implementation focused on the formation of the system stiffness, $\mathbf{K}_{sys}$. A schematic illustration of the determination of the system stiffness can be seen on Algorithm 2.

```
1 K_sys = 0 ;                          /* Initialisation of system          */
                                          stiffness matrix.
2 for Elemno = 1 to n_elem do
3     K_elem = B' D B w ;              /* Determination of element          */
                                          stiffness.
4     Dofs = Dof[Elemno,:] ;           /* D.o.f. associated with element    */
                                          Elemno.
5     K_sys[Dofs,Dofs] = K_sys[Dofs,Dofs] + K_Elem ;   /* Assembly into
                                                          system stiffness
                                                          matrix.
      */
```

Algorithm 2: Schematic principle of determination of system stiffness.

The implementation was mainly tested due to the fact that the formation of the matrix, agrees well with the following arguments:

Independence The formation of the stiffness matrix agrees very well with the requirement that each calculation must be independent. The global stiffness matrix is composed of an assembly of the element stiffness matrices which is completely independent of each other, and only dependent on the coordinates of the nodes and the constitutive matrix.

Balanced For the determination of the element stiffness the computations are the same, no matter the element and thus the computational load is the same for every element.

Scalability Since every element is independent from all others and the computational load needed to determine the element stiffness is the same for every element, the scheme should respond very well to an increase in the number of threads used in the computation.

The parallel implementation in the program of Algorithm 2 is done useing the code in Listing 4.1.

System stiffness determination

```
1  Parallel.For(0, ToInt32(nelem), elemnr =>
2 {
3     // Create temporary arrays
4     double[] tmpElemX = new double[ElemX.GetLength(1)];
5     double[] tmpElemY = new double[ElemY.GetLength(1)];
6     for (int i = 0; i < tmpElemX.GetLength(0); i++)
7     {
8         tmpElemX[i] = ElemX[elemnr, i];
```

```
9           tmpElemY[i] = ElemY[elemnr, i];
10      }
11
12      double[,,] tmpDeff_array = new double[4, 4, ToInt32(GT.
            order)];
13      for (int k = 0; k < GaussOrder; k++)
14      {
15          for (int j = 0; j < 4; j++)
16          {
17              for (int i = 0; i < 4; i++)
18              {
19                  tmpDeff_array[i, j, k] = Deff_array[i, j, k
                        , elemnr];
20              }
21          }
22      }
23
24      // Calculate element stiffness matrix
25      double[,] ElemK = K_LST(tmpElemX, tmpElemY,
            tmpDeff_array, GT.XI, GT.W, ToInt32(GT.order), 2);
26
27      // Degree of freedom associated with elemnr
28      int[] tmpElemDof = new int[ElemDof.GetLength(1)];
29      for (int i = 0; i < 12; i++)
30      {
31          tmpElemDof[i] = ToInt32( ElemDof[elemnr, i]) - 1;
32      }
33
34      // Load element stiffness into global stiffness matrix
35      for (int dofnr = 0; dofnr < 12; dofnr++)
36      {
37          for (int dofnrCol = 0; dofnrCol < 12; dofnrCol++)
38          {
39              SysKelas[tmpElemDof[dofnr], tmpElemDof[dofnrCol
                    ]] += ElemK[dofnr, dofnrCol];
40          }
41      }
42
43  });
```

C# code 4.1: Model.cs

Combination of parameter parallelization

The second implementation strategy was to take each set of soil parameters and run them on their own thread. As with the system stiffness parallelization, this strategy agreed well with the following arguments:

Independence The formation of the sets of parameters are done ind such a way, that when a simulation finishes, it simply continues with the next combination, and does

nothing else with the result, than to display it if it meets some requirements, cf. Section 3.2.

Balanced It is not exactly certain that each simulation takes the same amount of computational time or demands the same computational power, but since every simulation is highly independent this does not have that much of an influence. The biggest issue with an unbalanced loop is that other threads needs to wait until the slowest one has finished. This will only affect the last simulations, since every combination of parameters are calculated.

Scalability Since the entire simulation is done on its own thread, it should demonstrate a high degree of scalability.

Further more this strategy is able to utilise the highly useful feature about C# , namely the object-oriented programming. Looking at Listing 4.2, it is straight forward to create a model object with the sets of parameters, in several nested loops, run the model and then subsequently evaluate the result.

Parameter combination

```
1  Parallel.ForEach(TestE, (iE, loopState) =>
2  {
3      // Loop thorugh the values of Coh
4      for (int iCoh = 0; iCoh < TestCoh.Length; iCoh++)
5      {
6          // Loop thorugh the values of Nu
7          for (int iNu = 0; iNu < TestNu.Length; iNu++)
8          {
9              // Loop through the value of Phi and Psi
10             for (int iPP = 0; iPP < TestPsi.Length; iPP++)
11             {
12                 // Create placeholder for combination
                      material properties
13                 double[] CombMat = new double[]
14                 {
15                     TestE[iE]       , // Youngs modulus
16                     TestNu[iNu]     , // Poisson ratio
17                     TestCoh[iCoh]   , // Cohesion
18                     TestPhi[iPP]    , // Friction angle
19                     TestPsi[iPP]      // Dilation angle
20                 };
21                 Model DetModel = new Model()
22                 {
23                     // Material specific
24                     E        = CombMat[0],
25                     nu       = CombMat[1],
26                     coh      = CombMat[2],
27                     Mphi     = CombMat[3],
28                     Mpsi     = CombMat[4],
```

```
                    gam_tot   = MatSet[5],
                    Material  = ModelMaterial,
                    // FEM specific
                    r_footing  = FEMSet[0],
                    h_domain   = FEMSet[1],
                    b_domain   = FEMSet[2],
                    k_mesh     = SoilDeterDensity,
                    GaussOrder = FEMSet[4],
                    // Calculation specific
                    tolfac      = CalSet[0],
                    v_final     = CalSet[1],
                    nink        = ToInt32(CalSet[2]),
                    n_maxglobal = ToInt32(CalSet[3]),
                    nsigma      = ToInt32(CalSet[4])
                };
                // Run model in object
                DetModel.RunModel();
                // Determine the relative error
                double[] RelError = ErrorFuntion(ExData1,
                   DetModel.OutputStressPoints);
                // Check if an exact solution is found

                // Print the values and relative error in
                   object
                if (RelError[0] >= 0 && RelError[0] <= 5 &&
                    RelError[1] >= 0 && RelError[1] <= 2 &&
                    RelError[2] >= 0 && RelError[2] <= 2)
                {
                    // Print the results to the command
                       window and files

                }
            } // End of Phi Psi loop
        } // End of Nu loop
    } // end of Coh loop

});
```

C# code 4.2: Program.cs

Generally the use of nested loops was a necessity due to the extremely large number of different combinations of the variables. There are simply so many combinations, depending on the increment of each variable, that if they were to be put in a single matrix they would form an array to large to be stored in the computer memory.

4.4.2 Difficulties of the implementation

Both of the aforementioned strategies were tested for their usefulness, but only one of them made the cut for the final program. The first one, where the determination and assembly of

the stiffness matrix are done in parallel, showed, in some situations, very strange behaviour when evaluating the result. After countless trial and error to pinpoint the error, it was discovered that it was an issue of asynchronous thread safety of the $\mathbf{K}_{sys}$-variable. As can be remembered, the determination and assembly is done by multiple threads, and the issue is not the determination of the individual element stiffness, $\mathbf{K}_{elem}$, but rather the assembly to the global stiffness matrix. In some situations, two or more threads were accessing and writing to the $\mathbf{K}_{sys}$-variable and in doing so, they would interfere with each other, thus producing an incorrect stiffness matrix. Some different routines exists to prevent this, but the one which showed most promising results were locking routines.

The locking routines work by restricting access to variables which are used by one thread, and will release the variable, when the current thread are done using it. The locking routines were implemented successfully but they resulted in a significantly slower executing time. The main cause of this, is that the variable becomes a bottleneck for the system, meaning that when the variable is locked, all other threads which needs to write to said variable, will wait. Because of this major increase in computational time, the system stiffness parallelization approach was abandoned.

This leaves the last parallelization strategy as the option which was used, and illustrated in Listing 4.2. This strategy really utilises the key programming concept of C# , namely the object-oriented approach. Each of the threads, in the parallel pool, creates a new model object, and in doing so, no thread are able to access variables used by other threads, thereby securing the program against the issue of thread safety. Further more only one parallel thread pool are initiated versus the first strategy. In that, a new thread pool would be initiated and closed in each iteration, which would amount to a substantial increase in computational time due to the overhead associated with initiating a thread pool, as mentioned in Section 2.4.1.

4.5 Model convergence

When doing numerical modelling, it is of paramount importance to show that, when the domain is approximated with an increasing number of elements, the model converges towards a steady value. For this reason, the final load on the footing is used as a reference for the examination. The number of degrees of freedom is refined in a consistent manner by steady increasing the number of elements, *nCor*, along the coarse edge, as discussed in Section 4.1.

The study can be seen on Figures 4.3 and 4.4, where a consistent refinement is done from $nCor = 2$ with an increase of 2 for every refinement, leading up to $nCor = 30$. The number of degrees of freedom subsequently starts from $n_{ndof,nCor=2} = 322$ and goes up to $n_{ndof,nCor=30} = 5940$. The very first mesh is a test mesh created manually with 4 elements and 30 degrees of freedom.

As expected, when increasing the number of degrees of freedom, some form of trend is apparent and a steady value appears as the mesh is increasingly refined. The first couple of refinements appears to converge extremely fast in the first three refinement, then the following four fluctuates a bit before finally settling at a steady value of around 0.6 MPa.

The convergence however, comes at a significant cost of computational time. For some reason, and completely against the expectations of the author, the program execution time is slow, as illustrated in Figure 4.5. This issue poses a big challenge for the whole idea for

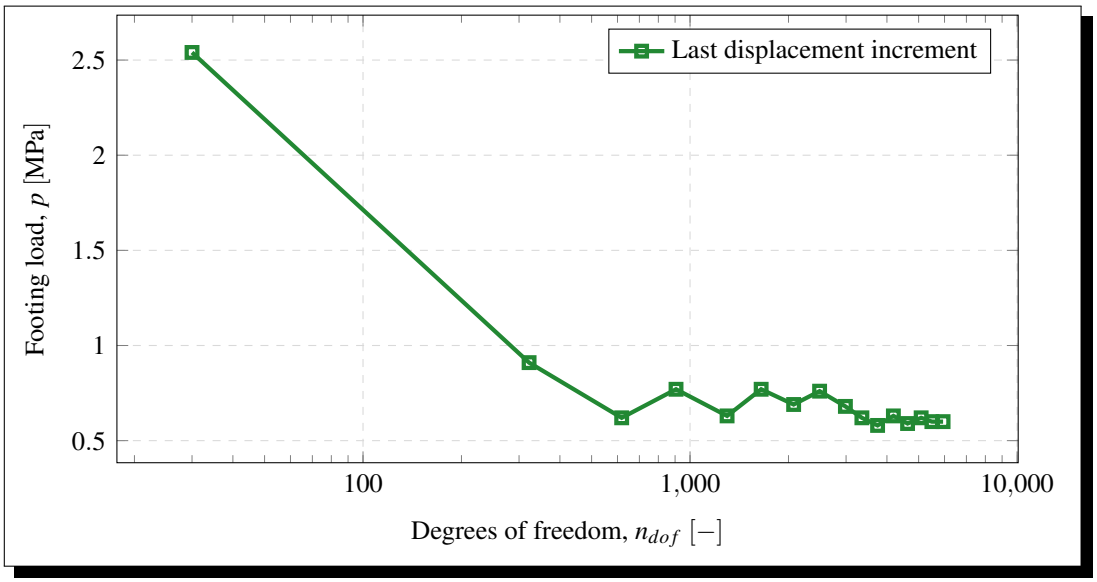

FIGURE 4.3: Convergence study with an increasing number of degrees of freedom. Model parameters are: $E = 50\,\text{MPa}$, $\nu = 0.49$, $c = 30\,\text{kPa}$, $\varphi = 20°$ and $\psi = 20°$

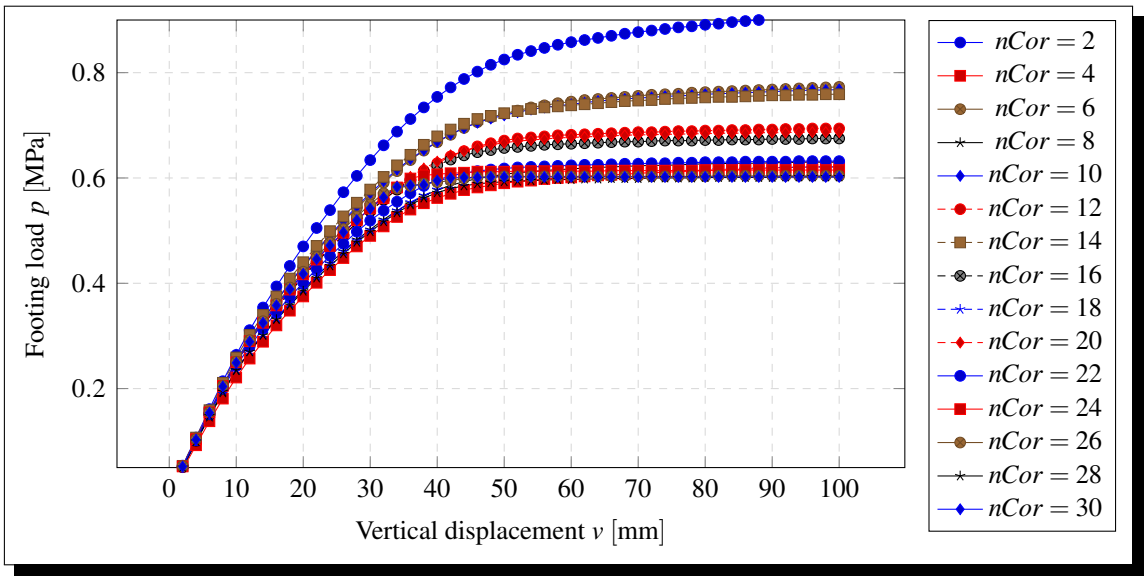

FIGURE 4.4: Load-displacement response from convergence study with an increasing number of degrees of freedom. Model parameters are: $E = 50\,\text{MPa}$, $\nu = 0.49$, $c = 30\,\text{kPa}$, $\varphi = 20°$ and $\psi = 20°$

which the project is based upon, since the program is intended to calculate an extremely large solution space until the closest material parameters are determined. But when just a single calculation, with a number of degrees of freedom which have converged, takes the good part of 30 min the program looses its meaning.

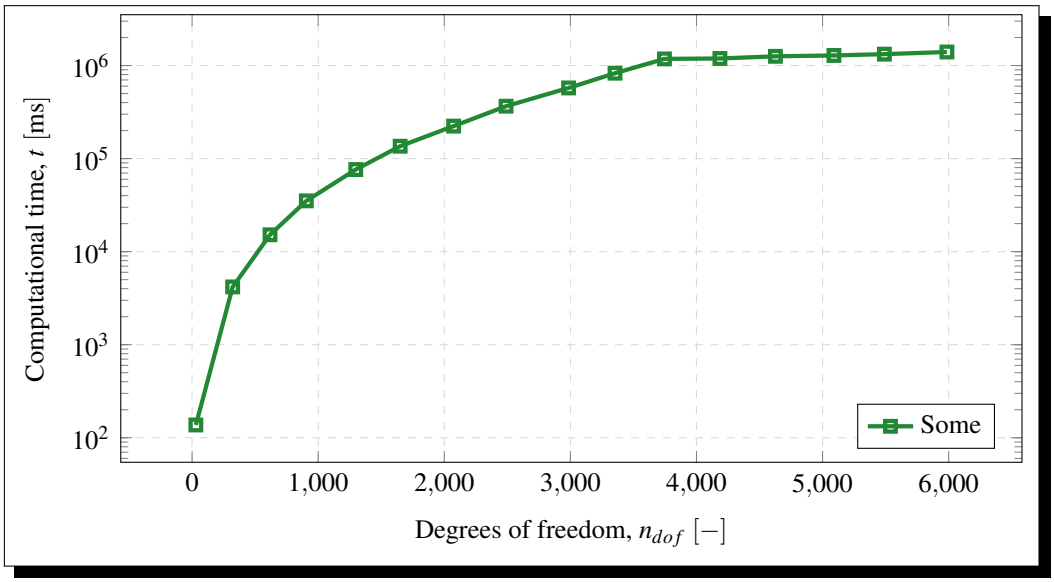

FIGURE 4.5: Degrees of freedom with the associated computational time for the convergence study.

However, the second refinement step in Figure 4.3 with $n_{dof,nCor=2} = 322$ offers some salvation to the issue with the long computation time. Even though the footing load has not converged to some steady value, it offers a significant improvement to the load estimate than the very first mesh density. And when comparing with the higher mesh densities, the footing load fluctuates until final convergence happens at around 5000 degrees of freedom. With the refinement step $n_{dof,nCor=2} = 322$ the computation takes approximately 4 s which must be deemed acceptable.

4.6 Results

The result part is divided in two parts, namely one where the program is used to evaluate the material parameters and secondly a part where the significance of the parallelization is investigated.

4.6.1 Determination of material parameters

In order to get a decent estimation of the material parameters without waiting an unreasonable amount of time, the mesh is created with $n_{dof,nCor=2} = 322$. Three different combinations of parameters are targeted and are listed in Table 4.1. Compared to the suggested test sets in Section 3.1, only a portion of the data is used for the parameter determination.

The combinations listed in Table 4.1 are used as a proof of concept, and if the program provides decent results for these combinations, it should, in principle, provide decent results for any combination.

Table 4.1: *Material parameters used as input to the material determination. For all combinations* $\gamma_{total} = 20\,\mathrm{kN/m^3}$.

Combination	E [MPa]	ν [-]	c [kPa]	φ [deg]	ψ [deg]
1	50	0.2	30	20	20
2	50	0.3	30	20	20
3	50	0.4	30	20	20

When the program compares the fitted parameters with the target data and evaluates the error, some requirements were mentioned in Section 3.2. For the sake of simplicity the requirements are kept the same for all three combinations of target data. For Equation (3.2) the estimated error must be smaller that 1 MPa, for Equation (3.3) it must be smaller than 0.2 MPa and finally for Equation (3.4) the requirement is set at 5 %. These values were chosen, in some degree of randomness, and in some degree of experience with the program. During testing some values for the requirement were tried to figure out how many sets of fitted data the chosen value would output. With the chosen bounds for the requirements, a substantial number of fitted parameters were found. The number of sets of fitted parameters are listed in Table 4.2.

Table 4.2: *Number of fitted parameters for each combination of the values in Table 4.1.*

Combination	1	2	3
Sets of fitted parameters	52	32	17

Since a large number of sets of parameters are found which fulfil the requirement for the error functions, it was chosen to display only a portion of the sets of fitted parameters, for some of the data. This is done since it becomes increasingly more difficult to distinguish the different sets of fitted parameters as more are plotted in the same figure.

For the case of combination 1 which have 52 sets of fitted parameters, 10 were chosen at random. For combination 2, 32 sets qualified, but here 15 were chosen, again at random. Finally, combination 3 had 17 sets, all of which are shown. The three combinations are shown in Figures 4.6 to 4.8 and all sets of fitted parameters are listed in Appendix Tables B.1 to B.3.

It is quite interesting to see how well the fit is to the input data. Common for all combinations is that the fitted parameters fit well to the initial part of the load-displacement curve. For combination 1 and 2, Figures 4.6 and 4.7, the last part of the fit, in what begins to exhibit plastic behaviour, is not that good and begins to drift slightly. This is not the case for combination 3, Figure 4.6, where all the fitted sets does not appear to exhibit any sort of drift and are well within the requirement of the error functions.

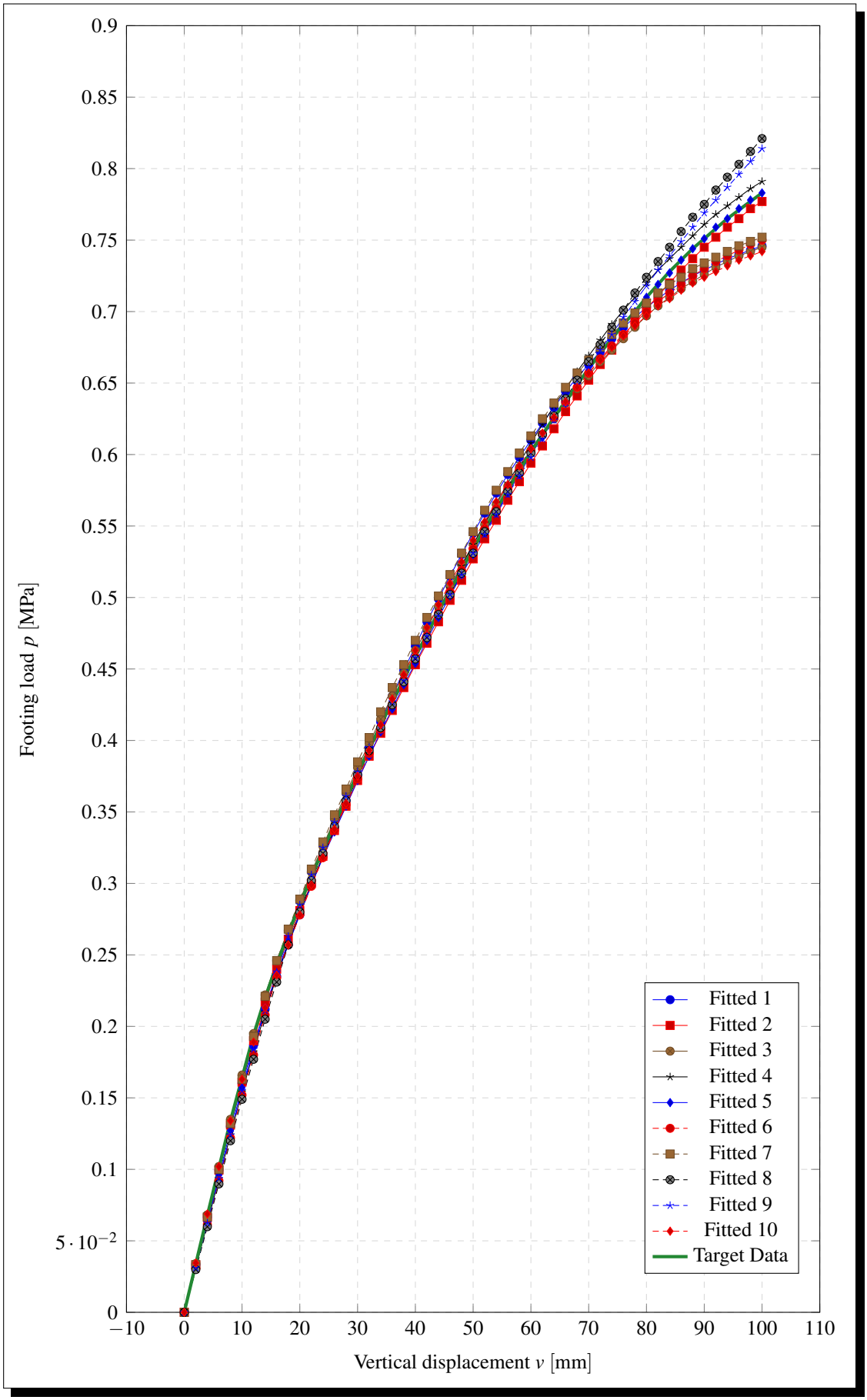

FIGURE 4.6: Results for Combination 1.

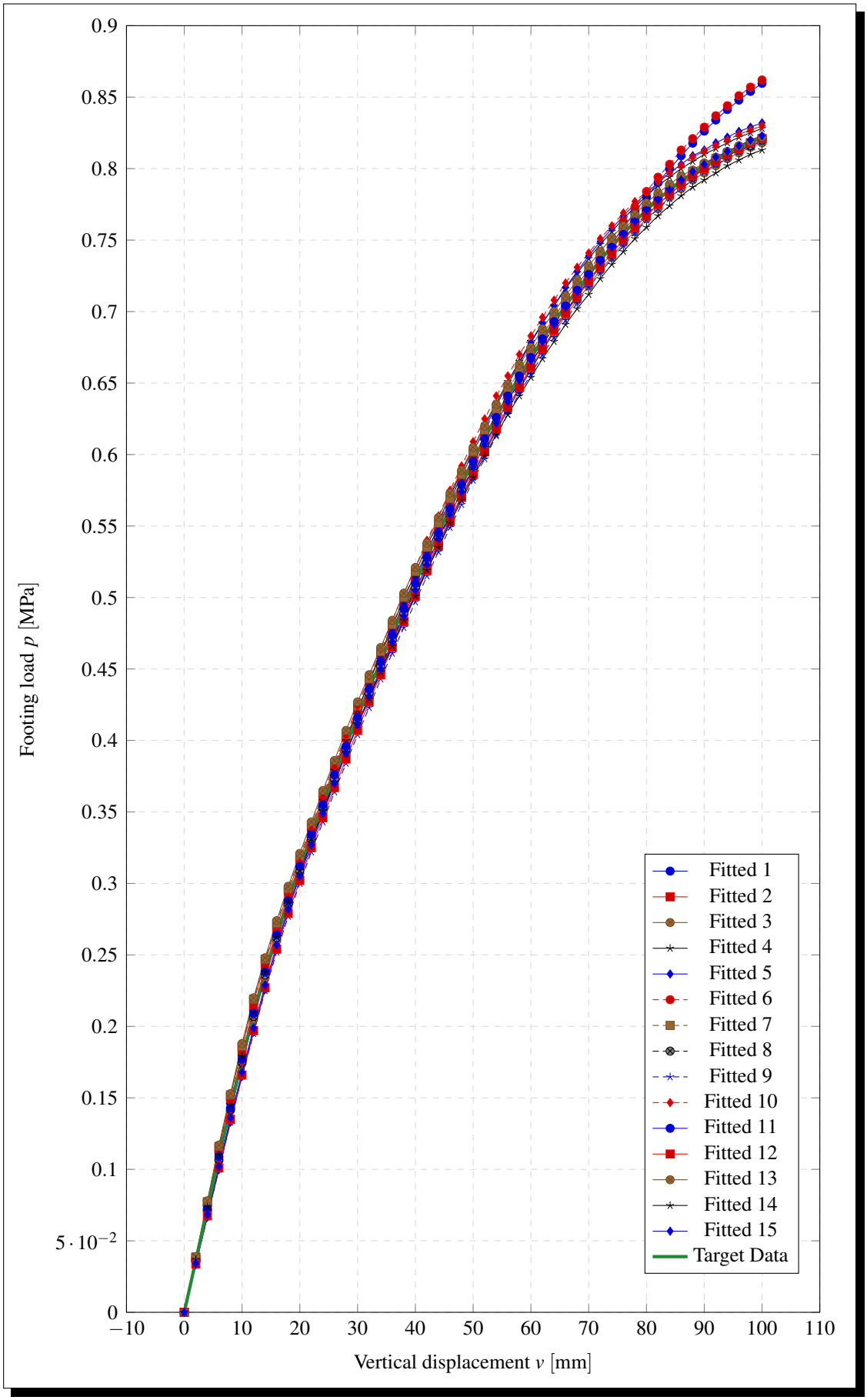

FIGURE 4.7: Results for Combination 2.

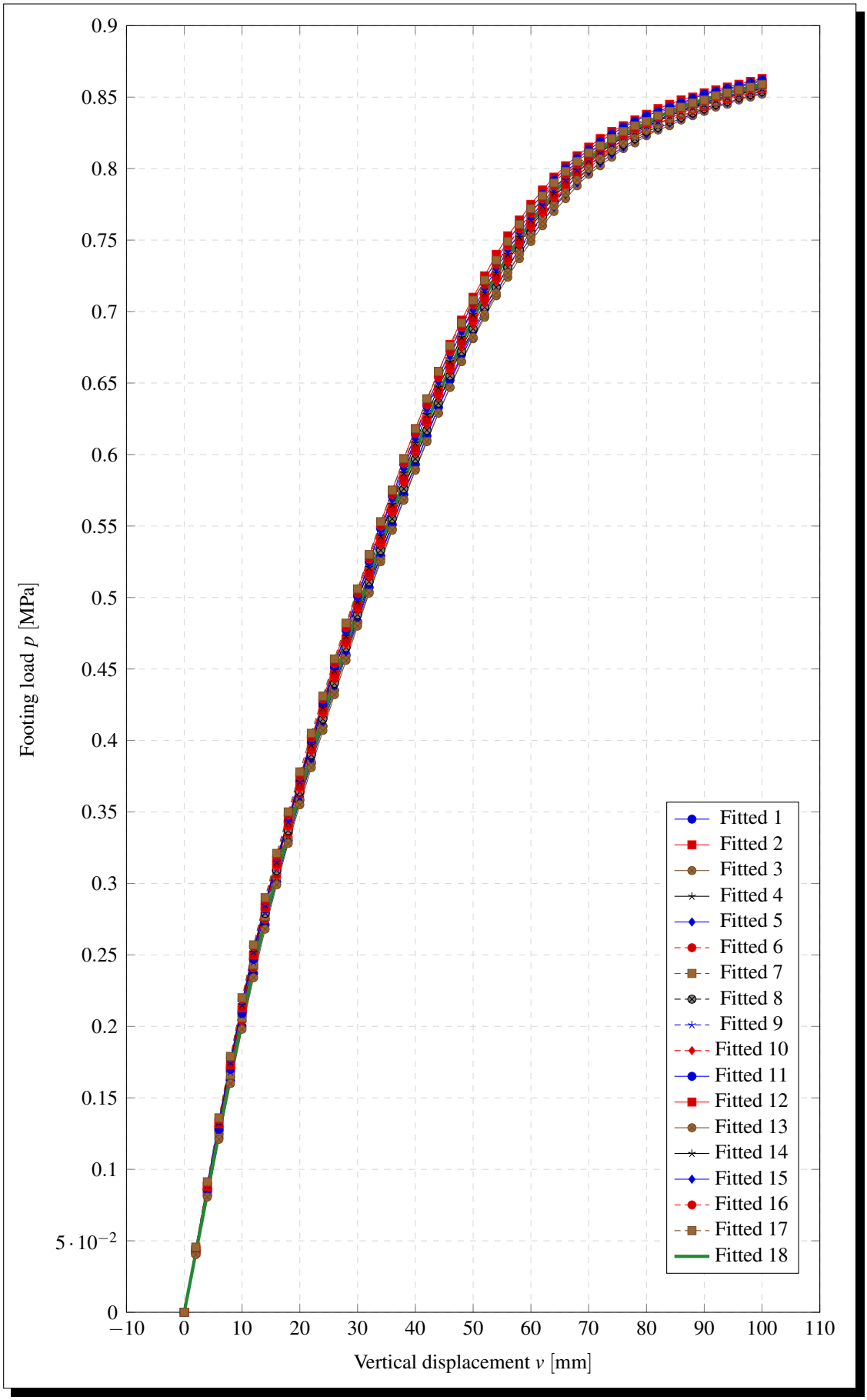

FIGURE 4.8: Results for Combination 3.

4.6.2 Significance of parallelization

As mentioned in Sections 2.4 and 2.4.1 a system which responds positive to an increase in the number of available threads are said to be highly scalable. In order to determine how scalable the developed program is, a baseline is needed. This baseline is determined by measuring how much time a set of simulations takes to perform on one thread.

For this case, the finest mesh is chosen in order to speed up the simulations and since all threads are rated to have the same processing power, it does not matter if the computed mesh is coarse or extremely fine.

With a baseline of the computation time established it is merely a case of running the same sets of simulations with a linear increasing thread pool which is illustrated on Figure 4.9.

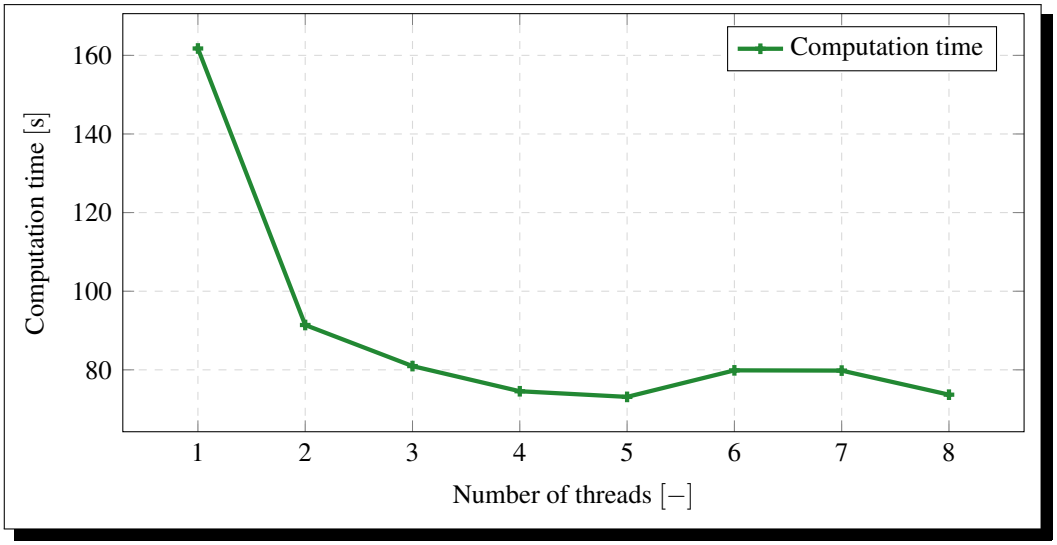

FIGURE 4.9: Measured time for computations as increasing the number of threads. Computations performed on a INTEL® CORE™ i7-4710MQ with 20 Gb of DDR3 RAM.

In order to determine how much the simulation is speed up, the baseline value of the computational time with one thread is divided by the computational time with n-number of threads. This is what is illustrated on Figure 4.10 along with Equation (2.1) and listed in Table 4.3.

Table 4.3: *Number of thread, measured computation time and the calculated relative speedup.*

Number of threads [-]	1	2	3	4	5	6	7	8
Computation time [ms]	161.74	91.40	80.95	74.55	73.11	79.86	79.80	73.68
Relative speedup [-]	1	1.76	1.99	2.16	2.21	2.02	2.02	2.19

In order to reach a high level of scalability, the parallel execution time should be above 95 % but it is evident that the program executes the computations between 50 % and 75 % in parallel. This poses a problem since adding more computational power, does not yield a significant change in computational time, which is clear supported by Figures 4.9 and 4.10.

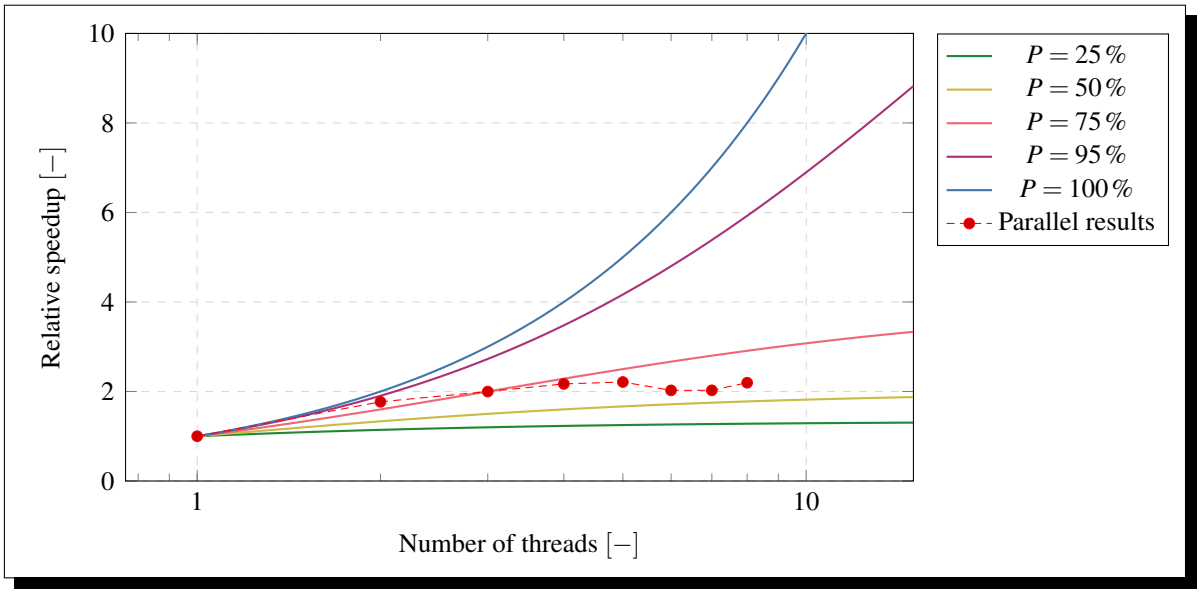

FIGURE 4.10: Actual speedup plotted with red dots. Computations performed on a INTEL® CORE™ i7-4710MQ with 20 Gb of DDR3 RAM.

Chapter 5

Conclusion

The conclusions to be drawn from the individual chapters will be summarised.

Chapter 2 In this chapter different key points of the parallel implementation were discussed and highlighted. Two different programming models were discussed and a choice of one were made, namely on the so-called Fork/Join model. This programming model forms the basis for the entire parallel implementation.

A choice of programming language were made, which fell upon C# . The advantages and disadvantages of the choice were discussed and weighed.

Typical pitfalls within parallel programming is highlighted, and an investigation into the overhead incurred by initiating and terminating parallel pools are made. The investigation showed that for small problem sizes, sequential loops perform marginally better than its parallel counterpart. But for large problem sizes the parallel loop comes out performing significantly better than the sequential loop.

Lastly, the speedup which can be expected, by the parallel implementation, is determined. This showed that loops need to have a large degree of parallelism in order to get a significant speedup, when adding more computational power.

Chapter 3 In this chapter the data for the program is initially analysed and a choice of combinations are recommended. The recommended target data shows both elastic and plastic behaviour during the load-displacement of the footing.

Three different functions were presented, which were able to estimate the relative error between the target data and the simulated data. These functions are used to either reject or accept the simulated data based on a target accuracy.

Chapter 4 The structure of the program is explained and presented in a straight forward manner. The limitations are shortly explained followed by a thorough walk-through of the input settings.

Two different schematics of explicit parallel implementation is presented and discussed. The first one is centred around the determination and assembly of the system stiffness and the second implementation is focused on creating a model object on parallel threads. Some issues, which were highlighted in Chapter 2, were present in the first implementation and thus the second option were chosen with great success.

An investigation into the validity of the program is done by means of a convergence analysis. In this analysis the model mesh is consistently refined from an initial state with 322 degrees of freedom up to the final, converged mesh, of 5986.

The results from this chapter is two fold. Firstly, from Chapter 3, some of the suggested target data is used for the material determination. The program performed acceptable and a number of different fitted sets of parameters were found for each target data. Secondly, the significance of adding more compute threads were analysed. As discussed in Chapter 2, for the program to be highly scalable, a large portion of the code must execute in parallel. It is found that the fraction of time which is spend in parallel is approximately 50-75 % thus resulting in a speedup of around 2.19.

5.1 Further Work

The program shows promising results when fitting values to some target data which is created manually. However for the program to be applicable in every day work life, some improvements must be done to make the program execute faster.

Some further improvements could be done in the user interface if the program is to be used in a company. Although the input is thoroughly explained, it is not intuitive.

An exiting improvement would to get the parallelization to work with the GPU and test the significance of this. This however requires that it is reprogrammed in another language.

Bibliography

[1] Brian Bjerrum. “Plate load test on Light Weight Clay Aggregate - A comparison of relevant material models and parameters”. In: (2018).

[2] Par Persson Mattsson. *Why Haven’t CPU Clock Speeds Increased in the Last Few Years?* Read 05-06-2018. URL: https://www.comsol.com/blogs/havent-cpu-clock-speeds-increased-last-years/.

[3] *History of Processor Performance*. Read 30-05-2018, Course in "Fundamental of Computer Systems". Columbia University, 2012. URL: http://www.cs.columbia.edu/~sedwards/classes/2012/3827-spring/advanced-arch-2011.pdf.

[4] Stephen Toub. *Patterns of parallel programming*. Microsoft Corporation, 2010.

[5] Bryon Mayer. *Managing Loops and Data Synchronization Are Keys to Succesful Parallelization*. Read 15-12-2017. URL: https://www.ecnmag.com/article/2011/01/managing-loops-and-data-synchronization-are-keys-successful-parallelization.

[6] Microsoft Corporation. *Task Parallel Library*. Read 30-03-2017. URL: https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/task-parallel-library-tpl.

[7] Robert G. Brown. *Amdahl’s law and Parallel Speedup*. Read 30-05-2018. Duke University, 2000. URL: http://webhome.phy.duke.edu/~rgb/brahma/Resources/als/als/node3.html.

[8] Blaise Barney. *Introduction to Parallel Computing*. Read 31-10-2017. Lawrence Livermore National Laboratory. URL: https://computing.llnl.gov/tutorials/parallel_comp/#Examples.

[9] Johan Christian Clausen. *Efficient Non-linear Finite Element Implementation of Elasto-Plasticity for Geotechnical Problems*. Esbjerg Institute of Technology, Aalborg University: John Wiley & Sons, Inc., 2007. ISBN: 978-87-7606-018-3.

[10] Zhen-Yu Yin et al. “Optimization techniques for identifying soil parameters in geotechnical engineering: Comparative study and enhancement”. In: (2017).

[11] Robert D. Cook et al. *Concepts and Applications of Finite Element Analysis*. 4th edition. University of Wisconsin–Madison: John Wiley & Sons, Inc., 2002. ISBN: 978-0-471-35605-9.

[12] Niels Saabye Ottosen and Matti Ristinmaa. *The Mechanics of Constitutive Modelling*. Lund University: Elsevier, 2005. ISBN: 0-008-044606-X.

[13] Sien Kok et al. “A Review of Basic Soil Constitutive Models for Geotechnical Application”. In: (2009).

Appendix

Appendix A

Parallel test program

To validate the postulate in Chapter 2.3, p. 9, a relatively simple test program has been written in order to examine the implications which parallel programming leads to. The program evaluates EQ. (A.1) to an increasingly larger array of numbers, with the goal in mind, to log the time that it takes.

$$f(x) = \left(x^2\right)^2 \tag{A.1}$$

It is expected that evaluating the equation for a relatively small array of numbers will be faster in sequential but increasing the array to a relatively large size will be done faster in parallel. This is expected to be due to the fact that starting up and ending a parallel pool takes some amount of time. Because of this, problems which can be calculated extremely fast, suffers slightly in parallel versus sequential loops.

In order to make it more clear what has been done, the program has been divided into parts, with a brief explanation, as presented in the following section.

A.1 Program Code

The block of code is basically a bunch of settings for the program to follow. As can be observed, these include the maximum size of the array to be evaluated given from lines 1-10. Next is settings for saving when the program is done computing and finally is the initialisation of diagnostics tools, which are used to log the time it takes for each sequential and parallel loop to finish.

Initialization

```
1 /// - - - - - - - - - - - - -
2 /// Pre set-up
3 int dimension = 980; // Size of the array
4 int[] stopNum = new int[dimension]; // Pre-allocate array
5    stopNum[0] = 1; // Manually set first value
6 for(int i = 1; i < dimension; i++) // Assign values to the
     rest of the array
```

```
7  {
8      if    (i<6) { stopNum[i] = stopNum[i - 1] * 10;}
9      else        { stopNum[i] = i * 100000; }
10 }
11
12 int tmp = 0;
13 bool Finished = true;
14 string[] txtFilePath = {
15     @"D:\Programmer\Dropbox\ProjektMapper\MasterThesis\
          Programmer\Data\SeqData.dat",
16     @"D:\Programmer\Dropbox\ProjektMapper\MasterThesis\
          Programmer\Data\ParData.dat" };
17
18 /// Diagnostics
19 Stopwatch SWTask = new Stopwatch();
20 Stopwatch SWSeq = new Stopwatch();
21 Stopwatch SWPar = new Stopwatch();
22
23 TimeSpan TSTask = new TimeSpan();
24 TimeSpan[] TSSeq = new TimeSpan[stopNum.Length];
25 TimeSpan[] TSPar = new TimeSpan[stopNum.Length];
```

C# code A.1: Program.cs

The program now enters a `foreach` loop which creates an array, `y`, with the size of each integer value in `stopNum`. What follows is first the sequential loop followed by the parallel loop, each of which evaluates EQ. (A.1) from $0 \rightarrow$ size of `y`. The last part of the loop is a incremental counter, which are used to place the computation time of each loop, determined by the diagnostics tools, in the correct position.

Primary and secondary loop

```
26 /// Computing
27 SWTask.Start(); // Stopwatch initiate
28 foreach (int j in stopNum)
29 {
30   // Create array to store values
31   double[] y = new double[j];
32   Console.WriteLine("␣");
33   Console.WriteLine("Problem␣size␣:␣{0}", j);
34
35   // SEQUENTIAL LOOP
36   Console.Write("Sequential␣run␣number␣:␣{0}", tmp);
37   SWSeq.Start();
38   for(int i = 0; i < j; i++)
39   {
40       y[i] = Math.Pow(Math.Pow(i, 2), 2);
```

```
41      }
42      SWSeq.Stop();
43      TSSeq[tmp] = SWSeq.Elapsed;
44      Console.WriteLine("␣␣␣␣␣Finished␣in␣:␣{0}␣ms!", TSSeq[tmp
           ].TotalMilliseconds);
45
46      // PARALLEL LOOP
47      Console.Write("␣␣Parallel␣run␣number␣:␣{0}", tmp);
48      SWPar.Start();
49      Parallel.For(0, j, i =>
50           {
51                y[i] = Math.Pow(Math.Pow(i, 2), 2);
52           });
53      SWPar.Stop();
54      TSPar[tmp] = SWPar.Elapsed;
55      Console.WriteLine("␣␣␣␣␣Finished␣in␣:␣{0}␣ms!", TSPar[tmp
           ].TotalMilliseconds);
56
57      // Increment counter
58      tmp++;
59  }
60  SWTask.Stop(); // Stopwatch
61  TSTask = SWTask.Elapsed; // Time to execute
```

C# code A.2: Program.cs

When the program has finished each entry in `stopNum` it continues to save the obtained results, given that it has been specified in C CODE A.1. The program finally outputs a `.dat` files which can be used in another afsnit.

Exporting data to file

```
61  /// Printing
62  Console.WriteLine("␣");
63  Console.WriteLine("␣Total␣runtime␣:␣{0}␣ms", TSTask.
       TotalMilliseconds);
64
65  // Save to file
66  if (Finished)
67  {
68      foreach(string key in txtFilePath)
69      {
70          using (StreamWriter par = new StreamWriter(key))
71          {
72              for (int i = 0; i < tmp; i++)
73              {
74                  if (key == txtFilePath[0])
75                  {
```

```
76                  par.WriteLine("{0}\t{1}",stopNum[i],TSSeq[i
                        ].TotalMilliseconds);
77              }
78              else if (key == txtFilePath[1])
79              {
80                  par.WriteLine("{0}\t{1}", stopNum[i], TSPar[
                        i].TotalMilliseconds);
81              }
82          }
83          par.Close();
84      }
85  }
86
87      Console.WriteLine("␣");
88      Console.WriteLine("Data␣is␣save␣to␣:");
89      Console.WriteLine("{0}", txtFilePath[0]);
90      Console.WriteLine("{0}", txtFilePath[1]);
91  }
```

C# code A.3: Program.cs

A.2 Results

The results were not that surprising and met the expectations quite well and are listed in TAB. A.1 and illustrated on FIG. A.1 . For smaller sample sizes, the sequential loop is quicker than the parallel but at an approximate sample size of 10^6 the parallel execution is quicker that the sequential execution. It should however be noted that the sample sizes from $10^0 \rightarrow 10^4$ are executed, in parallel, literally in the blink of an eye.

Table A.1: *First couple of sample sizes for the computation times also illustrated on* FIG. *A.1.*

Sample size	10^0	10^1	10^2	10^3	10^4
Parallel	15.3889	15.4283	15.5194	15.6405	37.4217
Sequential	0.0069	0.0151	0.0303	0.1371	1.1856

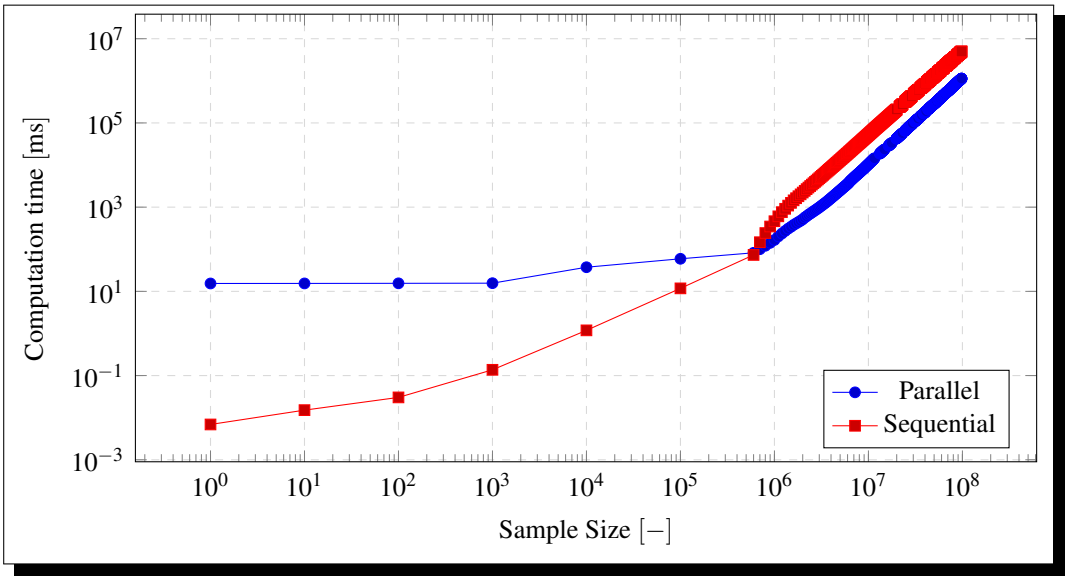

FIGURE A.1: Study of computation times from the test program. Computations are performed on a INTEL® CORE™ i7-4710MQ with 20 Gb of DDR3 RAM.

Appendix B

Material Parameters

This appendix presents the results from the determination of material parameters for the combinations in Table 4.1.

Table B.1: *Fitted parameters for combination 1.*

E [MPa]	ν [-]	c [kPa]	φ [deg]	ψ [deg]	Err. func. 1	Err. func. 2	Err. func. 3 %
46.00	0.30	20.00	22	22	0.92	0.18	2.17
49.00	0.20	30.00	20	20	0.58	0.09	1.31
49.00	0.25	30.00	19	19	0.79	0.18	1.77
43.00	0.30	30.00	20	20	0.63	0.10	2.85
46.00	0.25	30.00	20	20	0.30	0.05	1.99
40.00	0.35	30.00	19	19	0.77	0.16	3.47
46.00	0.30	30.00	19	19	0.90	0.16	1.92
43.00	0.25	40.00	19	19	0.92	0.19	4.18
46.00	0.20	40.00	19	19	0.64	0.14	2.79
49.50	0.25	20.00	22	22	0.87	0.19	2.07
46.50	0.30	20.00	22	22	0.98	0.18	2.13
49.50	0.20	30.00	20	20	0.29	0.04	0.65
49.50	0.25	30.00	19	19	0.88	0.17	1.94
43.50	0.30	30.00	20	20	0.83	0.13	2.64
40.50	0.35	30.00	19	19	0.72	0.14	3.00
46.50	0.20	30.00	21	21	0.91	0.14	3.12
46.50	0.25	30.00	20	20	0.28	0.04	1.53
46.50	0.20	40.00	19	19	0.78	0.17	2.65
50.00	0.25	20.00	22	22	0.88	0.19	2.02
41.00	0.35	20.00	22	22	0.96	0.18	3.39
50.00	0.20	30.00	20	20	0.00	0.00	0.00
50.00	0.25	30.00	19	19	0.99	0.18	2.28
41.00	0.35	30.00	19	19	0.79	0.14	2.68
47.00	0.20	30.00	21	21	0.78	0.13	2.62
47.00	0.25	30.00	20	20	0.45	0.07	1.32
50.50	0.25	20.00	22	22	0.95	0.19	2.16
41.50	0.35	20.00	22	22	1.00	0.17	3.02
50.50	0.20	30.00	20	20	0.28	0.04	0.65
44.50	0.30	30.00	19	19	0.72	0.17	2.22
41.50	0.30	30.00	20	20	0.82	0.13	4.32
41.50	0.35	30.00	19	19	0.92	0.15	2.56
47.50	0.20	30.00	21	21	0.73	0.14	2.24
47.50	0.25	30.00	20	20	0.69	0.11	1.46
45.00	0.30	20.00	22	22	0.96	0.20	2.80
54.00	0.20	20.00	22	22	0.94	0.20	2.77
45.00	0.25	30.00	20	20	0.92	0.13	3.20
51.00	0.20	30.00	20	20	0.56	0.08	1.30
45.00	0.30	30.00	19	19	0.70	0.16	1.88
42.00	0.30	30.00	20	20	0.61	0.10	3.72
45.00	0.20	40.00	19	19	0.83	0.13	3.54
48.00	0.20	30.00	21	21	0.76	0.16	2.05
48.00	0.25	30.00	20	20	0.96	0.15	1.86
45.50	0.30	20.00	22	22	0.90	0.18	2.41
45.50	0.25	30.00	20	20	0.60	0.09	2.56
45.50	0.30	30.00	19	19	0.78	0.15	1.77
42.50	0.30	30.00	20	20	0.53	0.09	3.23
51.50	0.20	30.00	20	20	0.63	0.09	1.72
45.50	0.20	40.00	19	19	0.69	0.12	3.10
42.50	0.25	40.00	19	19	0.95	0.17	4.51

Table B.2: *Fitted parameters for combination 2.*

E [MPa]	ν [-]	c [kPa]	φ [deg]	ψ [deg]	Err. func. 1	Err. func. 2	Err. func. 3 %
55.0	0.20	30.0	21	21	0.91	0.18	1.93
55.0	0.25	30.0	20	20	0.29	0.05	1.95
49.0	0.30	30.0	20	20	0.56	0.08	1.27
46.0	0.35	30.0	20	20	0.68	0.11	1.4
40.0	0.40	30.0	20	20	0.82	0.13	2.98
55.5	0.20	30.0	21	21	0.88	0.19	2.07
55.5	0.25	30.0	20	20	0.48	0.08	2.41
49.5	0.30	30.0	20	20	0.28	0.04	0.63
43.5	0.35	30.0	20	20	0.96	0.14	3.34
46.5	0.35	30.0	20	20	0.94	0.15	1.75
50.0	0.30	30.0	20	20	0	0	0
44.0	0.35	30.0	20	20	0.63	0.1	2.7
56.0	0.25	30.0	20	20	0.72	0.11	2.91
53.0	0.25	30.0	20	20	0.82	0.13	1.53
44.5	0.35	30.0	20	20	0.35	0.06	2.11
50.5	0.30	30.0	20	20	0.27	0.04	0.63
56.5	0.25	30.0	20	20	0.96	0.14	3.43
53.5	0.25	30.0	20	20	0.6	0.1	1.3
45.0	0.35	30.0	20	20	0.29	0.05	1.63
51.0	0.30	30.0	20	20	0.55	0.08	1.25
54.0	0.25	30.0	20	20	0.66	0.11	1.6
45.5	0.35	30.0	20	20	0.45	0.07	1.35
51.5	0.30	30.0	20	20	0.81	0.12	1.87
57.5	0.20	30.0	20	20	0.85	0.13	2.62
58.0	0.20	30.0	20	20	0.69	0.11	2.78
54.5	0.20	30.0	21	21	0.99	0.18	1.96
54.5	0.25	30.0	20	20	0.28	0.04	1.56
48.5	0.30	30.0	20	20	0.85	0.13	1.91
58.5	0.20	30.0	20	20	0.58	0.09	3.03
59.0	0.20	30.0	20	20	0.53	0.09	3.34
59.5	0.20	30.0	20	20	0.52	0.09	3.71
60.0	0.20	30.0	20	20	0.62	0.11	4.1

Table B.3: *Fitted parameters for combination 3.*

E [MPa]	ν [-]	c [kPa]	φ [deg]	ψ [deg]	Err. func. 1	Err. func. 2	Err. func. 3 %
49.00	0.40	30.00	20	20	0.51	0.08	1.18
52.00	0.40	30.00	20	20	0.97	0.15	2.31
55.50	0.35	30.00	20	20	0.87	0.13	1.56
49.50	0.40	30.00	20	20	0.25	0.04	0.59
56.00	0.35	30.00	20	20	0.68	0.10	1.41
50.00	0.40	30.00	20	20	0.00	0.00	0.00
50.50	0.40	30.00	20	20	0.25	0.04	0.58
56.50	0.35	30.00	20	20	0.51	0.08	1.46
51.00	0.40	30.00	20	20	0.49	0.08	1.16
57.00	0.35	30.00	20	20	0.38	0.06	1.67
51.50	0.40	30.00	20	20	0.73	0.11	1.74
57.50	0.35	30.00	20	20	0.37	0.06	1.99
48.50	0.40	30.00	20	20	0.77	0.12	1.78
58.00	0.35	30.00	20	20	0.44	0.07	2.38
58.50	0.35	30.00	20	20	0.57	0.10	2.80
59.00	0.35	30.00	20	20	0.73	0.13	3.25
59.50	0.35	30.00	20	20	0.90	0.16	3.71

Appendix C

Finite Element Theory

This Appendix presents the relevant theory which is necessary to understand what goes on behind the scenes in the program. The relevant theory is based on [11].

In structural mechanics different types of non linearity include the following

Material non linearity Material properties are functions of stress states or strain.

Geometric non linearity Deformations are large enough that equilibrium with respect to the deformed geometry.

C.1 Non-linear Finite Element formulation

The general linear finite element formulations is given by

$$[\mathbf{K}]\ \{\mathbf{u}\} = \{\mathbf{f}\} \tag{C.1}$$

When working with non-linear problems it is not possible to obtain a solution in a single step. This implies that incremental solutions must be employed such as

$$[\mathbf{K}]\ \Delta\{\mathbf{u}\} = \Delta\{\mathbf{f}\} \tag{C.2}$$

The finite element method can either be used as applying a displacement or a force increment. In the present case incremental displacement are added and the resulting force is determined.

When imploying non-linear fem, and solving systems, an equlibrium between internal and external forces must be established

$$\mathbf{r}_k = \mathbf{f}_{int} - \mathbf{f} = 0 \tag{C.3}$$

where $\mathbf{r}_k$ is the residual force vector and $\mathbf{f}$ is the external for vector which is establish by

$$\mathbf{f}_{int} = \mathbf{B}^T\ \sigma\ w \tag{C.4}$$

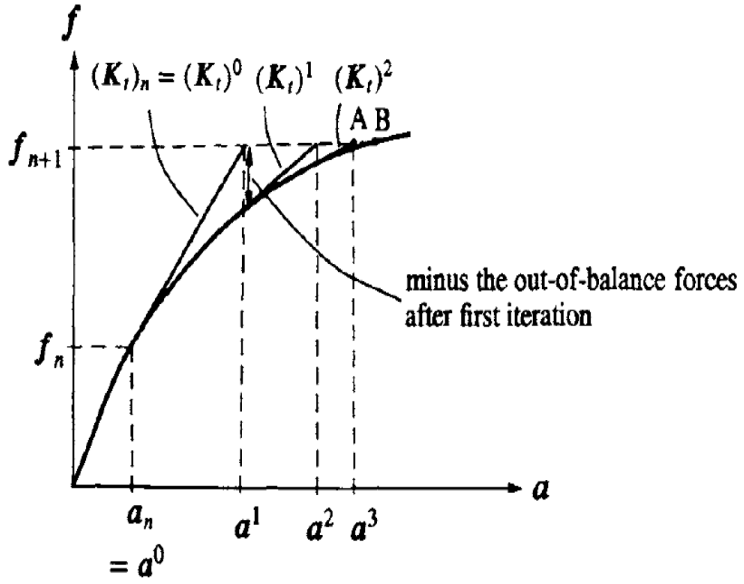

FIGURE C.1: Newton-Raphson scheme.

where w is the gauss points weight since numerical integration is employed, $\mathbf{B}^T$ is the transposed strain interpolation matrix and σ are the stresses.

What is important to realise is that at any step the following quantities are known: The current displacement, force, strain and stress. The equations of equilibrium for the entire body are known as global equations and the constitutive equations for a single integration point are know as local equations.

For the case of the Newton-Raphson scheme, the tangent stiffness is updated in each iteration which is illustrated on Figure C.1 . The calculation of the matrix is done by

$$\mathbf{k}_{elem} = \mathbf{B}^T \, \mathbf{D} \, \mathbf{B} \, \mathbf{w} \tag{C.5}$$

Appendix D

Material Models

This Appendix presents the relevant theory used in the program to model the properties of the material. The relevant theory is based on [12] and [9].

D.1 The Mohr-Coulomb model

While undergoing plastic straining, the yield surface is fixed in principal and deviatoric stress space, and for any stress state inside the yield surface the response is elastic. If the stress state is however located on the yield surface the behaviour of the material changes and the response becomes plastic and subsequently strains become irreversible. The yield surface is comprised of six planes in principal stress plane, forming an hexagonal pyramid structure as illustrated by Figure D.1

By ordering the principal stresses according to the following

$$\sigma_1 \geq \sigma_2 \geq \sigma_3 \tag{D.1}$$

The Mohr-Coulomb criterion and its corresponding plastic potential in principal stresses

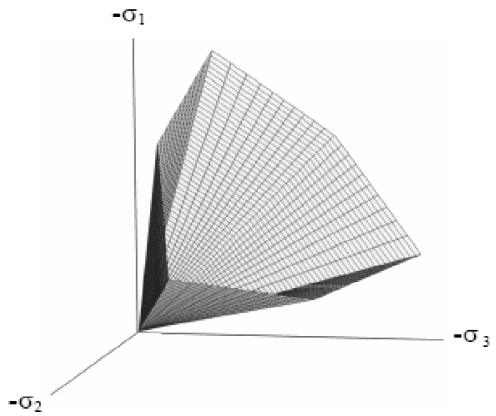

FIGURE D.1: The Mohr-Coulomb yield surface in principal stress space.[13, Figure 4]

are usually written as

$$f(\sigma) = k\,\sigma_1 - \sigma_3 - \sigma_c \tag{D.2}$$
$$g(\sigma) = m\,\sigma_1 - \sigma_3 \tag{D.3}$$

where σ_c is the uniaxial compressive strength given by

$$\sigma_c = 2\,c\sqrt{k} \tag{D.4}$$

and

$$k = \frac{1+\sin\varphi}{1-\sin\varphi} \tag{D.5}$$
$$m = \frac{1+\sin\psi}{1-\sin\psi} \tag{D.6}$$

where φ is the internal friction, c is the cohesion and ψ is the dilation angle. The model accounts for non-associated flow, ans is widely used as a result of its simplicity, and as already mentioned, consists of the five material parameters Young's modulus, E, Poisson's ratio, ν, cohesion, c, friction angle, φ and the dilation angle ψ.

D.2 Return Mapping

The basic relation in small strain plasticity is that a strain increment is composed of an elastic and plastic part:

$$d\varepsilon = d\varepsilon^e + d\varepsilon^p \tag{D.7}$$

For perfect plasticity, plastic strains during yielding

$$f(\sigma) = 0 \tag{D.8}$$
$$\left(\frac{\partial f}{\partial \sigma}\right)^{\mathrm{T}} = 0 \tag{D.9}$$

with f being the yield function and σ being the stress vector. Stress and strain vectors are

$$\sigma = \left[\sigma_x\ \sigma_y\ \sigma_z\ \tau_{xy}\ \tau_{xz}\ \tau_{yz}\right]^{\mathrm{T}} \tag{D.10}$$
$$\varepsilon = \left[\varepsilon\ \varepsilon\ \varepsilon\ 2\varepsilon_{xy}\ 2\varepsilon_{xz}\ 2\varepsilon_{yz}\right]^{\mathrm{T}} \tag{D.11}$$

Equation (D.9) describes a surface in stress space, and a stress state inside the surface is elastic where $f(\sigma) < 0$. It is possible to related elastic stress increments to elastic strain increments by Hooke's law

$$d\sigma = \mathbf{D}\,d\varepsilon^e \tag{D.12}$$
$$= \mathbf{D}\,(d\varepsilon - d\varepsilon^p) \tag{D.13}$$
$$= \mathbf{D}\,d\varepsilon - \mathbf{D}\,d\varepsilon^p \tag{D.14}$$

The constitutive matrix is given by

$$\mathbf{D} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & & & \\ \nu & 1-\nu & \nu & & & \\ \nu & \nu & 1-\nu & & & \\ & & & \frac{1}{2}-\nu & & \\ & & & & \frac{1}{2}-\nu & \\ & & & & & \frac{1}{2}-\nu \end{bmatrix} \tag{D.15}$$

since linear, isotropic elasticity is considered. A finite stress increment comes from integration of Equation (D.14)

$$\Delta\sigma = \Delta\sigma^{elastic} - \Delta\sigma^{plastic} \tag{D.16}$$

Rewriting Equation (D.16) into the following equation can be illustrated on Figure D.2

$$\sigma^{\mathrm{C}} = \sigma^{\mathrm{B}} - \sigma^{P} \tag{D.17}$$

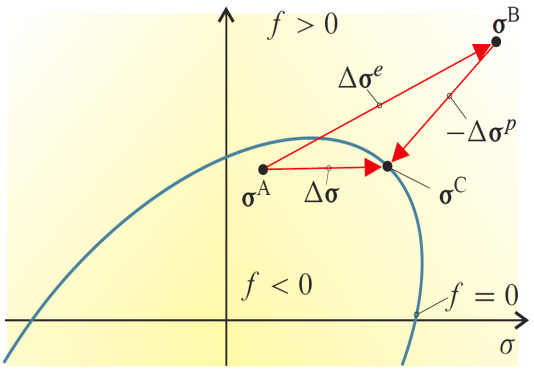

FIGURE D.2: Principle of return mapping. [9, Figure 3.1]

D.3 Stress update in principal stress space

The predictor stress is transformed into principal stress space and returned to the yield surface. From here the updated stress is transformed back into the original coordinate system along with the constitutive matrices, which are also formed in principal stress space.

When the predictor stress is transformed into principal stress space, and returned to the yield surface, three different cases of stress returns should be considered for the Mohr-Coulomb criterion:

- Return to the yield plane.
- Return to a line between two intersecting yield planes.
- Return to the apex point.

No matter which case is considered, the stress must be updated and constitutive matrices must be formed. This is done by the principle in Figure D.3.

1. Determine principal predictor stresses, $\bar{\boldsymbol{\sigma}}^{\mathrm{B}}$ from $\boldsymbol{\sigma}^{\mathrm{B}}$.
2. Determine the stress region in order to assess the type of stress return.
3. Calculate returned principal stress, $\bar{\boldsymbol{\sigma}}^{\mathrm{C}}$.
4. Calculate the appropriate infinitesimal constitutive matrix in principal stress space, $\bar{\mathbf{D}}^{ep}$.
5. Calculate consistent constitutive matrix in principal stress space, $\bar{\mathbf{D}}^{ep}$.
6. Transform $\bar{\boldsymbol{\sigma}}^{\mathrm{C}}$ and $\bar{\mathbf{D}}^{ep}$ back into the original co-ordinate stress space, using standard co-ordinate transformation methods.

FIGURE D.3: Return mapping principle. [9, Table 4.1]