DEM simulation of cemented sand under drained triaxial condition

Doszhan Tuzelbayev, B. Eng.

Submitted in fulfilment of the requirements
for the degree of Master of Science
in Civil & Environmental Engineering

NAZARBAYEV
UNIVERSITY

School of Engineering and Digital Sciences
Department of Civil & Environmental Engineering
Nazarbayev University

53 Kabanbay Batyr Avenue,
Astana, Kazakhstan, 010000

Supervisors: Sung-Woo Moon, Jong Kim

March 28, 2025

Declaration

I hereby, declare that this manuscript, entitled “DEM simulation of cemented sand under drained triaxial condition”, is the result of my own work except for quotations and citations which have been duly acknowledged.

I also declare that, to the best of my knowledge and belief, it has not been previously or concurrently submitted, in whole or in part, for any other degree or diploma at Nazarbayev University or any other national or international institution.

Name: Doszhan Tuzelbayev

Date: March 28, 2025

Abstract

This study presents a discrete element method (DEM)-based numerical investigation into the mechanical behavior of cemented sand under triaxial compression. The primary objective is to optimize the simulation parameters by minimizing the deviation between numerical and experimental stress-strain responses. Bayesian optimization via the Optuna framework was employed to systematically explore the parameter space, utilizing an objective function that combines root mean square error (RMSE) and Euclidean distance between experimental and simulated data. Sensitivity analysis was performed on 15 parameters, identifying 10 as the most influential in replicating cemented sand behavior. Through 200 optimization iterations, the best-fit parameters were determined, capturing essential mechanical properties such as stiffness, cohesion, and friction. Numerical simulations provided insights into displacement evolution and contact force distribution, demonstrating strong agreement with experimental observations. At the end, a parallel coordinates plot was employed to visualize the multidimensional parameter space, highlighting correlations and trade-offs among the variables. The post-optimization analysis validated the consistency of the sensitivity study, reinforcing the reliability of the calibrated parameters. The findings contribute to the broader understanding of DEM calibration for cemented geomaterials, particularly in applications requiring high-fidelity simulation of mechanical responses. This research provides a systematic approach for improving DEM model accuracy, reducing reliance on trial-and-error parameter selection, and paving the way for further studies in cemented sand behavior modeling.

Table of Contents

List of Abbreviations

DEM	– Discrete Element Methods
OPC	– Ordinary Portland cement
CSA	– Calcium Sulfoaluminate cement
$PFC3D$	– "Particle Flow Code in Three Dimensions" software (by Itasca)
TPE	– Tree-structured Parzen Estimator
$\bar{\sigma_c}$	– parallel-bond tensile strength
$\bar{c}$	– parallel-bond cohesion
$\overline{k_n}$	– parallel-bond normal stiffness
$\overline{k_s}$	– parallel-bond shear stiffness
$\bar{\phi}$	– parallel-bond friction angle
$\bar{E}$	– bond effective modulus
μ	– friction coefficient
k_n	– linear normal stiffness
k_s	– linear shear stiffness
E	– linear effective modulus
ζ	– ball particle's local damping factor
n	– porosity
R_{bond}	– bond gap ratio (ratio of bond gap to the smallest particle diameter)
$N_{pressure}$	– pressure divisions (number of confining pressure increments)
M_{scale}	– particles magnification scale

List of Tables

List of Figures

Chapter 1 – Introduction

1.1 Overview

Soil stabilization is an important aspect of geotechnical engineering as it improves mechanical and physical characteristics of soil, such as strength, durability, and load-bearing capacity. There are several methods for stabilization. One of the example is a chemical stabilization, which involves chemical additives or industrial by-products to alter soil properties (Mustafayeva et al., 2024). In soil stabilization, selecting suitable binding materials plays a crucial role. It can be achieved using cement, fly ash, lime, fibers, or hydro-carbon emulsion (Andavan & Maneesh Kumar, 2020).

Many types of cement can be utilized to stabilize the soil. Ordinary Portland Cement (OPC), which is affordable and widely used, is the most typical type. Moreover, many studies were conducted on the effects of this cement as a binding material in soil stabilization. It helps soil to gain more strength and resist liquefaction. However, the manufacturing stage of OPC presents an environmental problem as it produces many carbon emissions (Rauf et al., 2025).

Due to its greater early strength and low carbon emission during manufacture, Calcium Sulfoaluminate cement (CSA) is currently attracting more attention (Mustafayeva et al., 2023; Rauf et al., 2024; Sagidullina et al., 2024; Vinoth et al., 2018). As a result of the synthesis of ettringite from the hydration of ye'elimite, CSA-treated sands had better initial strength development, according to studies comparing OPC and CSA cement-treated sands (Subramanian et al., 2018).

It is important to point out that any process of soil stabilization changes the properties of the sand. Thus, the Discrete Element Method (DEM) can be applied to identify the micro-properties of sand grains. DEM simulates the force and moment of each particle of sand individually using special parameters for the particles and the bonds between them. For the sake of saving the computing effort, DEM particles are often enlarged several times. DEM simulation result is compared with real-life laboratory experiments to characterize the micro parameters of soil particles. Some studies have already been conducted on cement-treated soils using DEM. Most of them use compressive triaxial tests for reference as they generate an accurate sample mechanical property. One of the papers used DEM to simulate the complex macroscopic mechanical behavior

of granular materials (de Bono et al., 2015). It is accomplished by detecting particle micro-properties and interaction rules (Feng et al., 2017).

1.2 Thesis Statement

The need for precise and efficient soil characterization methods arises from the complexity of soil behavior and the increasing demand for sustainable infrastructure development. Several studies have been conducted on soils treated by Ordinary Portland Cement using DEM. However, the problem is that there is a need for comprehensive research on the mechanical response of Calcium Sulfoaluminate cement-treated soil using advanced simulation techniques like DEM. The generalization of soil behavior simulation is important to bridge the gap between laboratory experiments and computational modeling of cemented sands. Thus, the primary goal of this study is to use DEM to determine and assess the micro parameters of CSA-treated soil particles using drained triaxial tests. Furthermore, the study aims to automate the soil characterization process using Optuna optimizaition framework for efficient parameter estimation. The theoretical and academic context of this study aligns with the fields of geotechnical engineering, computational methods for soil mechanics, and optimization algorithms for parameter estimation.

1.3 Objectives

The main objectives of the thesis can be described in the following:

1) Selecting a reference paper with experimental data that provides comprehensive data on CSA cement-treated sands subjected to drained triaxial testing. This reference paper will be the basis for gradation analysis and stress-strain relationship characterization.
2) Conducting DEM simulations of drained triaxial tests using state-of-the-art methods to accurately replicate the mechanical behavior of CSA cement-treated sands under varying loading conditions.
    a) Define membrane design
    b) Define the contact model between particles
    c) Set initial values and boundaries for DEM simulation parameters
3) Validating the DEM simulation results by comparing them with empirical data obtained from laboratory experiments on CSA-treated sands.
    a) Automating the soil characterization process using the Optuna algorithm for optimization and more precise parameter estimation.

Chapter 2 – Literature Review

First of all, this section establishes reference experimental data of triaxial test with a CSA cement-treated specimen. This is a required step to set a target data for DEM simulations. After that, throughout literature review was conducted to anaylze DEM simulation design methods and procedures for the triaxial test.

2.1 Experimental data

Ocheme et al. (2022) contain information that can be used as a reference for the simulation as its subject is identical to the research topic of this paper. Moreover, it consists of details about the triaxial test setup, which can help model the test using the discrete element method (DEM). Ocheme et al. (2022) studied the strength of CSA cement-treated sands by conducting a triaxial test under high confining pressures as their primary method of investigation.

It is worth noting that contrasting confining pressures result in different soil behavior in compressive triaxial tests. A paper by de Bono et al. (2015) claims that treated sands get improved stiffness, peak strength, and dilation under conventional confining pressure. Such pressure level also leads to brittle behavior of the sample (de Bono et al., 2015; Obermayr et al., 2013). However, the brittleness of the sample limits the advantages of the cementization due to the nature of the binding effects of cement (Cui et al., 2020). Thus, increasing the confining pressure restrains the strength development due to cement treatment. This change also makes the sample show ductile behavior (de Bono et al., 2014).

Ocheme et al. (2022) used quartz, CSA cement, and gypsum for the samples. Also, 30% of cement was substituted by gypsum to improve the sample strength. The sample was prepared with the 7% cement content and optimal moisture content of 16.75%. After that, all samples were cured for 7 days. The sample was placed on porous stone and filter paper. After that, the sample was wrapped in a latex membrane. Finally, a consolidated drained (CD) triaxial test was performed to show stress-strain at 1.0 MPa confining pressure, as illustrated in Figure 2.1 below.

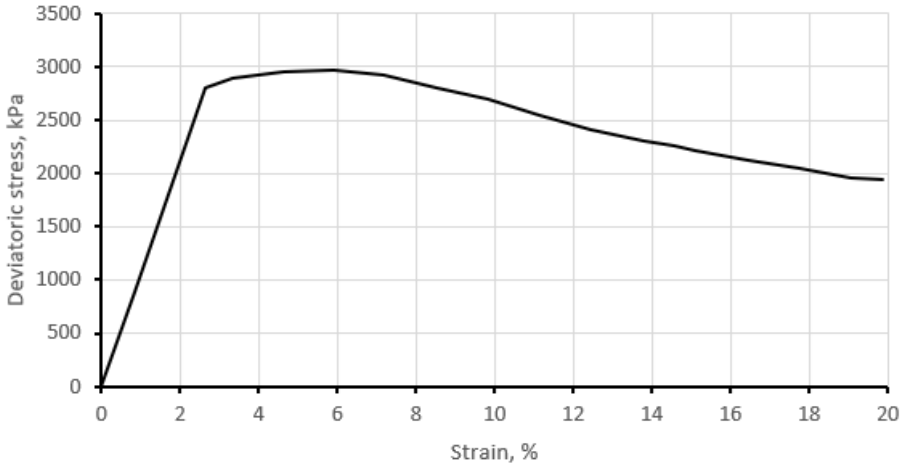


Figure 2.1: Stress-strain plot of the CSA-treated quartz sand sample at 1.0 MPa (Ocheme et al., 2022).

At confining pressures of 1.0 MPa failure stress was at 2975.2 kPa. In case of volumetric change, sample was slightly compressed in the initial stage of the test. The samples at lower confining pressure experienced increased volume due to dilation.

2.2 Literature review of DEM simulation

Several studies have used discrete element methods on cemented sands in triaxial test setup (de Bono et al., 2015; Feng et al., 2017; Yang et al., 2019). This section reviews methods and advantages of some research papers related to DEM simulations of triaxial test. First of all, the design of membrane in DEM should be addressed, ensuring that it acts as a boundary that maintains confinement of specimen particles. However, in DEM simulation, all particles are discrete and in spherical shape. Contact forces and moment tranfer between particles also should be established to accurately the behaviour of sand particles bonded by CSA cement.

2.2.1 Membrane design

Lu (2010) highlighted the challenge of simulating flexible membranes in triaxial tests within the framework of DEM. Conventional DEM methods encounter challenges in accurately modeling continuum-like boundary conditions, like those seen in soil tests with flexible membranes. These conditions, dictated by elasticity principles, pose a substantial challenge in fully capturing their behavior in DEM simulations, which are primarily governed by Newton's second

law and force-displacement principles. Nonetheless, recent progress presents novel solutions to overcome this obstacle, enabling more effective simulation of membranes within DEM frameworks.

Yang et al. (2019) conducted a DEM simulation using Yade software. In their research, they performed two types of triaxial test simulation: constant volume and coupled methods. The constant volume method assumes unchanged volume; thus, rigid walls can be used to simulate the membrane. The coupled method, in contrast, requires a flexible membrane. Yang et al. (2019) used other academic sources to reference actual data. They used the PFacet element to model flexible membranes. Particle size distribution was generated randomly. Both confining and axial stresses were simulated by applying a velocity to objects. The result showed that rigid membrane simulations show higher strength than flexible. Though rigid membrane tests show a good correlation, flexible membranes are more accurate regarding stress-strain and volumetric strain curves (Binesh et al., 2018; Yang et al., 2019). Moreover, using a flexible membrane allows the development of shear bands (Jiang et al., 2013). Thus, the application of flexible membranes has a more realistic representation of the simulation process than rigid ones.

There are several ways to generate flexible membranes. de Bono et al. (2015) used PFC3D software to simulate sands treated by Ordinary Portland cement (OPC) under high confining pressure. They modeled a triaxial test using a flexible membrane with contact bonds that consist of small particles. As de Bono et al. (2015) claim, it helps to sustain a confining pressure without limiting the deformations. The size of sleeve particles is a third of the smallest sand grains to bound and hold the soil. And the bonds between these membrane particles are generated by microscopic bonds with no moment transfer.

Binesh et al. (2018) generated a flexible membrane by incorporating MATLAB code to identify the boundary particles of the sample as a membrane. Even though the proposed method of the flexible membrane has better results than the rigid membrane, the confining pressure forces are applied to each external particle separately. Thus, it cannot behave like one whole body to propagate the forces.

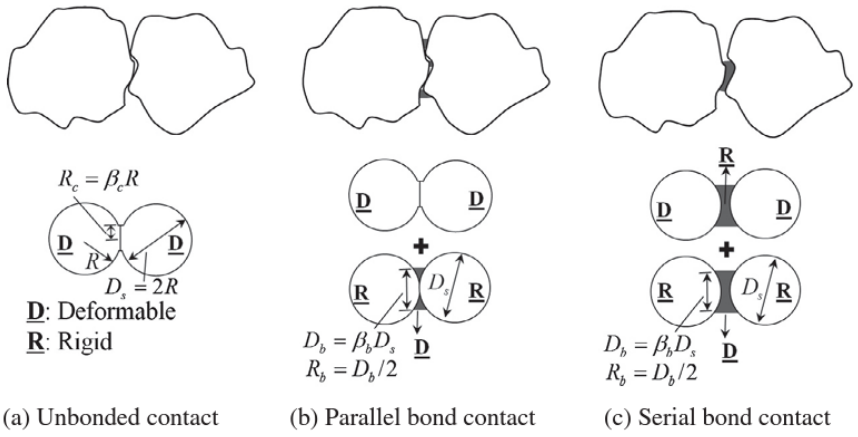
The membrane method used in the study by Ng and Asce (2004) involved a novel hydrostatic boundary condition. This boundary simulated the behavior of chamber fluid without the need to model the rubber membrane typically found in experimental setups. The pressure

resulting from particle (ellipsoids) interaction with the hydrostatic boundary was determined analytically based on geometric considerations. This method was seamlessly integrated into an existing ellipsoidal DEM code. Findings indicated an enhanced friction angle compared to periodic boundaries, demonstrating close agreement with experimental observations.

2.2.2 Contact model

Yang et al. (2019) and Pu et al. (2017) used moment transfer law (MTL) on Yade software to imitate rolling resistance between smooth particles. Both of their research focused on sands strengthened by microbially induced carbonate precipitation (MICP). As MICP enhances the soil, a cohesive bond model should be used between particles (Yang et al., 2019). That is why a cohesive shear strength parameter was introduced to each particle.

Shen et al. (2016)considered two types of contact models: parallel and serial bond contact models. The parallel bond contact model assumes there is a collision between particles. Also, it is considered that there are two parallelly joined contacts: physical and binding components. On the other hand, the serial bond contact model does not require physical contact. In this case, the aforementioned contacts are joined in series. Both contact models can be referred to in Figure 2.2. Both models reached a critical state in their simulation mostly because of shear and rolling pressure.



(a) Unbonded contact (b) Parallel bond contact (c) Serial bond contact

Figure 2.2: Idealized representation of microstructure in cemented sand (Shen et al., 2016)

de Bono et al. (2015) used a build-in parallel bonds model of "Particle Flow Code in 3 dimensions" software to set the parameters of bonds between particles. This paper consists of

excellent details about the model setup and parameters. Though this paper is very similar to the current study's research topic, it considers only ordinary Portland cement-treated sands and lacks some required steps, such as the energy release of the generated particles.

One of the unconventional ways of designing the contact model was introduced in a paper by Obermayr et al. (2013). They assumed bond behavior between each particle as a linear Timoshenko beam. This approach also makes the simulation process a little bit simpler.

A study by Ning et al. (2017) employed the Hertz-Mindlin contact model to depict particle interaction, incorporating viscous damping at contact points to mitigate non-physical vibrations. This damping, proportional to relative particle velocities, suits non-collisional scenarios, accounting for shear wave propagation and large deformations in triaxial compression tests. Cementation was modeled via parallel bonds, which transmit normal and shear forces and moments at contact points, mimicking the effects of real cementation to some extent. While not replicating all cementation effects, parallel bonds provide connections between particles, calibrated based on physical tests and detailed in subsequent sections.

The same contact model was used in a study by Li et al. (2017). They generated three particle types for their simulation: sand, cementing, and membrane particles. Contact stiffness was estimated using the Hertz-Mindlin theory, accounting for normal and tangential stiffnesses based on confining pressure, shear modulus, and Poisson's ratio. Cementation was simulated by generating tiny cementing particles around sand particle contacts and attaching them via parallel bonds, allowing for better control of cement content and closely resembling real cementation formation.

2.2.3 Other DEM parameters

DEM simulation is a complex computation process where each grain of soil can move freely with specified contact model bonds. To reduce the computational cost, most papers magnify the size of grain particles (Ahmadi & Sizkow, 2020; Feng et al., 2017; Rakhimzhanova et al., 2019). This process is called particle scaling, which decreases particle numbers and significantly reduces the computation process and time burden.

There are different DEM simulation software applications, and each program has its required parameters to be specified prior to computation. On Yade software, Yang et al. (2019)

concluded that the following parameters were essential for DEM simulation: normal and tangential contact stiffness, interparticle friction angle, rolling stiffness coefficient, and plastic moment limit coefficient.

The paper by de Bono et al. (2015) modeled a triaxial test using a flexible membrane and contact bonds that consist of small particles. The following parameters of the sand particles were set: initial void ratio – 0.55, coefficient of uniformity – 2, minimum sand particle diameter – 2 mm, and normal particle stiffness – 10e6 N/m. A build-in parallel bonds model of the software was applied to set the parameters of bonds between particles. Minimum sand particle diameter affects the number of particles generated.

Wu et al. (2021) discuss a DEM simulation of sands treated by ordinary Portland cement using PFC2D. A biaxial compression test was simulated as their study is limited to two dimensions. To characterize the parameters of the sand, they used a trial-and-error method by conducting several tests in DEM and comparing the results with other papers. Most parameters of the sand were assumed during the simulation process using a parallel bond model. When modeling a flexible membrane, they concluded that its stiffness and strength have negligible effects on a sample result. Wu et al. (2021) also considered the effect of irregularly shaped particles using a bubble packing algorithm to distribute clump size uniformly. For data analysis, they used Weibull's distribution concept with a different homogeneity index set, as it corresponded with the failure strength deviation of the sample. Stress-strain and volumetric strain curves were used to compare the simulation with the real data. Their results showed that within 0.5% of strain, the simulation fits the trend of the experimental data. Upon reaching the failure state of the sample, there were minor deviations. Wu et al. (2021) concluded that the homogeneity index and confining pressure greatly affect the test result. This paper contains a very detailed process of cemented sand simulation in compression tests. Moreover, their sample parameters can serve as a good starting point for the follow-up research on this topic. However, their research is limited to 2D space, and the energy release method during the particle generation process was not mentioned.

After reviewing several papers, Table 2.1 was generated to show and compare different parameters for particles, bonds, and membranes suggested by different studies on cemented soils. Later, Table 2.1 can be used to calibrate the parameters of CSA cement-treated sands.

Table 2.1: Comparison of DEM parameters from different sources.

Sources	(Wu et al., 2021)	(de Bono et al., 2015)	(Obermayr et al., 2013)	(Jiang et al., 2011)	(Wang & Leung, 2008)	(Feng et al., 2017)
Particle properties						
density, kg/m^3	1,930	2,650	1,400	2,600	2,650	
void ratio		0.55		0.27	0.2	
number of balls	28,566	6,759	varies	24,000	2,000	14,000
damping coefficient	0.7	0.7		0.5		
friction coefficient	0.25	0.5	0.32	0.5	0.55	0.5
effective modulus, N/m^2			8e8			
normal stiffness, N/m	1e7	1e6		7.5e7	5e7	2e8
shear stiffness, N/m	1e7	1e6		5e7	5e7	5e6
Bond properties						
contact model	parallel bond	linear springs	Timoshenko beam	linear springs	cement particles	
normal strength, N/m^2		1.592e7			3e6	5e6
shear strength, N/m^2		1.592e7			3e6	1e6
Membrane properties						
friction coefficient	0	0	0	0	0	
density, kg/m^3	1,000	1,000		1,000	1,800	
effective modulus	1e8		2e8			
normal strength, N/m^3	1e200			1e100	1e300	
shear strength, N/m^3	1e200			1e100	1e300	
thickness, mm		1.33		2.0	0.05	
normal stiffness, N/m				3.75e6	5e6	2e8
shear stiffness, N/m				2.5e6	5e6	2e8

2.3 Review of DEM simulation procedure

In PFC3D, an example model resembles a triaxial test entirely written in PFC3D's own coding language – FISH code (Itasca, 2019). That example shows how to execute triaxial tests on a bonded-particle model (BPM) using structural elements based on shells (Itasca, 2021b). This kind of model could be studied solely with PFC interface. However, a sizable amount of FISH coding would be needed for the sleeved wall to behave as an elastic membrane in that scenario.

2.3.1 Model Geometry design

In this example, a flexible shell is converted into a membrane, and particles are produced inside the membrane. The walls are enslaved to shell-based structural elements. BPM makes contact with wall facets, and the contact forces and moments at each ball-facet contact are utilized to calculate the equivalent force system that should be applied to the associated shell nodes. The specimen is formed in this model with no beginning tension for the sake of simplicity, and the platens are shifted at a fixed speed (Itasca, 2021b).

Figure 2.3 illustrates the initial lines of FISH code setting up model parameters. PFC balls are contained within a cylindrical sleeve composed of FLAC3D shells. The "`model random`" command establishes a specific random seed to ensure the reproducibility of the initial ball cloud when executing data files. Changing the random seed generates a different ball set, but they should yield the same outcomes statistically. In this instance, as depicted in Figure 2.3, line 7, the random seed used is 10001.

```
 5  model new
 6  model title 'Sleeved Triaxial Test of a Bonded Material'
 7  model random 10001
 8  model domain extent -2 2 condition destroy
 9  model largestrain on
10  model mechanical timestep scale
```

Figure 2.3: Initial settings.

All PFC model elements, such as balls, clumps, walls, and contacts, are confined within a user-defined "`model domain`" distinct from zones and structural elements. This constraint simplifies contact detection and spatial searches. As indicated in Figure 2.3, line 8, the domain range spans from -2 to 2 along all three axes, with elements outside this range being removed.

Since PFC operates in large-strain mode by default, this mode must be explicitly activated for zones and structural components, as shown in Figure 2.3, line 9. By default, PFC incorporates the actual inertia characteristics of balls and clumps when solving dynamic equations of motion. It's advisable to employ timestep scaling in PFC, which mirrors FLAC3D's default operation, as depicted in Figure 2.3, line 10.

Figure 2.4 shows the initial steps of constructing the test's shape. Lines 12-16 specify the basic parameters of the cylinder. In line 18-29, the shell's edges are defined using the command "`geometry edge create`" via arcs. The keyword "`by-arc origin`" defines the origin of the arc,

and "`start`" specifies the starting direction and the arc radius. The "`end`" location sets the plane and sweep angle of the arc, but the distance from the origin is ignored. The "`segments`" keyword divides the arc by the specified segments, which is 6 in this example. These geometries create the cylindrical sleeve so it can be imported as a shell. To make the cylindrical shell, arcs are produced as geometry edges and extruded vertically (Figure 2.4, line 30).

```
12 [rad = 1.0]
13 [height = 3.0]
14 [segments = 6]
15 [halfLen = height/2.0]
16 [freeRegion = height/2.0*0.8]
17
18 geometry edge create by-arc origin (0,0,[-halfLen]) ...
& 19    start ([rad*(-1)],0,[-halfLen]) end (0,[rad*(-1)],[-halfLen]) ...
& 20    segments [segments]
21 geometry edge create by-arc origin (0,0,[-halfLen]) ...
& 22    start (0,[rad*(-1)],[-halfLen]) end ([rad],0,[-halfLen]) ...
& 23    segments [segments]
24 geometry edge create by-arc origin (0,0,[-halfLen]) ...
& 25    start ([rad],0,[-halfLen]) end (0,[rad],[-halfLen]) ...
& 26    segments [segments]
27 geometry edge create by-arc origin (0,0,[-halfLen]) ...
& 28    start (0,[rad],[-halfLen]) end ([rad*(-1)],0,[-halfLen]) ...
& 29    segments [segments]
30 geometry generate from-edges extrude (0,0,[height]) segments [segments*2]
```

Figure 2.4: Shell edge formation.

The shell is made from the created geometry using the "`structure shell import`" command (Figure 2.5, line 32). To simplify boundary conditions, elements and nodes are assigned to the respective group name (Figure 2.5, lines 33-37). Line 37 assigns isotropic shell property, where `1e6` is a Young's modulus, and `0.0` is Poisson's ratio. Between lines 41 and 56, the code sample also demonstrates a FISH function that adjusts each node's local axis system to point in the direction of the cylinder's center. This enables the structure shell to apply commands to exert pressure on shell elements in the model's center. The y-direction points upward, while the z-direction is perpendicular to the cylinder's center. Line 59 sets the nodes' rotation and speed fixed to allow the balancing of the balls inside the sleeve.

```
32 structure shell import from-geometry 'Default' element-type dkt-cst
33 structure node group 'middle' range position-z [-freeRegion] [freeRegion]
34 structure node group 'top' range position-z [freeRegion] [freeRegion+1]
35 structure node group 'bot' range position-z [-freeRegion-1] [-freeRegion]
36 structure shell group 'middle' range position-z [-freeRegion] [freeRegion]
37 structure shell property isotropic (1e6, 0.0) thick 0.25 density 930.0
38
39 model cycle 0
40
41 fish define setLocalSystem
42     loop foreach local s struct.node.list()
43         local p = struct.node.pos(s)
44         local nid = struct.node.id.component(s)
45         local mvec = vector(0,0,comp.z(p))
46         zdir = math.unit(p-mvec)
47         ydir = vector(0,0,1)
48         command
49             structure node system-local z [zdir] y [ydir] ...
&50                       range component-id [nid]
51         endcommand
52     endloop
53     command
54       structure node fix system-local
55     endcommand
56 end
57 @setLocalSystem
58 structure damp local
59 structure node fix velocity rotation
```

Figure 2.5. Shell generation.

In Figure 2.6, line 61, the "`wall-structure create`" command creates the sleeve. In this instance, wall facets are made for each shell face and slaved to the corresponding shell nodes' vertex velocities and positions. Between lines 64 and 71, the "wall generate" command creates the top and bottom platens with dimensions equal to 1.3 times the cylinder radius. In line 73, the "`wall resolution`" command sets the wall resolution mode to full. PFC balls and wall facets typically make contact. All contacts between facets and balls will be defined and contribute to the force-displacement response of the balls if the resolution mode is set to none. However, if the wall is idealized as a perfect surface without facets, this may not result in as many contacts as when the wall facet sizes are close to the ball sizes. So, in lines 73 and 74, the "`cutoff-angle`" is used to regulate the smoothness of the wall, and the "`full`" scheme is used to mitigate this situation.

```
61  wall-structure create
62  [rad2 = rad*1.3]
63
64  wall generate name 'platenTop' polygon ([-rad2],[-rad2],[halfLen]) ...
&65                                        ([rad2],[-rad2],[halfLen])  ...
&66                                        ([rad2],[rad2],[halfLen])   ...
&67                                        ([-rad2],[rad2],[halfLen])
68  wall generate name 'platenBottom' polygon ([-rad2],[-rad2],[-halfLen]) ...
&69                                           ([rad2],[-rad2],[-halfLen])  ...
&70                                           ([rad2],[rad2],[-halfLen])   ...
&71                                           ([-rad2],[rad2],[-halfLen])
72
73  wall resolution full
74  wall attribute cutoff-angle 20
```

Figure 2.6: Top and bottom platens generation.

2.3.2 Balls generation and contact modelling

Generating PFC balls and clumps with or without predetermined beginning overlap is possible. In Figure 2.7, line 77, a parallelepiped volume is created by using the "`ball distribute`" command to generate a cloud of balls with arbitrary overlaps. After a porosity of 0.3, a radius ranging between 0.05-0.10, and the distribution medium of the cylinder were defined. In lines 81-82, the "`contact cmat`" command is then used to provide the contact model behavior after assigning the ball attributes. The linear parallel bond ("linearpbond") contact model was used for this case. This contact method requires the user to specify the effective modulus – "`emod`", and normal-to-shear stiffness ratio – "`kratio`" (Itasca, 2021a). All these actions enable the balls to reconfigure themselves. In line 85, the "`model cycle`" command uses the "`calm`" keyword to expel all kinetic energy from the system periodically. In that way, the elements will be free to move for 1000 steps with the resets of linear and angular velocities to null every 100 steps.

```
76  [ballemod = 1.0e8]
77  ball distribute box [-rad] [rad] [-rad] [rad] [-halfLen] [halfLen] ...
&78                     porosity 0.3 radius 0.05 0.1 ...
&79                     range cylinder end-1 (0,0,[-halfLen]) end-2 (0,0,[height]) ...
&80                           radius [rad*0.95]
81  ball attribute density 2600 damp 0.8
82  contact cmat default model linearpbond method deformability ...
&83                           emod [ballemod] kratio 1.0
84
85  model cycle 1000 calm 100
86
87  contact cmat default model linearpbond method deformability ...
&88                           emod [ballemod] kratio 1.0 property fric 0.3
89  contact cmat apply
90
91  model solve ratio-average 1e-8
92
93  contact method bond gap 1.0e-2 range contact type 'ball-ball'
94  contact property lin_mode 1 lin_force 0 0 0 pb_ten 2e5 pb_coh 2e6
95  contact method pb_deformability emod [ballemod] kratio 1.0 ...
&96                range contact type 'ball-ball'
97
98  model calm
99  ball fix velocity spin
100 model cycle 2
101 ball free velocity spin
102
103 structure node free velocity rotation range group 'middle'
```

Figure 2.7: Ball generation and contact modelling.

In Figure 2.7, lines 87-89, the friction was introduced. Once the Contact Model Assignment Table has been modified, all contacts are updated due to the "`contact cmat`" command being issued. In line 91, the system of balls is cycled to a small average ratio with the help of some friction to drain extra energy. In line 93, the "`contact method bond`" command is used to install parallel bonds at all ball-ball contacts with gaps that are less than or equal to `1.0e-2` to create the bonded particle model. A "`contact method`" is a collection of operations used to manipulate various contact attributes when carrying out a task on contacts. After the bonds have been installed, the parallel bond strength is selected, and the linear parallel bond contact model's normal force computation is set to an incremental model.

Following the installation of the bonds, the parallel bond strength is determined, and the normal force calculation of the linear portion of the linear parallel bond contact model is set to the incremental model by specifying `lin_mode = 1`. Doing so allows one to swiftly create a bonded particle model without considering internal pressure. Correspondingly, the parallel bond component's stiffnesses are determined. After a few cycles, all contact forces are eliminated by eliminating all velocities and setting the ball's velocity and spin. As a result, there are no internal stresses, and the bonded particle model is in equilibrium. The shell nodes in the middle of the sleeve are finally released to prepare for the triaxial testing.

The experimental setup for the triaxial tests needs to be built after the established stress-free BPM. For now, a FISH function that uses the platens to determine the stress and strain needs to be prepared. Figure 2.8, lines 105-110, defines the inputs needed for the function.

```
105 [platenTop = wall.find('platenTop')]
106 [platenBottom = wall.find('platenBottom')]
107 [failureStress = 0]
108 [currentStress = 0]
109 [failureStrain = 0]
110 [area = math.pi()*rad^2.0]
111
112 fish define stress
113     local topForce = math.abs(comp.z(wall.force.contact(platenTop)))
114     local botForce = math.abs(comp.z(wall.force.contact(platenBottom)))
115     currentStress = 0.5*(topForce+botForce)/area
116     stress = currentStress
117     strain = (height - (comp.z(wall.pos(platenTop)) - ...
&118                 comp.z(wall.pos(platenBottom))))/height * 100
119     if failureStress <= currentStress
120         failureStress = currentStress
121         failureStrain = strain
122     endif
123 end
```

Figure 2.8: "Stress" function.

Here, the average stress on the platens is determined using the forces acting in the z-direction. The specimen height is used to compute the strain as well. The variables "`failureStress`" and "`failureStrain`" measure maximum stress and strain. The FISH function shown in Figure 2.9 below uses this information to stop cycling.

```
125 fish define halt
126     halt = 0
127     if currentStress < failureStress * 0.85
128         halt = 1
129     endif
130 end
```

Figure 2.9: "Halt" function.

Cycling will stop if the present stress level reaches 85% of the maximal stress. Later, the "`model solve fish-halt`" command, followed by a function name, can stop cycling in a FISH function. When "`model solve fish-halt halt`" command is used, every cycle uses a "`halt`" FISH function to check whether cycling should continue. Cycling continues if the function returns false; otherwise, cycling ends.

The "`rampUp`" FISH function show in Figure 2.10 is utilized to enhance the specimen's confining stress gradually. This function's main objective is progressively raising the confining stress on the platens and sleeves. It is simple to apply confining stress using the "`structure shell apply`" command (line 142) because the local system for the shells has already been defined previously. Additionally, to exert tension from both platens, the "wall servo" command uses the built-in wall servo mechanism. The "`wall servo`" command allows users to use a servo-mechanism to regulate the translational velocity of specific wallso apply or maintain a specified force (Itasca, 2021c).

```
fish define rampUp(beginIn,ending,increment)
    command
        ball attribute displacement (0,0,0)
        structure node initialize displacement (0,0,0)
    endcommand
    begin = beginIn
    loop while (math.abs(begin) < math.abs(ending))
        begin = begin + increment
        command
            ;apply the confining stress
            structure shell apply [begin] range group 'middle'
            ;apply the same confining stress on the platens
            wall servo force (0,0,[begin*area]) activate true ...
&               range name 'platenTop'
            wall servo force (0,0,[-begin*area]) activate true ...
&               range name 'platenBottom'
            model cycle 200
            model calm
        endcommand
    endloop
    command
        model cycle 1000
        wall servo activate false
        wall attribute velocity (0,0,0) range name 'platenTop'
        wall attribute velocity (0,0,0) range name 'platenBottom'
    endcommand
end

[platenVel = 0.000003]

model save 'beforeApplication'
```

Figure 2.10: "Ramp Up" function.

Figure 2.11 shows an example of how to perform a triaxial test with an isotropic confining stress of 1e4.

```
model restore 'beforeApplication'
@rampUp(0,-1e4,-1e3)
model save 'to1e4'
wall attribute velocity-z [-platenVel] range name 'platenTop'
wall attribute velocity-z [platenVel] range name 'platenBottom'
ball attribute displacement (0,0,0)
structure node initialize displacement (0,0,0)
fish history @stress
fish history @strain
model solve fish-halt halt
model save 'triaxial1e4'
[io.out(string(failureStress) + 'Pa ')]
[io.out('at' + string(failureStrain) + '% strain')]
```

Figure 2.11: Triaxial test simulation with 1e4 stress.

Chapter 3 – Methodology

3.1 Software

The computer software used in this research is PFC3D (or Particle Flow Code: 3 dimensions), developed by Itasca. Several DEM studies used the same software (de Bono et al., 2015; Jiang et al., 2013; Wu et al., 2021). PFC3D already has a built-in linear parallel bond contact model that can imitate the cohesive properties of cement. This helps to model cement-treated soil by establishing an elastic interaction between particles (Itasca, 2021a). It also allows force and moment transfer between particles, as was suggested previously by Yang et al. (2019).

3.2 Model design

For this research, the dimensions of specimens for triaxial test simulations are 38 mm x 76 mm. As discussed before, the documentation of PFC3D software has a related triaxial test simulation example, and its model design is shown in Figure 3.1. However, in that case, the loading pistons are modeled as rigid walls that push against the specimen to apply axial load. The membrane edges are given zero degrees of freedom (DOF), meaning they cannot move in any direction. This leads the membrane to extend beyond the walls and causes pressure build-up as the specimen volume is decreased, as shown in Figure 3.2.

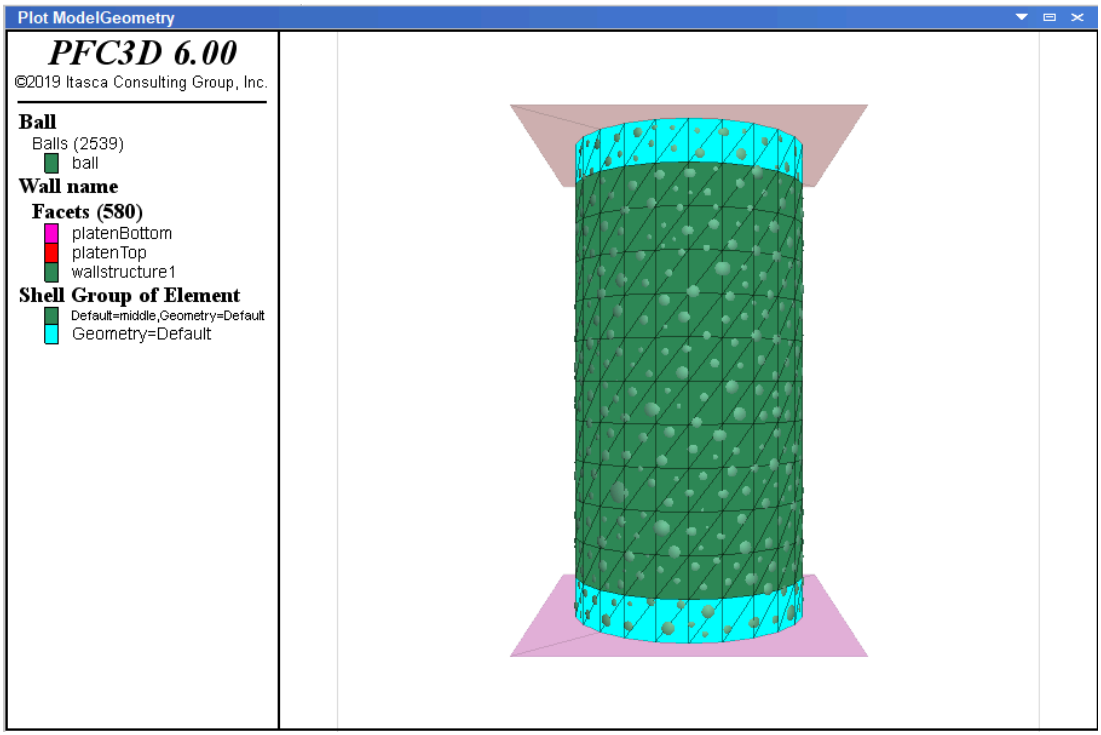


Figure 3.1: Model geometry from the documentation.

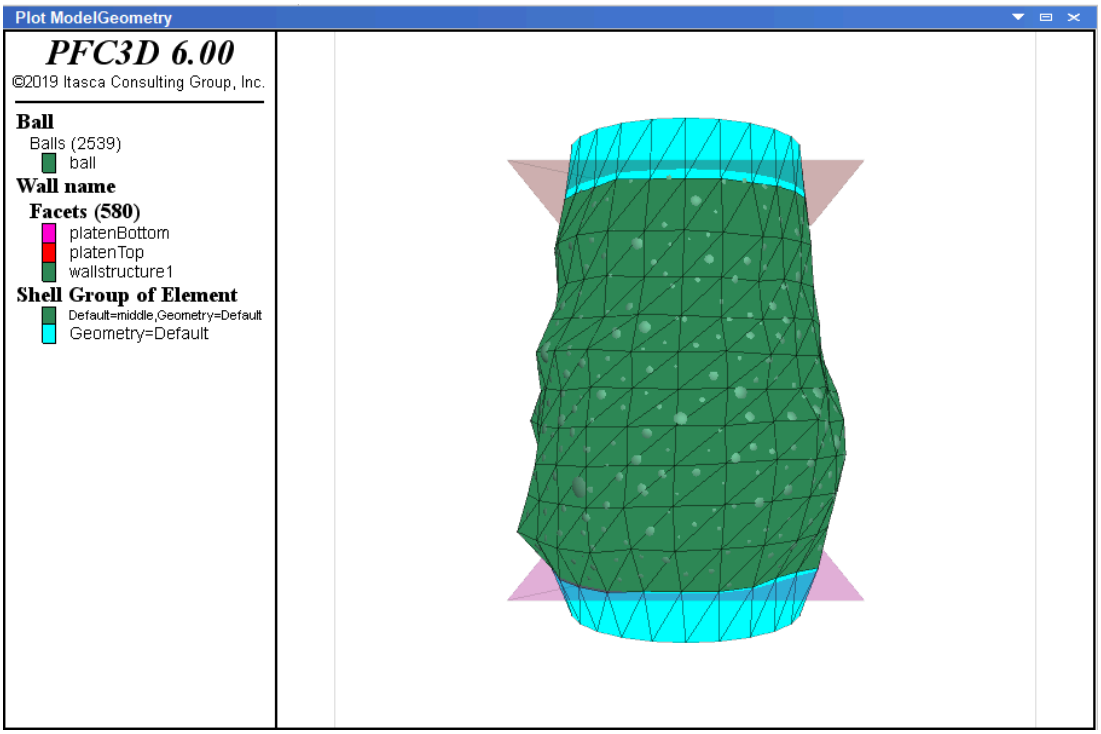


Figure 3.2: Model geometry from the documentation, after the test.

The loading pistons were modeled as cylindrical rigid bodies instead of simple rigid walls to address this issue. This adjustment better represents real triaxial test conditions, where pistons are typically cylindrical, and load is applied uniformly across the specimen's cross-section. The membrane length was extended to fully enclose the pistons, ensuring that ball particles would not spill out of the specimen. Moreover, in this modified design, the pistons and the membrane sections in contact with them move together as the triaxial test starts. This prevents the membrane from stretching beyond the specimen and eliminates the unintended pressure build-up caused by volume loss at the ends. By fully enclosing the pistons within the membrane, this approach allows for more uniform axial compression while maintaining proper confinement. This design change reduces stress concentrations near the ends and improves the accuracy of the simulated test, making the results more comparable to the actual physical triaxial experiments. Figures 3.3 show the modified model design.

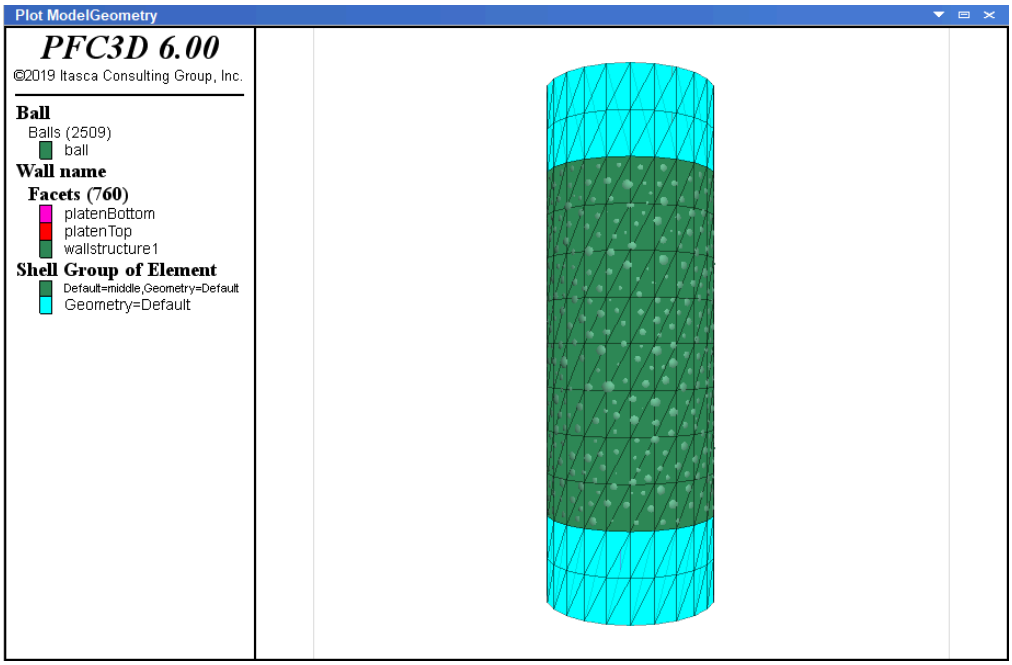


Figure 3.3: Modified model geometry.

The size of sand particles was magnified by 4 times to reduce the computational time of DEM simulation. The resultant model generated 2509 ball particles and 760 facet elements for a membrane, as seen in Figure 3.3.

As for the contact between particles, a linear parallel bond model was applied for interactions between balls with initial normal stiffness defined by the optimization algorithm. A

linear contact model for ball and wall connections with 10 GPa normal stiffness was chosen. After particle generation, additional parameters for ball-ball interactions were introduced, such as effective modulus, tensile strength, cohesion, normal stiffness, shear stiffness, friction angle, and friction coefficient. These variable properties were defined by the optimization algorithm Optuna.

Before simulation the test and applying confining pressure, the specimen should achieve equilibrium by releasing any extra kinetic energy stored within particles. This step is required as depending on the initial generated balls' position, they can overlap or have voids between them due to being randomly generated in a specified space. The initial contact forces of the generated triaxial test simulation can be seen in Figure 3.4. The mentioned initial state is an equilibrium state the specimen reached with the tolerance of 1e-7 average ratio. Between particles and structures, a total of 11943 contacts were established. Figure 3.5 shows the displacement of each particle and shell (sleeve) before the test.

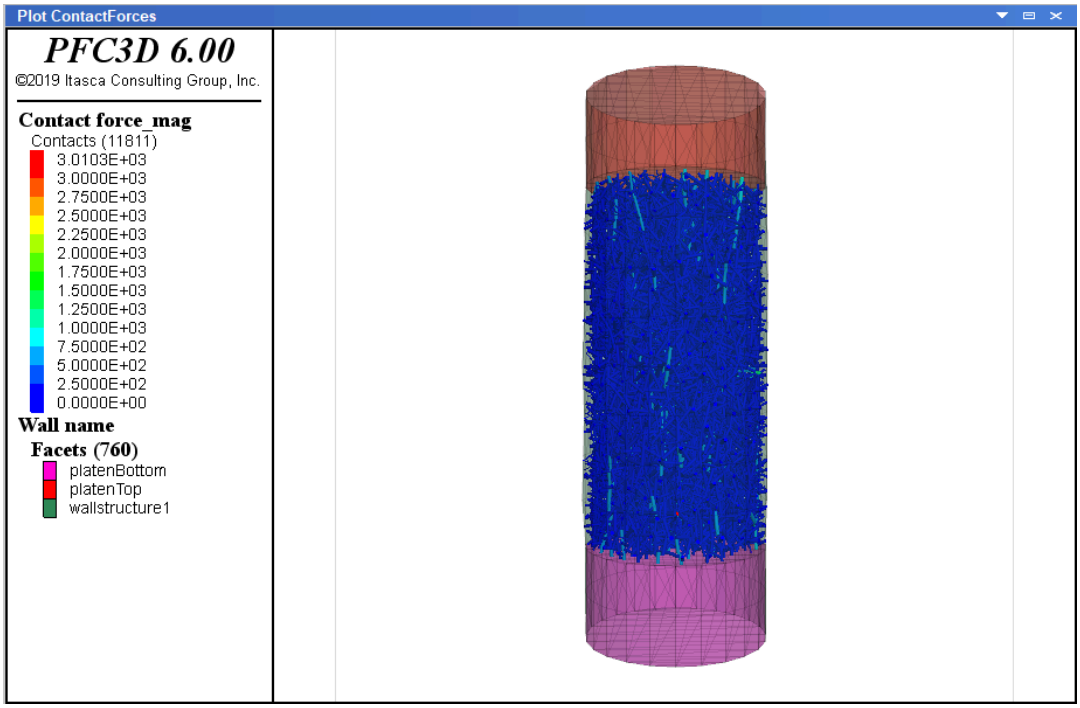


Figure 3.4: Contact forces before the test.

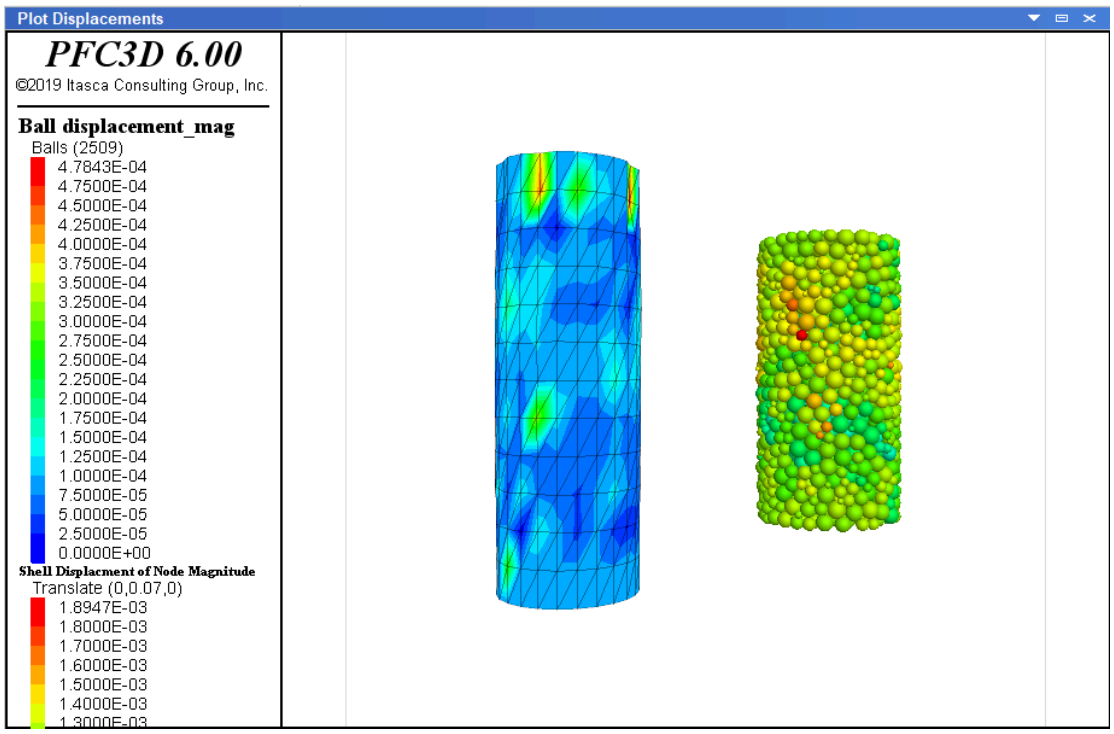


Figure 3.5: Displacements before the test.

3.3 Linear Parallel Bond Model

The linear parallel bond model describes the behavior of two types of interfaces. Those are linear elastic (friction) and bonded (force and moment) interfaces, as shown in Figure 3.6. Linear elastic interface behaves similarly to a linear model, allowing for slip within a Coulomb shear force limit and not resisting relative rotation. On the other hand, a parallel bond interface works alongside the first when bonded. In its bonded state, it resists relative rotation and behaves linearly elastic until it reaches its strength limit, breaking the bond and becoming unbonded. In its unbonded state, this interface doesn't bear any load. The unbonded linear parallel bond model aligns with the characteristics of the linear model.

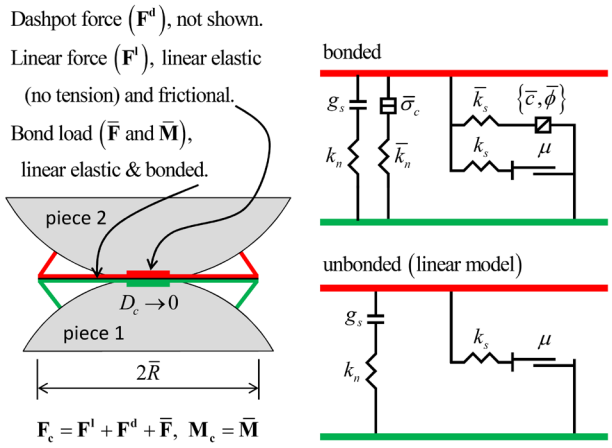


Figure 3.6: Behavior and rheological components of the linear parallel bond model with inactive dashpots (Itasca, 2021a).

A linear parallel bond model contact remains active when it is bonded or when the surface gap is equal to or less than zero. Inactive contacts are excluded from the force-displacement law. As depicted in the linear formulation figure, the surface gap indicates this condition. When the reference gap is zero, the theoretical surfaces of the first interface align with the actual surfaces. The theoretical surfaces of the second interface are illustrated in Figure 3.7.

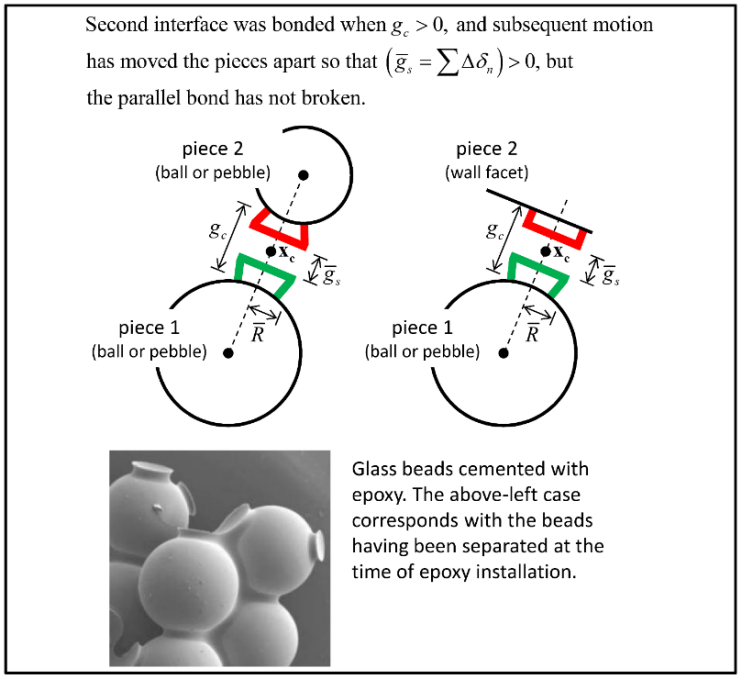


Figure 3.7: Finite-size notional surfaces (green and red) and parallel-bond surface gaps at parallel-bonded ball-ball (left) and ball-facet (right) contacts (Itasca, 2021a).

The force-displacement law for the linear parallel bond model governs the contact force and moment update. It involves components such as linear force, dashpot force, parallel-bond force, and moment. The parallel-bond force and moment are resolved into normal, shear, twisting, and bending components. Creating a parallel bond establishes an interface between surfaces, with the bond providing elastic interaction, removed upon bond breakage. The law includes steps for updating bond cross-sectional properties, normal, shear, twist, and bend increments, and enforcing strength limits. If strength limits are exceeded, the bond breaks in tension or shear, triggering a callback event. A diagram in Figure 3.8 illustrates the failure envelope for the parallel bond.

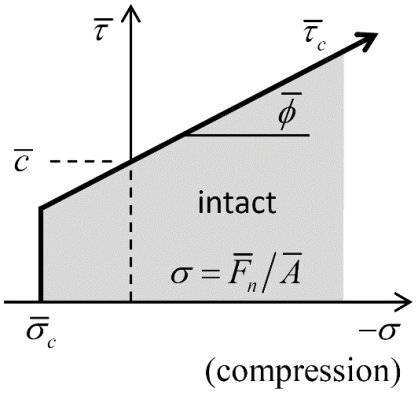


Figure 3.8: Failure envelope for the parallel bond (Itasca, 2021a).

The model uses 4 energy partitions:

- E_k – strain energy: stored in the linear springs;
- E_μ – slip energy: defined as the total energy dissipated by frictional slip;
- E_β – dashpot energy: defined as the total energy dissipated by the dashpots;
- $\bar{E}_k$ – bond strain energy: stored in the parallel-bond springs.

The strain, slip, and dashpot energy values are refreshed similarly to those in the linear model. The slip energy remains at zero until the bond rupture occurs. The bond strain energy is updated according to Equation (3.1):

$$\bar{E}_k = \frac{1}{2}\left(\frac{\bar{F}_n^2}{\bar{k}_n \bar{A}} + \frac{\|\bar{F}_s\|^2}{\bar{k}_s \bar{A}} + \frac{\bar{M}_t^2}{\bar{k}_s \bar{J}} + \frac{\|\bar{M}_b\|^2}{\bar{k}_n \bar{I}}\right) \tag{3.1}$$

Where:

- $\bar{k}_n$ – normal stiffness in a parallel-bond group;
- $\bar{k}_s$ – shear stiffness in a parallel-bond group;
- $\bar{A}$ – cross-sectional area;
- $\bar{I}$ – moment of inertia of the parallel bond cross-section;
- $\bar{J}$ – polar moment of inertia of the parallel bond cross-section;
- $\bar{F}$ – parallel-bond force (contact plane coordinate system: $-\bar{F}_n, \bar{F}_{ss}, \bar{F}_{st}$)
- $\bar{M}$ – parallel-bond moment (contact plane coordinate system: $-\bar{M}_t, \bar{M}_{bs}, \bar{M}_{bt}$)

3.4 Particle size distribution

When it comes to the simulation of the triaxial test of the soil with DEM, a particle size distribution is one of the critical parameters that must be considered in the coding. If no particle size distribution were specified, the particles' gradation curve in PFC3D by default would be the same, as shown in Figure 3.9 below.

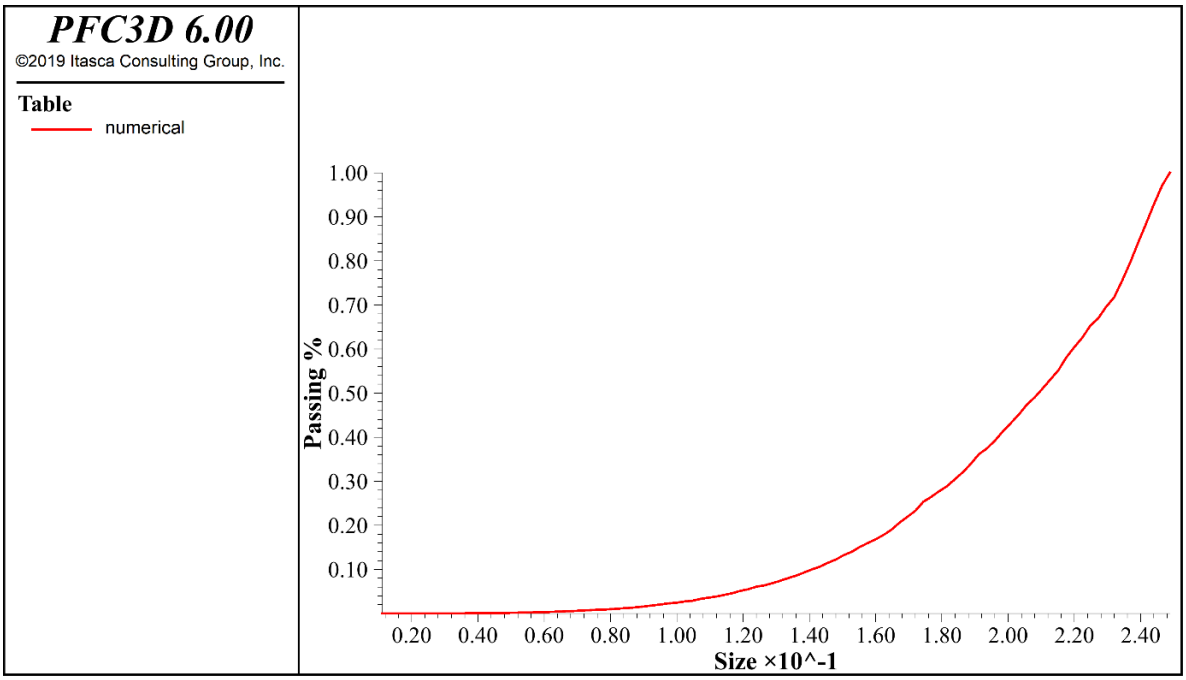


Figure 3.9: Default particle size distribution curve in PFC3D.

The default grain size distribution may not be the same as the one to be simulated. With a few tweaks, achieving the desired particle size distribution curve in PFC3D is possible. In this case, the paper by Ocheme et al. (2022) was used for reference. In his article, several triaxial tests were performed on CSA-treated sand samples with different confining pressures. From their paper, the particle distribution data and curve can be found in Table 3.1 and Figure 3.10.

Table 3.1: Particle distribution data from the paper; (Ocheme et al., 2022).

Sieve size, mm	Total % Passing
9.50	100.00%
4.75	100.00%
2.36	99.98%
1.18	98.65%
0.60	1.82%
0.30	0.20%
0.15	0.04%
0.00	0.00%

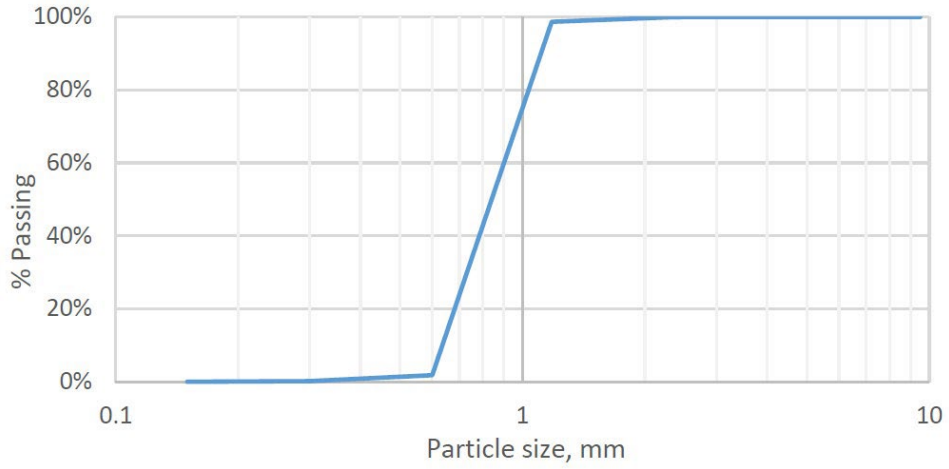


Figure 3.10: Particle distribution curve from the reference paper (Ocheme et al., 2022).

Before using the actual particle size distribution for simulated particles, it should be magnified to reduce simulation time. To achieve a reasonable time required for the simulation, this study applied a scaling factor of 4. In Figure 3.11, a blue line represents the experimental data from the reference paper. A red line is a simulated ball's gradation curve. Even though it seems to be slightly shifted to the right side, the only requirement for this step is to have the same gradation curve trend.

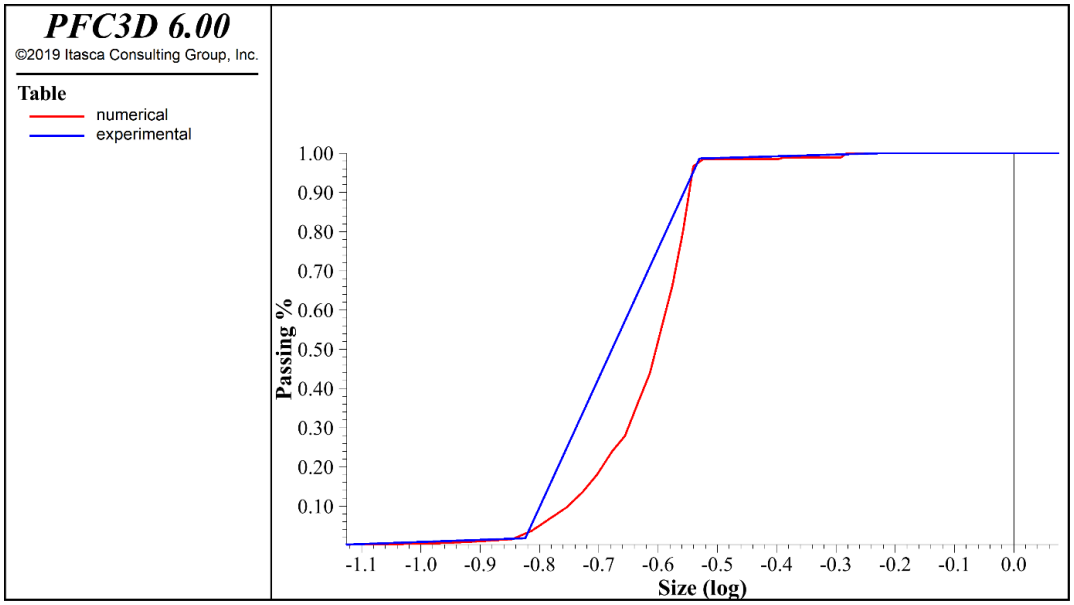


Figure 3.11. The adjusted particle distribution curve

3.5 Characterization optimization

3.5.1 Search space

Sand particle density was assumed to be 2650 kg/m^3 (Blake, 2008). Table 3.2 presents some parameters required for PFC3D simulation of the triaxial test with a linear parallel bond contact model. All listed properties directly affect the physical properties of simulated particles. The range of numbers was set after a throughout literature review from related academic papers and software documentation (de Bono et al., 2015; Feng et al., 2017; Itasca, 2021a; Jiang et al., 2011; Obermayr et al., 2013; Wang & Leung, 2008; Wu et al., 2021). New parameters were also introduced. R_{bond} is a ratio of bond gap formation distance to the smallest particle's diameter. This parameter determines the relative distance at which bonds should be formed before starting the

test. $N_{pressure}$ sets the number of steps required to reach target confining pressure. If more steps number is defined, the simulation will apply confining pressure slowly. And M_{scale} is a particle size magnification scale. Increasing particle size significantly reduces simulation time.

Table 3.2: Search space for triaxial test simulation parameters.

Symbol	Software Command	Parameter name	Value
$\overline{\sigma_c}$	contact property pb_ten	parallel-bond tensile strength	1e3 – 1e8
$\bar{c}$	contact property pb_coh	parallel-bond cohesion	1e3 – 1e8
$\overline{k_n}$	contact property pb_kn	parallel-bond normal stiffness	1e3 – 1e8
$\overline{k_s}$	contact property pb_ks	parallel-bond shear stiffness	1e3 – 1e8
$\bar{\phi}$	contact property pb_fa	parallel-bond friction angle	20.0 – 45.0
$\bar{E}$	contact method pb_deformability emod	bond effective modulus	1e7 – 1e10
μ	contact property fric	friction coefficient	0.0 – 1.2
k_n	contact property kn	linear normal stiffness	1e5 – 1e7
k_s	contact property ks	linear shear stiffness	1e4 – 1e7
E	contact method deformability emod	linear effective modulus	1e7 – 1e10
ζ	ball attribute damp	local damping factor	0.1 – 0.9
n	ball distribute porosity	porosity	0.30 – 0.50
R_{bond}	---	bond gap ratio	10 – 1000
$N_{pressure}$	---	pressure divisions	5 – 50
M_{scale}	---	magnification scale	2 – 5

3.5.2 Optimization framework – Optuna

These parameters are required for further calibration to characterize the micro-properties of CSA-cement treated soil. For that purpose, adopting a parameter optimization technique is critical to developing an accurate simulation model. Akiba et al. (2019) introduced Optuna – an open-source hyperparameter optimization framework. It has gained significant attention due to its flexibility, efficiency, and automatic search-space pruning capabilities. Optuna is also the best option for problems requiring structured selection and tuning (Shekhar et al., 2022). While commonly used in machine learning applications, Optuna also effectively optimizes parameters in various engineering and scientific simulations, including this study.

By default, Optuna employs a Bayesian optimization approach with a Tree-structured Parzen Estimator (TPE) as the primary optimization algorithm (Optuna Contributors, n.d.). Unlike grid and random search methods, which explore the search space inefficiently, TPE adaptively selects parameter values based on previous evaluations. This leads to faster convergence and reduced computational cost. Moreover, Optuna supports pruning techniques that terminate underperforming trials early, significantly accelerating the search process (Akiba et al., 2019).

3.5.3 Objective function

In this study, the objective function that Optuna is minimizing, represents the deviation in stress-strain graphs between the simulated behavior of the triaxial test and the actual behavior observed in experimental data. To achieve a more accurate representation of simulation differences, the deviation was formulated as a weighted combination of two distinct error metrics: Euclidean distance and root mean square error (RMSE), as defined in Equation (3.2).

$$Deviation = w_1 \cdot d_{scaled} + w_2 \cdot RMSE \tag{3.2}$$

where, w_1 and w_2 are weighting factors.

First term (d_{scaled}) is the scaled Euclidean distance between the failure points (peak stress states) of the numerical and experimental curves:

$$d_{scaled} = \sqrt{\left(\frac{x_{exp} - x_{num}}{R_\varepsilon}\right)^2 + \left(\frac{y_{exp} - y_{num}}{R_\sigma}\right)^2} \tag{3.3}$$

where, (x_{exp}, y_{exp}) and (x_{num}, y_{num}) are the failure points (strain, stress) in the experimental and numerical datasets, respectively. R_ε and R_σ are the ranges of strain and stress as shown in Equations (3.4 - 3.5):

$$R_\varepsilon = max(\varepsilon_{num}, \varepsilon_{exp}) - min(\varepsilon_{num}, \varepsilon_{exp}) \tag{3.4}$$

$$R_\sigma = max(\sigma_{num}, \sigma_{exp}) - min(\sigma_{num}, \sigma_{exp}) \tag{3.5}$$

As for the second term in deviation calculation, $RMSE$ is a root mean square error. It measures the overall difference between numerical and experimental stress values (Equation 3.6):

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\sigma_{num,i} - \sigma_{exp,i}\right)^2} \tag{3.6}$$

By incorporating Euclidean distance in deviation calculation, the optimization process specifically targets the accuracy of failure points. This ensures that the peak stress and strain values align well between the simulation and experiment. This is particularly important in capturing the material's strength and failure characteristics. Meanwhile, RMSE accounts for discrepancies throughout the entire stress-strain curve, ensuring that the overall response is well-represented. The weighted combination of these metrics enables the optimization to refine both the global fit of the stress-strain behavior and the precise matching of failure points, leading to a more robust calibration of DEM parameters. The parameters listed in Table 3.2 were tuned to minimize this deviation, and optimization process continued iteratively until sufficient results were presented.

Chapter 4 – Numerical Simulations Results and Discussions

4.1 Trial test simulation

Preliminary simulation results for the triaxial test with 1 MPa confining pressure can be seen in Figures 4.1-4.3. The simulation interruption criteria were set as a 20% strain rate toamtch the maximum strain rate recorded by the reference experimental data Additional conditions for test interuptions were also introduced to avoid potential errors. Simulation would stop if: the ratio of peak stress to current stress is greater than 2.5; and the ratio of current strain to failure strain is greater than 4. Figure 4.1 shows a failure by bulging, with ductile behavior on the stress-strain graph in Figure 4.4.

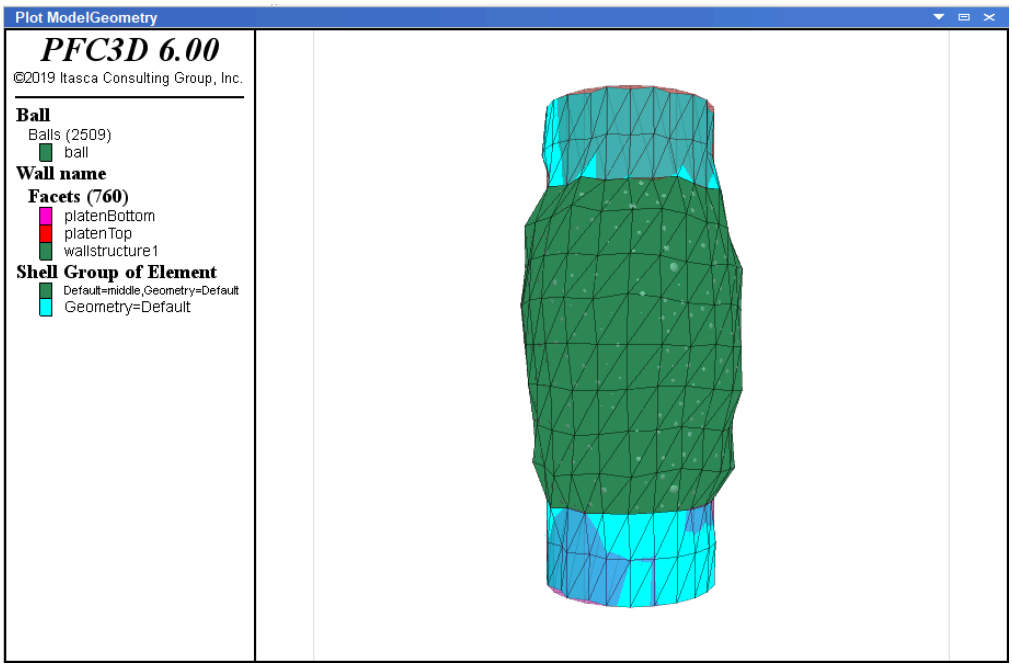


Figure 4.1: Model geometry after a trial test.

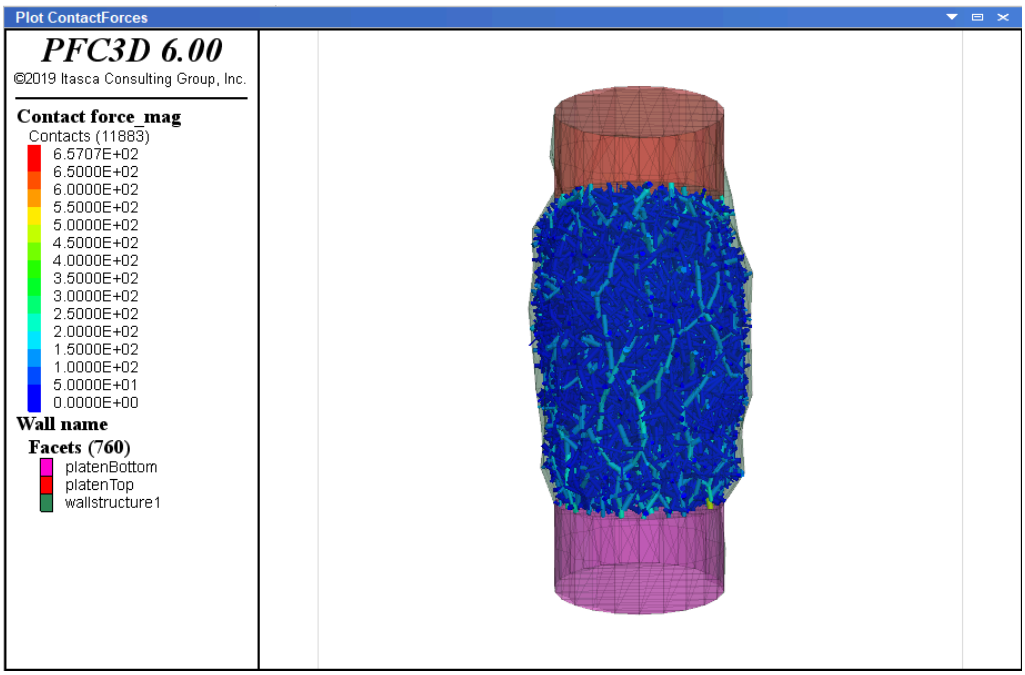


Figure 4.2: Contact forces after a trial test.

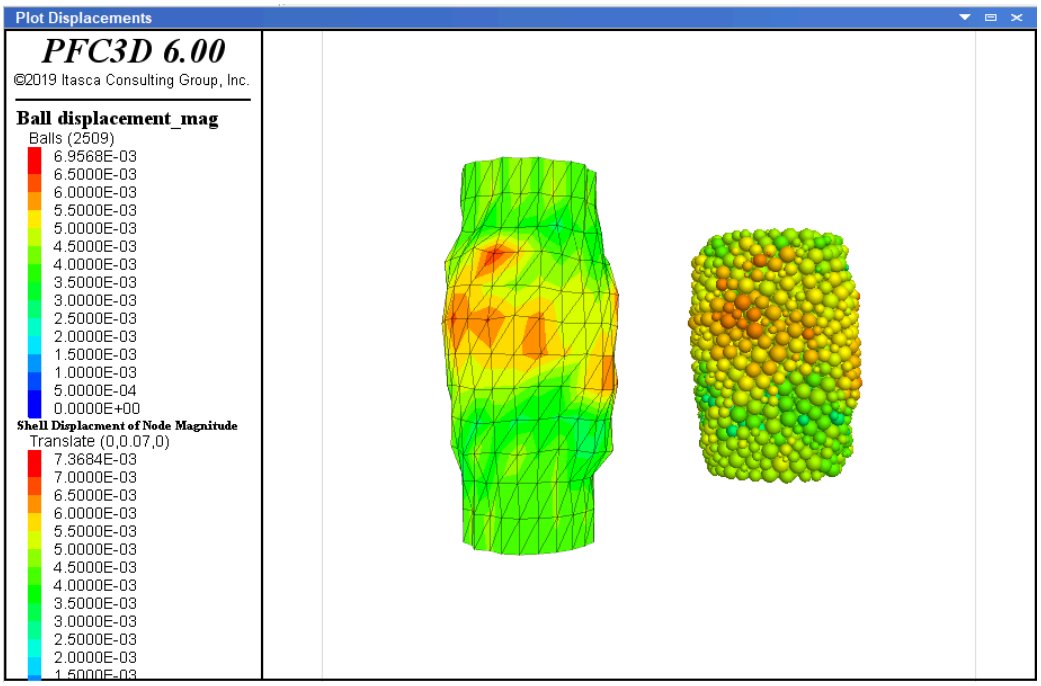


Figure 4.3: Displacements after a trial test.

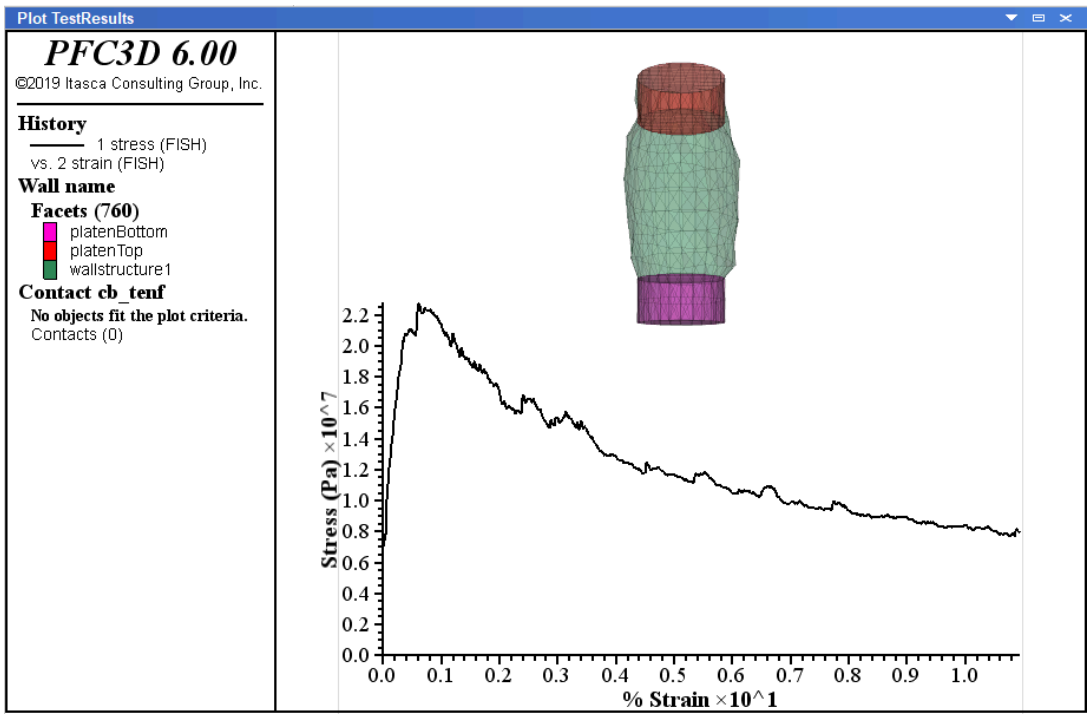


Figure 4.4: Stress vs. strain graph of the test.

4.2 DEM parameters sensitivity

Several parameters were chosen based on bond properties, linear interparticle interactions, and particle attributes. DEM simulations were conducted by systematically varying multiple parameters to evaluate their influence on the stress-strain response of CSA cemented sand. During this process, one parameter was systematically varied at a time. Each figure in this section presents multiple stress-strain plots corresponding to different parameters. Selected values for each parameter were distributed logarithmically or linearly depending on the parameter's expected influence range. The following parameters were investigated: $\overline{\sigma_c}$, $\bar{c}$, $\overline{k_n}$, $\overline{k_s}$, $\bar{\phi}$, $\bar{E}$, μ, k_n, k_s, E, ζ, n, R_{bond}, $N_{pressure}$, and M_{scale}. The results of each parameter analysis are shown in Figures 4.5 - 4.19. All virtual experiments were performed under a confining pressure of 0.5 MPa. The results highlight distinct trends across different mechanical properties, with some parameters significantly influencing the material behavior while others show negligible or no effect.

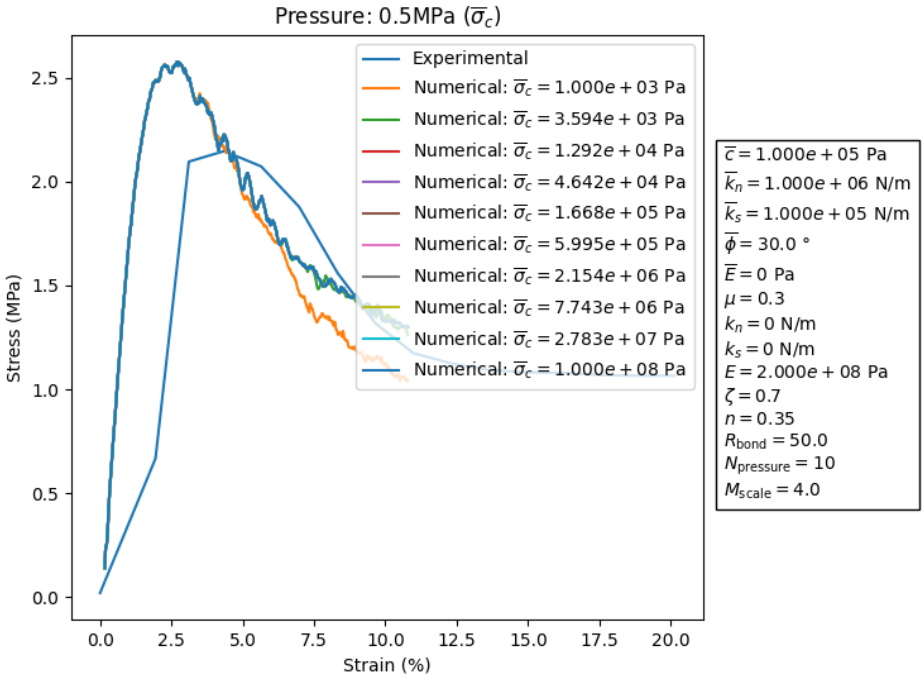


Figure 4.5: Stress-strain response: Parallel-bond tensile strength.

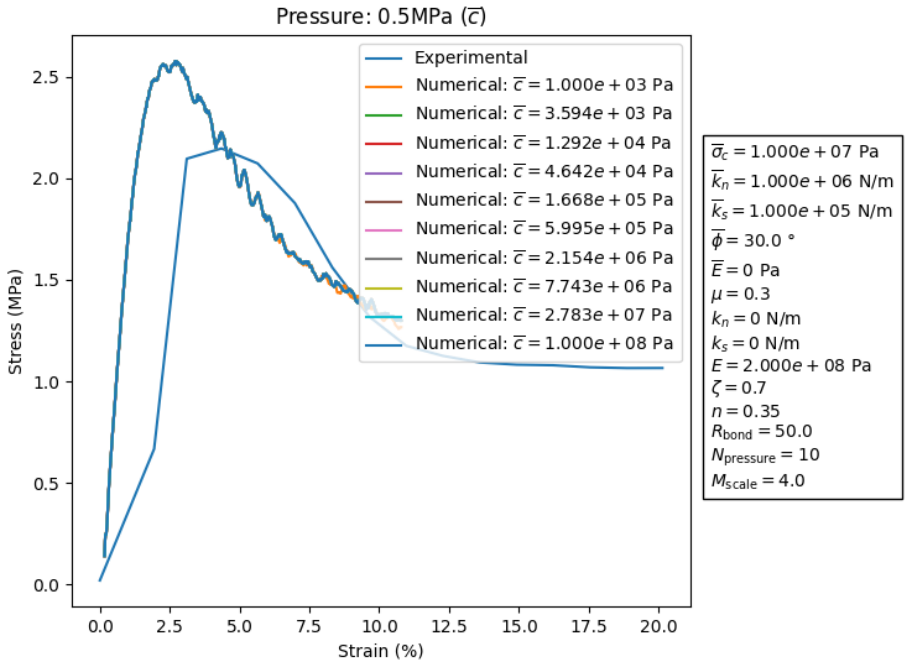


Figure 4.6: Stress-strain response: Parallel-bond cohesion strength.

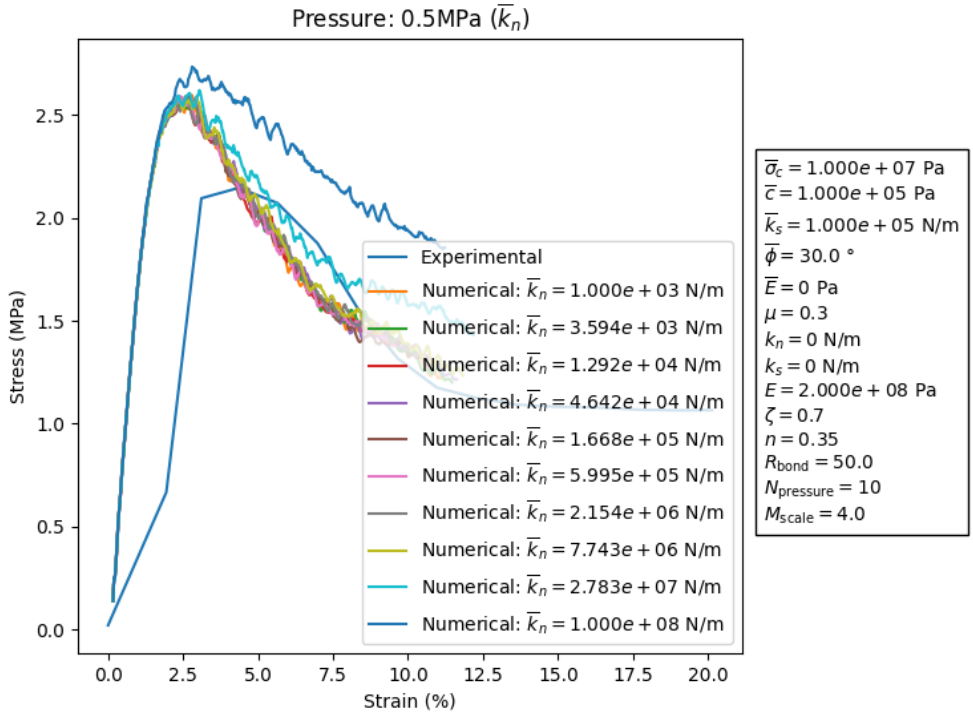


Figure 4.7: Stress-strain response: Parallel-bond normal stiffness.

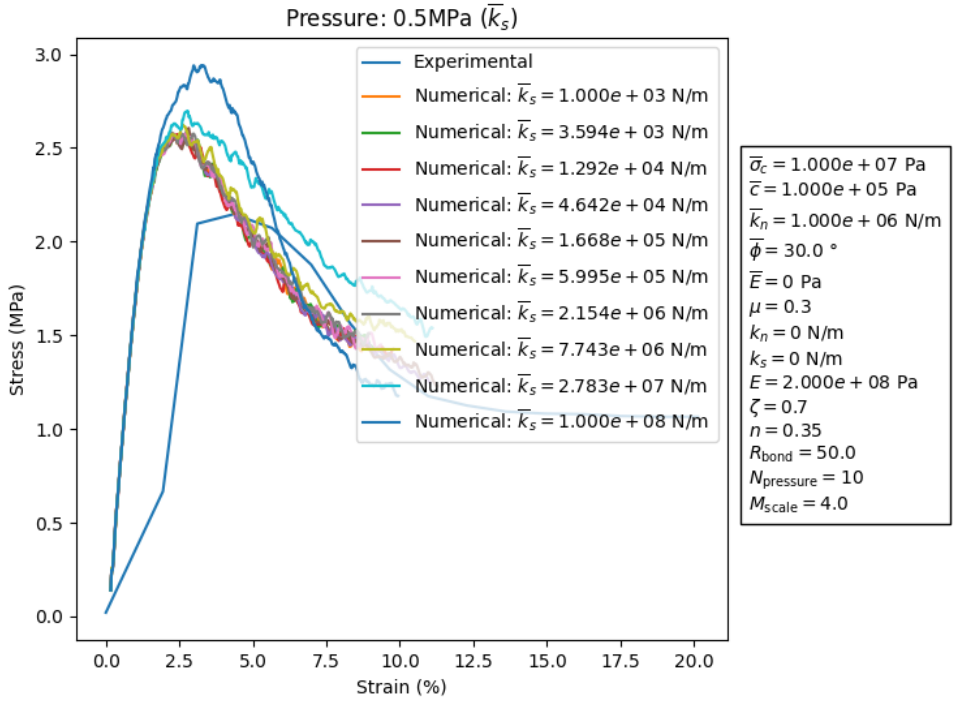


Figure 4.8: Stress-strain response: Parallel-bond shear stiffness.

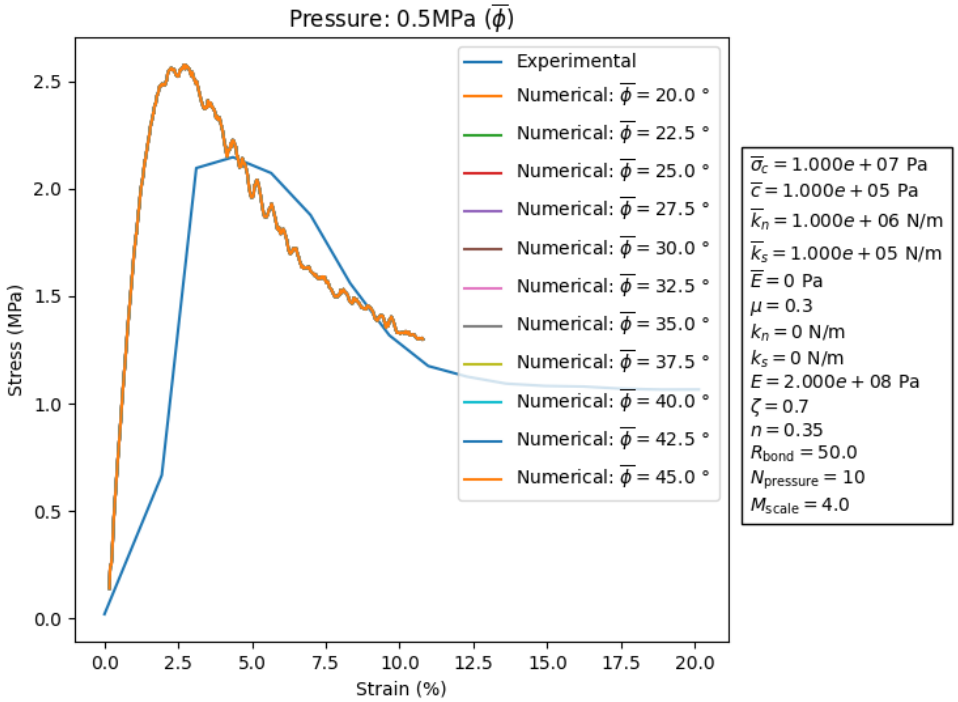


Figure 4.9: Stress-strain response: Parallel-bond friction angle.

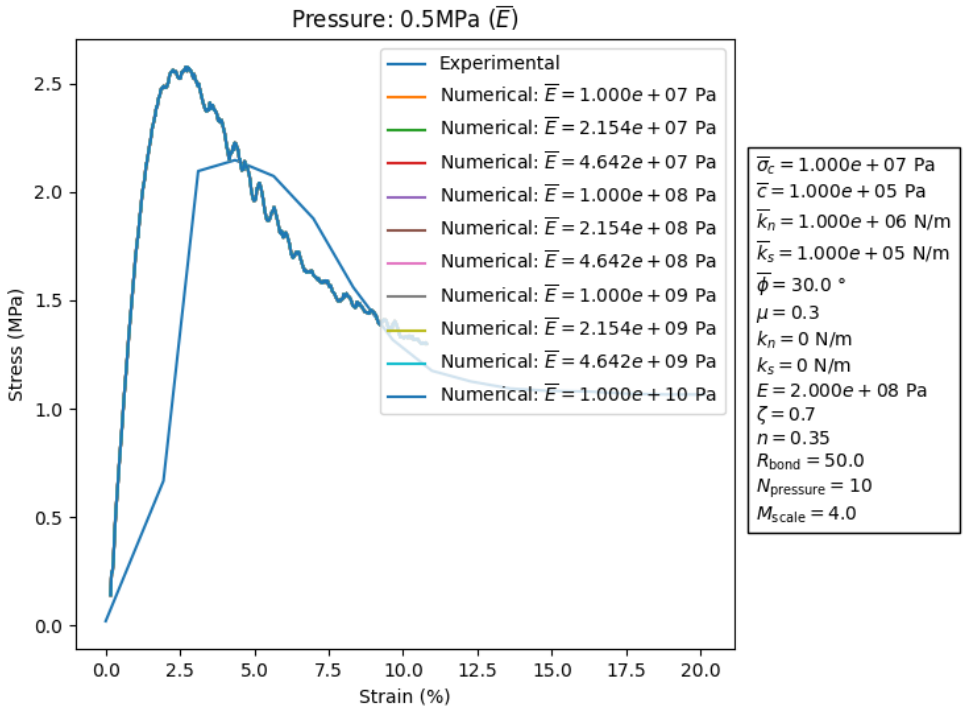


Figure 4.10: Stress-strain response: Bond elastic modulus.

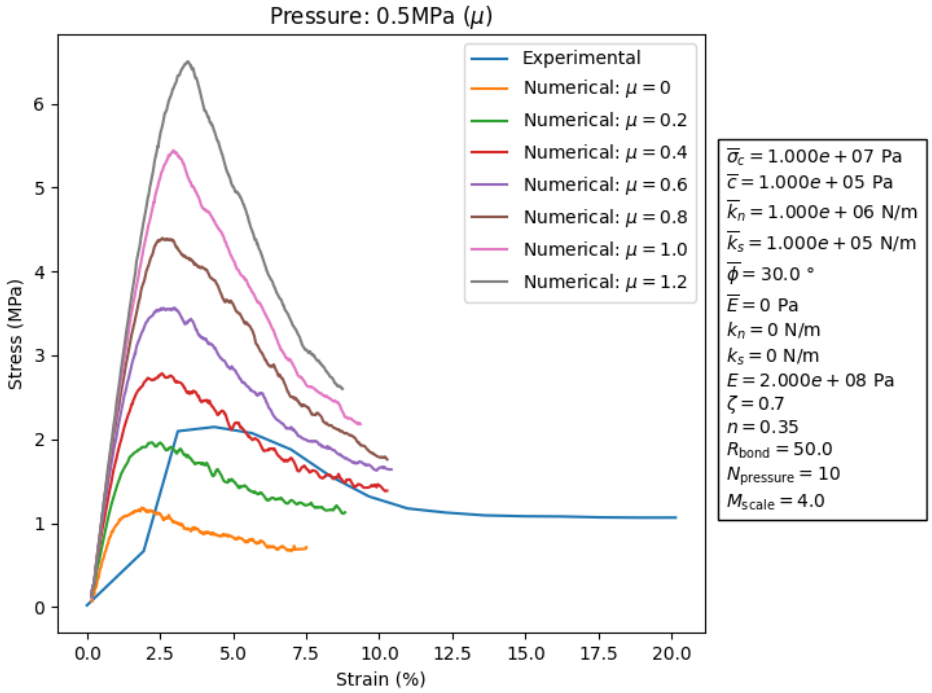


Figure 4.11: Stress-strain response: Friction coefficient.

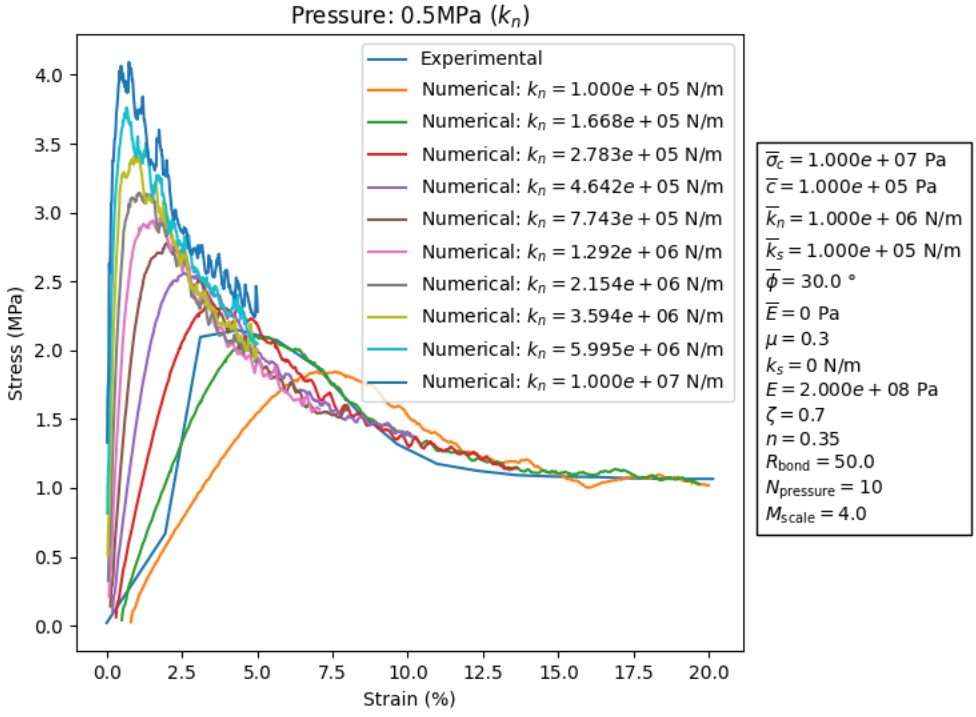


Figure 4.12: Stress-strain response: Linear normal stiffness.

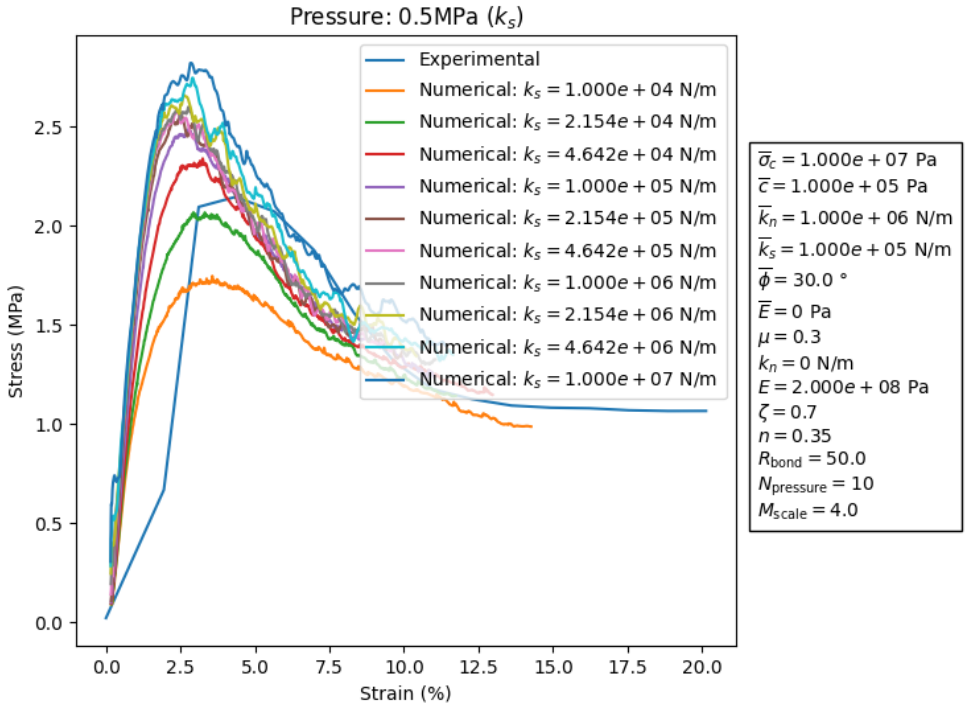


Figure 4.13: Stress-strain response: Linear shear stiffness.

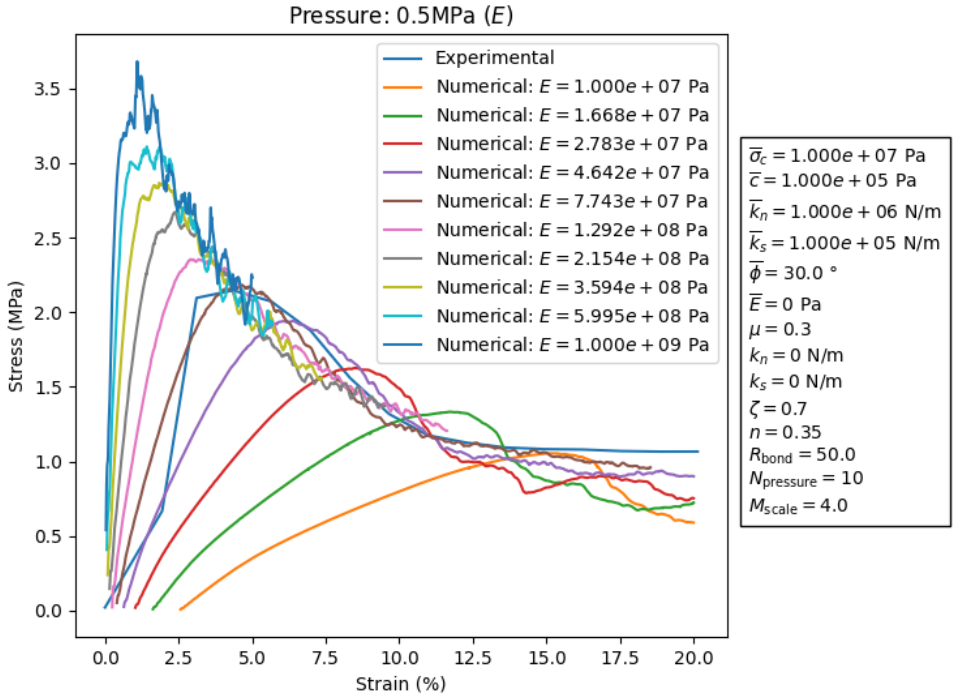


Figure 4.14: Stress-strain response: Linear elastic modulus.

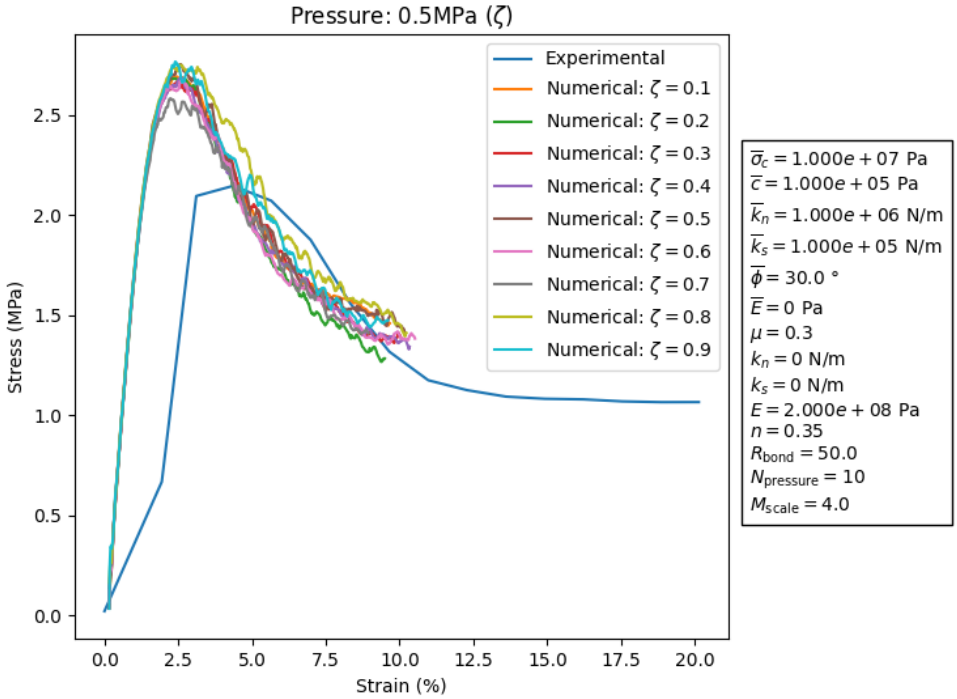


Figure 4.15: Stress-strain response: Damping factor.

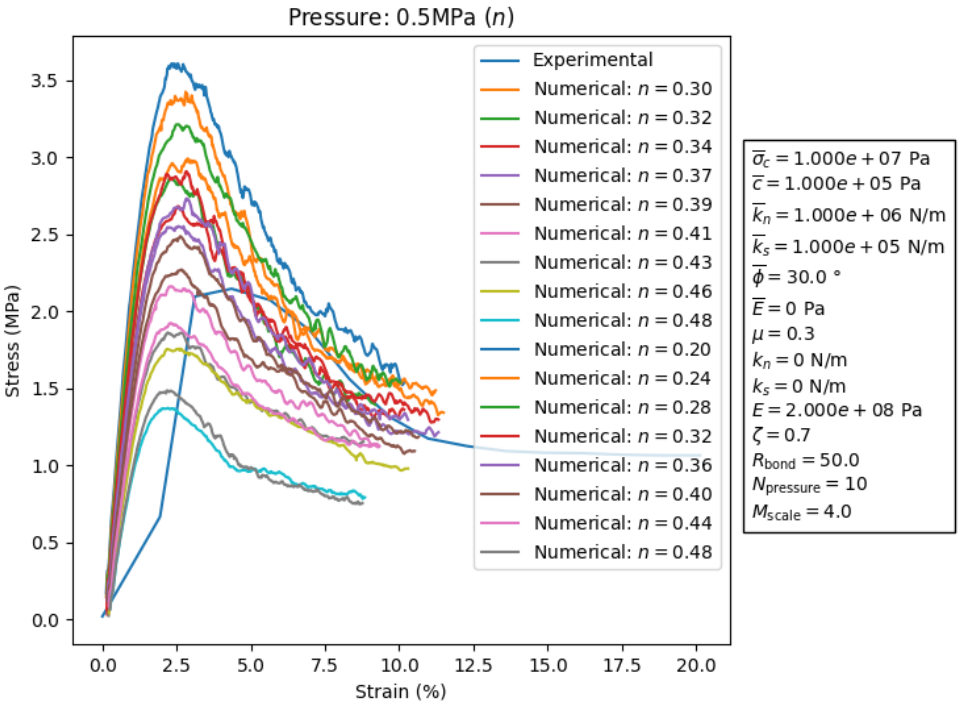


Figure 4.16: Stress-strain response: Porosity.

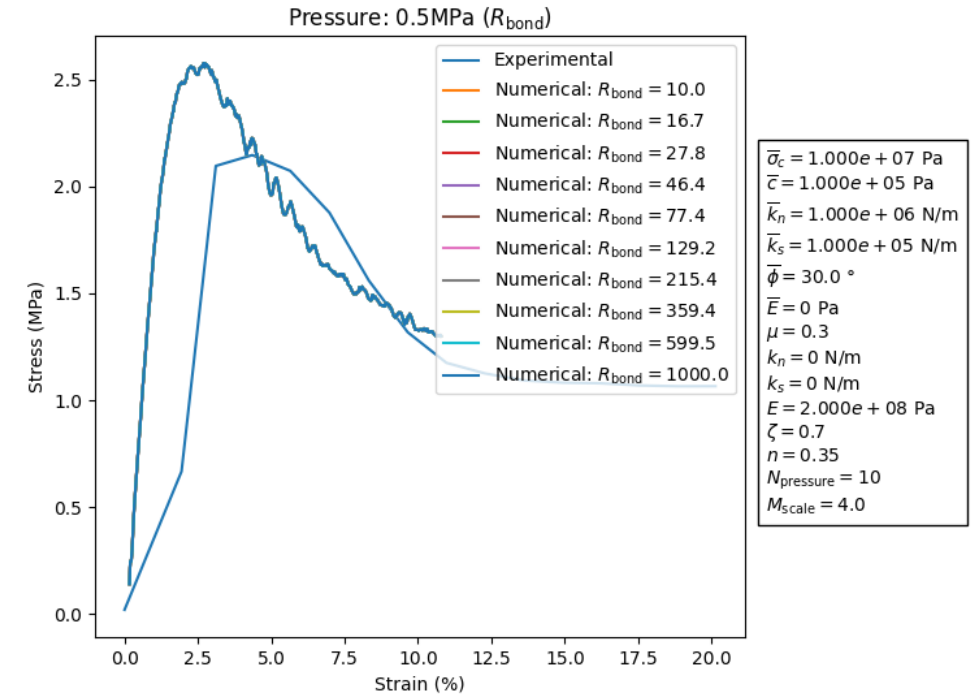


Figure 4.17: Stress-strain response: Bond gap ratio.

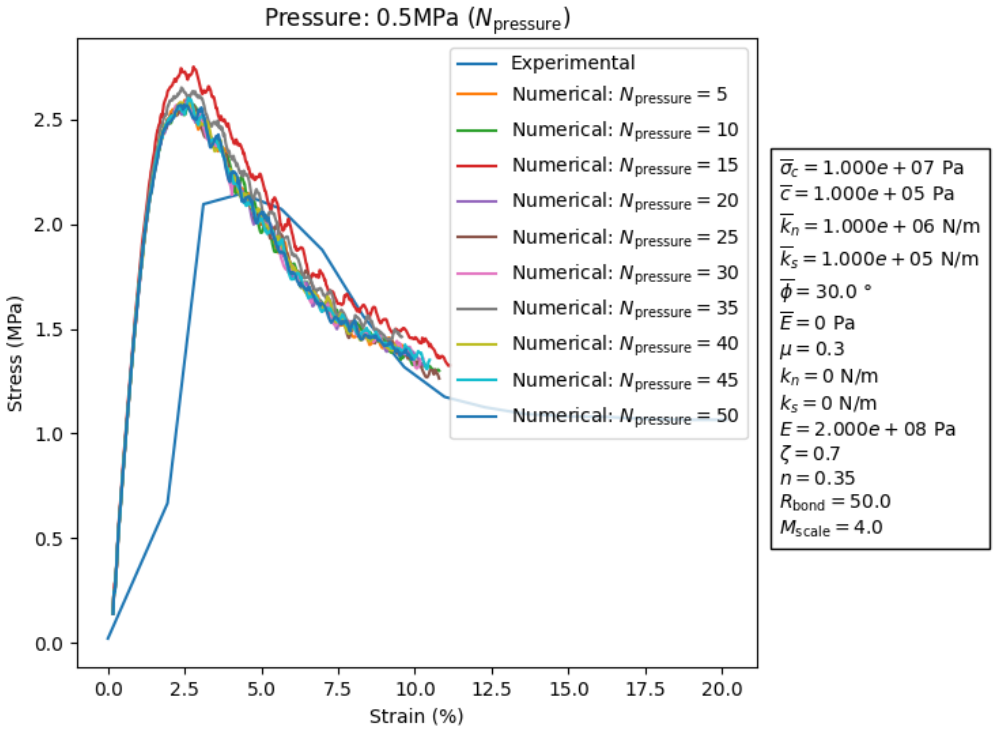


Figure 4.18: Stress-strain response: Pressure divisions.

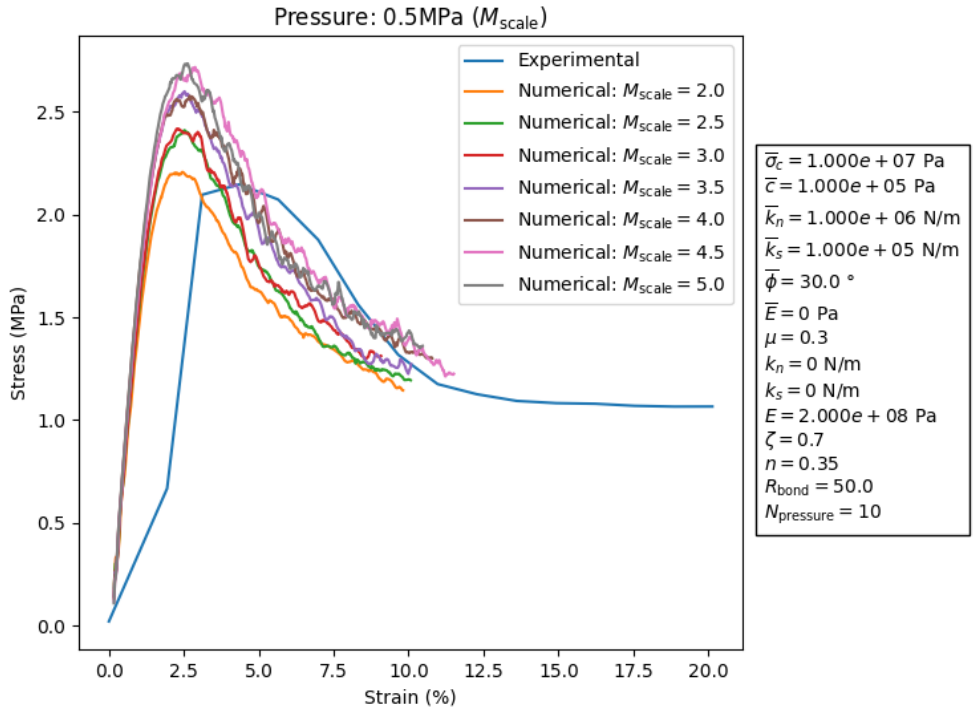


Figure 4.19: Stress-strain response: Particle magnification scale.

Linear elastic modulus (E), linear normal stiffness (k_n), and linear shear stiffness (k_s) play a significant role in governing the stiffness and peak strength of the material. Increasing E, k_n, or k_s leads to a steeper initial slope in the elastic region and higher peak stress, while lower values produce a more gradual pre-peak response. Post-peak behavior also varies considerably. High values of the mentioned parameters results in brittle failure and a sharp stress drop, whereas lower values lead to a more ductile response.

Friction coefficient (μ) and porosity (n) significantly impact the material strength. Both parameters strongly influence the peak stress, with higher μ and lower n leading to greater resistance to deformation. Post-peak residual strength is also more pronounced with increasing μ, while higher n reduces overall strength and accelerates stress drop, emphasizing its crucial role in governing material failure.

Parallel-bond normal ($\overline{k_n}$) and shear ($\overline{k_s}$) stiffnesses exhibited no significant influence on the initial stiffness or peak stress of the material but had a noticeable effect on the post-peak response. The variations in bond stiffness parameters did not change the elastic region or peak

strength. However, they contributed to differences in the stress drop and failure mechanism beyond peak stress section.

In contrast, several parameters exhibit minimal or no influence on the stress-strain response. Parallel-bond tensile ($\bar{\sigma_c}$) and cohesion ($\bar{c}$) strengths had no discernible effect, except for minor differences after the peak stress. Damping factor (ζ) also demonstrate negligible effects, with only minor variations near peak stress and in the post-peak phase. Increasing pressure divisions ($N_{pressure}$) moderately extends simulation duration, but does not significantly alter material behavior apart from minor peak and post-peak differences. Parallel-bond friction angle ($\bar{\phi}$), bond elastic modulus ($\bar{E}$), and bond gap ratio (R_{bond}) show no impact on either the elastic or post-peak behavior.

While most studies assume particle magnification (M_{scale}) scale has no effect, the results of the research indicate a noticeable impact on peak stress. This suggests that particle size scaling may influence contact mechanics beyond prior expectations.

Overall, these findings underscore the importance of selecting appropriate stiffness parameters (E, k_n, and k_s), while recognizing the critical roles of μ and n in governing strength and failure mechanisms. On top of that, several bond-related parameters of contact model ($\bar{\sigma_c}$, $\bar{c}$, $\overline{k_n}$, and $\overline{k_s}$) would also be introduced to cover the mechanics of CSA cement in the simulated sand particles.

4.3 DEM characterization optimization

The optimization process involved running multiple DEM simulations, where Optuna systematically adjusted the selected parameters to minimize the deviation between experimental and numerical stress-strain responses. Default single-objective optimizer – Tree-structured Parzen Estimator (TPE) was employed to efficiently navigate the high-dimensional search space. TPE is a Bayesian optimization method that models the probability density of well-performing and poorly performing parameter sets separately. This allows the optimizaer to make informed sampling decisions. The results of Optuna with TPE optimizer in terms of deviation reduction (Section 3.5.3) over iterations is presented in Figure 4.20.

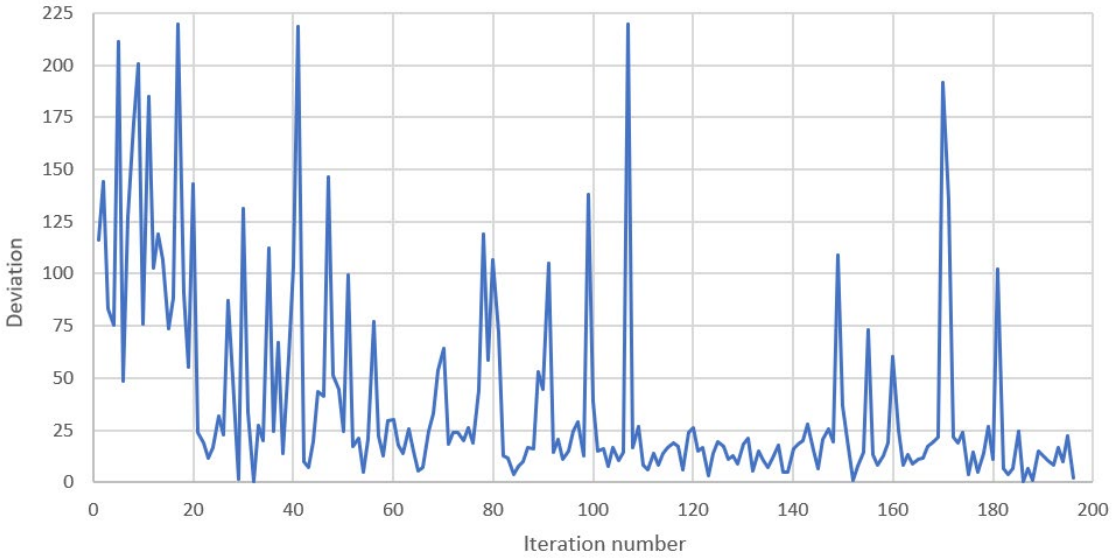


Figure 4.20: Optuna optimization progress: Deviation reduction over iterations

As shown in Figure 4.20, almost 200 iterations were performed. Initially, the deviation fluctuates widely, exceeding 200 in some cases, indicating a broad exploration of parameter space. As iterations progress, a downward trend can be observed. Deviations became smaller and more stable after around 100 simulations, suggesting convergence toward optimized parameter sets. However, occasional spikes in later iterations indicate continued exploration to escape local minima. The overall pattern suggests that the optimization effectively balances exploration and exploitation. It refines the parameter selection while maintaining adaptability to improve the characterization of CSA cemented sand behavior. Best-fit parameters for the reference experimental stress-strain plot of CSA cemented sand in triaxial test at 1.0 MPa confining pressure can be found in Figure 4.21.

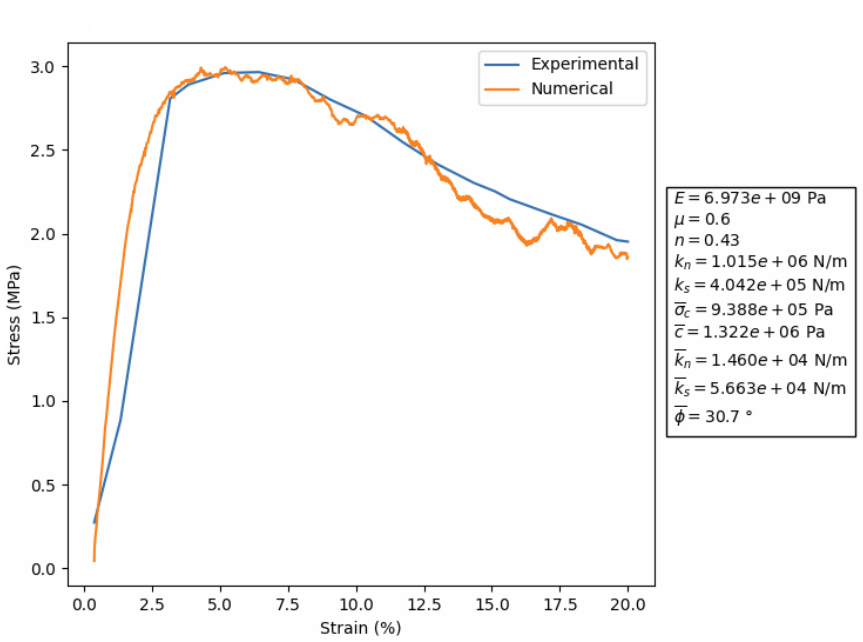


Figure 4.21: Best fit DEM simulation.

The blue curve represents the experimental data, while the orange curve corresponds to the numerical simulation. The numerical model captures the initial stiffness and peak strength well but slightly overestimates the strain-hardening behavior before the peak. After reaching peak stress, the numerical response shows more fluctuation than the experimental curve but follows a similar softening trend. The parameters are listed in Table.

Table 4.1: Optimized parameters.

Parameter	Symbol	Value	Unit
linear elastic modulus	E	$6.973 \cdot 10^9$	Pa
friction coefficient	μ	0.6	–
porosity	n	0.43	–
linear normal stiffness	k_n	$1.015 \cdot 10^6$	N/m
linear shear stiffness	k_s	$4.042 \cdot 10^5$	N/m
parallel-bond tensile strength	$\bar{\sigma}_c$	$9.388 \cdot 10^5$	Pa
parallel-bond cohesion strength	$\bar{c}$	$1.322 \cdot 10^6$	Pa
parallel-bond normal stiffness	$\bar{k}_n$	$1.460 \cdot 10^4$	N/m
parallel-bond shear stiffness	$\bar{k}_s$	$5.663 \cdot 10^4$	N/m
friction angle	$\bar{\phi}$	30.7	°

The displacement evolution of CSA cemented sand during triaxial compression was analyzed at different strain levels: 0%, 2%, 6%, 10%, 15%, and 20% (Figure 4.22). The numerical model effectively captures the deformation characteristics, highlighting localized strains and progressive failure patterns at high confining presure.

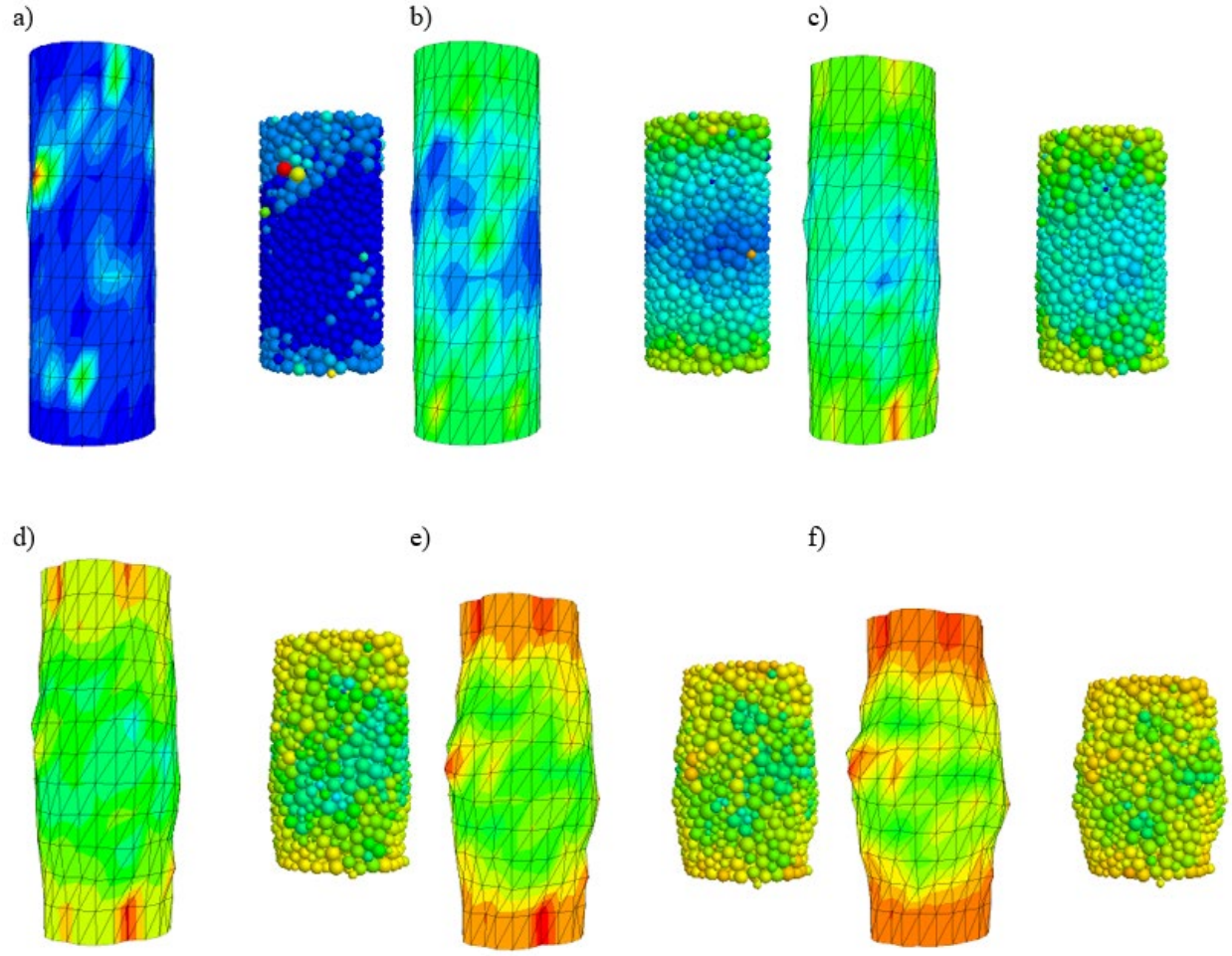


Figure 4.22: Displacement evolution of DEM simulated CSA cemented sand and membrane at different strain levels: (a) 0%, (b) 2%, (c) 6%, (d) 10%, (e) 15%, and (f) 20%.

At 0% strain, the sample remains undeformed, with minimal displacement variations. By 2% strain, slight deformations are observed, particularly at localized contact points between particles on top and bottom part of the specimen. As strain increases to 6%, the specimen already reached a failure state as shown in stress-strain plot (Figure 4.21). At 10% strain, visible barreling and particle rearrangement occur, leading to higher displacements in localized regions. By 15%, zones of high displacement extend across the sample, showing significant degradation of structural

integrity. At 20% strain, the failure mode is fully developed, characterized by large displacements and noticeable bulging, resembling experimentally observed behaviors in cemented sand under high strains.

Figure 4.23 illustrates the evolution of contact forces in DEM simulated CSA cemented sand. It reveals a progressive redistribution and weakening of force chains as strain increases. Initially, at low strain levels, a well-connected force network is observed, indicating a stable load-bearing structure. As deformation progresses, force chains become more heterogeneous, with localized stress concentrations forming around weaker regions. Beyond 10% strain, noticeable fragmentation of the network occurs, corresponding to strain localization and the development of shear bands. At higher strain levels, force chains significantly degrade, reflecting the material's loss of cohesion and transition into a more granular-like behavior, characteristic of post-peak failure.

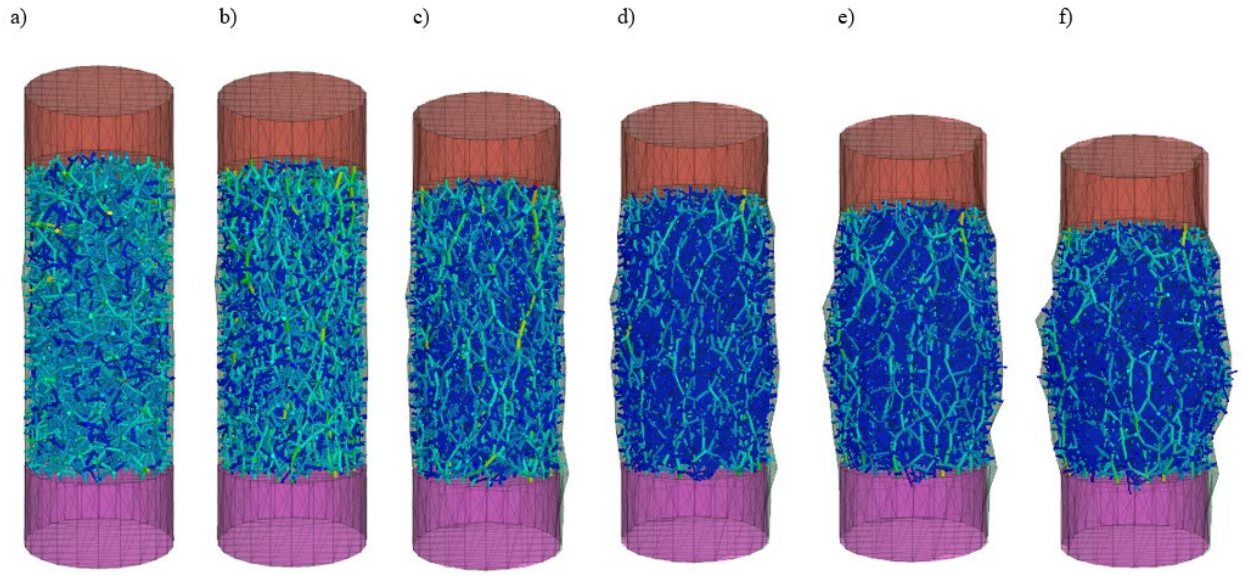


Figure 4.23: Evolution of contact force networks in DEM simulated CSA cemented sand during triaxial compression at different strain levels: (a) 0%, (b) 2%, (c) 6%, (d) 10%, (e) 15%, and (f) 20%.

4.4 Post-optimization analysis

A parallel coordinates plot is a visualization technique used to represent high-dimensional data in a two-dimensional space. Unlike traditional Cartesian plots, where data points are mapped along x and y axes, parallel coordinates plots display multiple variables as vertical axes placed

parallel to each other. Each data point is represented as a connected line traversing across these axes, illustrating how its values vary across different parameters. This method is particularly useful in this optimization study, as it allows for the identification of relationships between DEM parameters and highlights patterns in high-dimensional datasets. The color of the lines can be used to encode the deviation, making it easier to discern optimal regions in the parameter space.

Figure 4.24 represents the results of an Optuna optimization process conducted for a material model at 1.0 MPa. The objective function used in the optimization is a deviation metric designed to quantify the difference between experimental and simulation stress-strain responses. This metric is defined as a combination of RMSA (root mean square error) and Euclidean distance. It serves as a measure of how well the simulation replicates the experimental data. Each line corresponds to a parameter set tested during the Optuna optimization process. Colors of each line indicates the deviation magnitude. Darker shades represent lower deviations and lighter shades represent higher deviations.

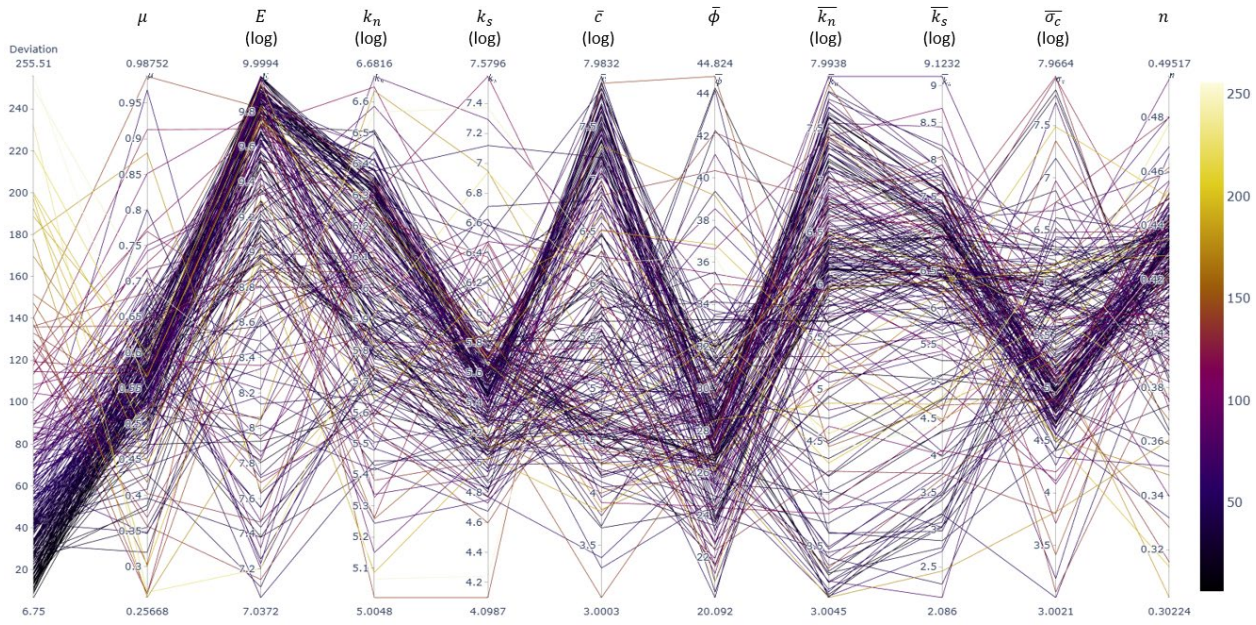


Figure 4.24: Parallel coordinates plot: Optuna optimization results.

The clustering of lower deviation cases (darker lines) in specific parameter ranges suggests that optimal values exist for achieving better agreement with experimental data. From Figure 4.24, it is evident that certain parameters, particularly μ, E, k_n, k_s, and n exhibit significant influence on the deviation, as indicated by dark line concentrations on specific ranges of their axes. This

suggests that their values play a crucial role in minimizing the error. And in contrast, impact of $\bar{\phi}$, $\bar{\sigma_c}$, $\bar{c}$, $\overline{k_n}$, and $\overline{k_s}$ parameters are not sufficient as dark lines are more dispersed across their respective axes, indicating weaker correlations with the deviation metric. This suggests that variations in these parameters do not significantly impact the deviation. And they have a weaker influence on the accuracy of the simulation results. Their broad spread implies that optimizing these parameters alone may not substantially improve the agreement between DEM simulations and experimental results. The same results was concluded in Section 4.2, but these parameters were introduced to mimic cement-like behavior.

Chapter 5 – Conclusion

5.1 Summary

This study focused on the Discrete Element Method (DEM) characterization of CSA cemented sand under drained triaxial conditions. To achieve an accurate representation of its mechanical behavior, a sensitivity analysis was conducted on 15 input parameters, identifying 10 as the most influential and replicating cement effects. These parameters were then optimized using Optuna's Tree-structured Parzen Estimator (TPE), aiming to minimize the deviation between numerical and experimental stress-strain responses. The optimization process successfully determined the best-fit parameters, capturing key mechanical properties such as stiffness, strength, failure mode. Displacement and contact force evolution in DEM simulations were analyzed to validate model behavior. Also, post-optimization parameter analysis confirmed the sensitivity study's findings. The results demonstrate that a data-driven approach, integrating machine learning-based optimization with DEM, enhances the accuracy and efficiency of cemented sand modeling. This methodology provides a foundation for further studies on cemented geomaterials using numerical simulations.

5.2 Future perspectives

The current study provides a robust framework for DEM calibration using machine learning-based optimization. The research conducted in this study holds significant implications for both academic and industry fields. Future research could expand on this work by:

- Incorporating additional experimental datasets to validate the model across a wider range of conditions.
- Investigating alternative contact models to improve the representation of cemented sand microstructures.
- Extending the optimization process to include multi-objective criteria, with several confining pressure levels.
- Exploring the impact of loading rate and confinement pressure variations to enhance predictive capabilities.

This study underscores the effectiveness of combining DEM with Optuna for geomechanical modeling, offering a data-driven approach to improving the accuracy and reliability of cemented sand simulations.

Chapter 6 – References

Ahmadi, H., & Sizkow, S. F. (2020). Numerical analysis of ground improvement effects on dynamic settlement of uniform sand using DEM. *SN Applied Sciences*, *2*(4), 1-9. https://doi.org/10.1007/S42452-020-2502-0/FIGURES/16

Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). Optuna: A Next-generation Hyperparameter Optimization Framework. https://github.com/pfnet/optuna/

Andavan, S., & Maneesh Kumar, B. (2020). Case study on soil stabilization by using bitumen emulsions – A review. *Materials Today: Proceedings*, *22*, 1200-1202. https://doi.org/10.1016/J.MATPR.2019.12.121

Binesh, S. M., Eslami-Feizabad, E., & Rahmani, R. (2018). Discrete Element Modeling of Drained Triaxial Test: Flexible and Rigid Lateral Boundaries. *International Journal of Civil Engineering*, *16*(10), 1463-1474. https://doi.org/10.1007/S40999-018-0293-0/METRICS

Blake, G. R. (2008). Particle density. In W. Chesworth (Ed.), *Encyclopedia of Soil Science* (pp. 504-505). Springer Netherlands. https://doi.org/10.1007/978-1-4020-3995-9_406

Cui, C., Wang, W., Jin, F., & Huang, D. (2020). Discrete-Element Modeling of Cemented Granular Material Using Mixed-Mode Cohesive Zone Model. *Journal of Materials in Civil Engineering*, *32*(4), 04020031-04020031. https://doi.org/10.1061/(ASCE)MT.1943-5533.0003069

de Bono, J., McDowell, G., & Wanatowski, D. (2015). Investigating the micro mechanics of cemented sand using DEM. *International Journal for Numerical and Analytical Methods in Geomechanics*, *39*(6), 655-675. https://doi.org/10.1002/NAG.2340

de Bono, J., McDowell, G. R., & Wanatowski, D. (2014). DEM of triaxial tests on crushable cemented sand. *Granular Matter*, *16*(4), 563-572. https://doi.org/10.1007/S10035-014-0502-8/FIGURES/9

Feng, K., Montoya, B. M., & Evans, T. M. (2017). Discrete element method simulations of bio-cemented sands. *Computers and Geotechnics*, *85*, 139-150. https://doi.org/10.1016/J.COMPGEO.2016.12.028

Itasca. (2019). *PFC*. https://www.itascacg.com/software/pfc

Itasca. (2021a). *Linear Parallel Bond Model*. https://docs.itascacg.com/pfc600/common/contactmodel/linearpbond/doc/manual/cmlinearpbond.html

Itasca. (2021b). *Sleeved Triaxial Test of a Bonded Material*. https://docs.itascacg.com/pfc600/pfc/wallsel/test3d/ExampleApplications/SleevedTriaxialTest/SleevedTriaxialTest.html?node5346

Itasca. (2021c). *wall servo command*. https://docs.itascacg.com/pfc600/pfc/pfcmodule/doc/manual/wall_manual/wall_commands/cmd_wall_servo.html#kwd:wall.servo.force

Jiang, M., Zhang, W., Sun, Y., & Utili, S. (2013). An investigation on loose cemented granular materials via DEM analyses. *Granular Matter*, *15*(1), 65-84. https://doi.org/10.1007/S10035-012-0382-8/METRICS

Jiang, M. J., Yan, H. B., Zhu, H. H., & Utili, S. (2011). Modeling shear behavior and strain localization in cemented sands by two-dimensional distinct element method analyses. *Computers and Geotechnics*, *38*(1), 14-29. https://doi.org/10.1016/J.COMPGEO.2010.09.001

Li, Z., Wang, Y. H., Ma, C. H., & Mok, C. M. B. (2017). Experimental characterization and 3D DEM simulation of bond breakages in artificially cemented sands with different bond strengths when subjected to triaxial shearing. *Acta Geotechnica*, *12*(5), 987-1002. https://doi.org/10.1007/s11440-017-0593-6

Lu, Y. (2010). *Reconstruction, characterization, modeling and visualization of inherent and induced digital sand microstructures* Georgia Institute of Technology]. http://hdl.handle.net/1853/37176

Mustafayeva, A., Bimykova, A., Olagunju, S. O., Kim, J., Satyanaga, A., & Moon, S.-W. (2023). Mechanical Properties and Microscopic Mechanism of Basic Oxygen Furnace (BOF) Slag-Treated Clay Subgrades. *Buildings*, *13*(12), 2962. https://doi.org/10.3390/buildings13122962

Mustafayeva, A., Moon, S.-W., Satyanaga, A., & Kim, J. (2024). Enhancing Mechanical Properties of Expansive Soil Through BOF Slag Stabilization: A Sustainable Alternative to Conventional Methods. *Minerals*, *14*(11), 1145. https://doi.org/10.3390/min14111145

Ng, T.-T., & Asce, M. (2004). Triaxial Test Simulations with Discrete Element Method and Hydrostatic Boundaries. *Journal of Engineering Mechanics*, *130*(10), 1188-1194. https://doi.org/10.1061/(ASCE)0733-9399(2004)130:10(1188)

Ning, Z., Khoubani, A., & Evans, T. M. (2017). Particulate modeling of cementation effects on small and large strain behaviors in granular material. *Granular Matter*, *19*(1), 1-19. https://doi.org/10.1007/S10035-016-0686-1/FIGURES/22

Obermayr, M., Dressler, K., Vrettos, C., & Eberhard, P. (2013). A bonded-particle model for cemented sand. *Computers and Geotechnics*, *49*, 299-313. https://doi.org/10.1016/J.COMPGEO.2012.09.001

Ocheme, J. I., Ruslan, K., Satyanaga, A., Kim, J., & Moon, S.-W. (2022). *Triaxial Test Behavior of CSA-treated Sand with High Confining Pressure* The 2022 World Congress on Advances in Civil, Environmental, & Materials Research, http://www.i-asem.org/publication_conf/acem22/2.%20GE/5.GE217_5/1.%20GE1169_7274F5.pdf

Optuna Contributors. (n.d.). *Efficient Optimization Algorithms — Optuna 4.2.1 documentation*. Retrieved 21/02/2025 from https://optuna.readthedocs.io/en/stable/tutorial/10_key_features/003_efficient_optimization_algorithms.html

Pu, Y., Sean, O. D., Nasser, H., Edward, K., & Narayanan, N. (2017). 3D DEM Simulations of Drained Triaxial Compression of Sand Strengthened Using Microbially Induced Carbonate Precipitation. *International Journal of Geomechanics*, *17*(6), 04016143-04016143. https://doi.org/10.1061/(ASCE)GM.1943-5622.0000848

Rakhimzhanova, A. K., Thornton, C., Minh, N. H., Fok, S. C., & Zhao, Y. (2019). Numerical simulations of triaxial compression tests of cemented sandstone. *Computers and Geotechnics*, *113*, 103068-103068. https://doi.org/10.1016/J.COMPGEO.2019.04.013

Rauf, A., Moon, S.-W., Lim, C.-K., Satyanaga, A., & Kim, J. (2024). Mechanical characteristics of CSA-treated sand reinforced with fiber under freeze-thaw cycles. *Case Studies in Construction Materials*, *21*, e03875. https://doi.org/10.1016/j.cscm.2024.e03875

Rauf, A., Moon, S.-W., Satyanaga, A., & Kim, J. (2025). Assessing Durability and Stability of Calcium Sulfoaluminate Cement-Stabilized Soils Under Cyclic Wet–Dry Conditions. *Buildings*, *15*(2), 228. https://doi.org/10.3390/buildings15020228

Sagidullina, N., Kim, J., Satyanaga, A., Ku, T., & Moon, S.-W. (2024). Mechanical and microstructural investigations on cement-treated expansive organic subgrade soil. *Geomechanics and Engineering*, *38*(4), 353-366. https://doi.org/10.12989/gae.2024.38.4.353

Shekhar, S., Bansode, A., & Salim, A. (2022). A Comparative study of Hyper-Parameter Optimization Tools. https://arxiv.org/abs/2201.06433

Shen, Z., Jiang, M., & Thornton, C. (2016). DEM simulation of bonded granular material. Part I: Contact model and application to cemented sand. *Computers and Geotechnics*, *75*, 192-209. https://doi.org/10.1016/J.COMPGEO.2016.02.007

Subramanian, S., Moon, S.-W., Moon, J., & Ku, T. (2018). CSA-Treated Sand for Geotechnical Application: Microstructure Analysis and Rapid Strength Development. *Journal of Materials in Civil Engineering*, *30*(12), 04018313-04018313. https://doi.org/10.1061/(ASCE)MT.1943-5533.0002523

Vinoth, G., Moon, S. W., Moon, J., & Ku, T. (2018). Early strength development in cement-treated sand using low-carbon rapid-hardening cements. *Soils and Foundations*, *58*(5), 1200-1211. https://doi.org/10.1016/J.SANDF.2018.07.001

Wang, Y. H., & Leung, S. C. (2008). Characterization of Cemented Sand by Experimental and Numerical Investigations. *Journal of Geotechnical and Geoenvironmental Engineering*, *134*(7), 992-1004. https://doi.org/10.1061/(ASCE)1090-0241(2008)134:7(992)

Wu, M., Huang, R., & Wang, J. (2021). DEM simulations of cemented sands with a statistical representation of micro-bond parameters. *Powder Technology*, *379*, 96-107. https://doi.org/10.1016/J.POWTEC.2020.10.047

Yang, P., Kavazanjian, E., & Neithalath, N. (2019). Particle-Scale Mechanisms in Undrained Triaxial Compression of Biocemented Sands: Insights from 3D DEM Simulations with Flexible Boundary. *International Journal of Geomechanics*, *19*(4), 04019009-04019009. https://doi.org/10.1061/(ASCE)GM.1943-5622.0001346

Chapter 7 – Appendices

Appendix A – DEM simulation code (Python, FISH code)

```
import numpy as np
import itasca as it
import json
import time
import os
from datetime import datetime

def simulation_prep(
    emod,
    fric,
    porosity,
    ln_kn,
    ln_ks,
    pb_ten,
    pb_coh,
    pb_kn,
    pb_ks,
    pb_fa
):
    it.command("""
    model new
    model title 'Sleeved Triaxial Test of a Bonded Material'
    model random 10001


    ;   -----   SEARCH PARAMETERS   -----
    ; Parallel-Bond
    [_pb_ten = {pb_ten}]
    [_pb_coh = {pb_coh}]
    [_pb_kn = {pb_kn}]
    [_pb_ks = {pb_ks}]
    [_pb_fa = {pb_fa}]

    [_fric = {fric}]
    [_ln_kn = {ln_kn}]
    [_ln_ks = {ln_ks}]
    [_emod = {emod}]
    [_porosity = {porosity}]

    ;   -----   SPECIMEN DIMENSIONS   -----
    [diameter = 0.038]
    [rad = diameter/2.0]
    [height = 2.0*diameter]
    [rad2 = rad*1.3]

    ;   -----   ETC   -----
    [magnification = 4.0]
    [damping = 0.7]
    [strain_rate = height*5.0e-6] ;   m/sec
    [bond_gap_ratio = 50]
    [dmin = 0.0005*magnification]
    [bond_gap = dmin/bond_gap_ratio]
    [kratio = 1.00]
    [density = 2600]
    [segments = 6]

    model domain extent [-rad2*1.1] [rad2*1.1] ...
                        [-rad2*1.1] [rad2*1.1] ...
```

```
                        [-height-0.1] [height+0.1] ...
                        condition destroy
    model largestrain on
    model mechanical timestep scale
    model gravity 9.81

    [halfLen = height/2.0]
    [freeRegion = height/2.0*0.8]
    [membrane_length = height+diameter]

    geometry edge create by-arc ...
        origin (0,0,[-membrane_length/2.0]) ...
        start ([rad*(-1)],0,[-membrane_length/2.0]) ...
        end (0,[rad*(-1)],[-membrane_length/2.0]) ...
        segments [segments]
    geometry edge create by-arc ...
        origin (0,0,[-membrane_length/2.0]) ...
        start (0,[rad*(-1)],[-membrane_length/2.0]) ...
        end ([rad],0,[-membrane_length/2.0]) ...
        segments [segments]
    geometry edge create by-arc ...
        origin (0,0,[-membrane_length/2.0]) ...
        start ([rad],0,[-membrane_length/2.0]) ...
        end (0,[rad],[-membrane_length/2.0]) ...
        segments [segments]
    geometry edge create by-arc ...
        origin (0,0,[-membrane_length/2.0]) ...
        start (0,[rad],[-membrane_length/2.0]) ...
        end ([rad*(-1)],0,[-membrane_length/2.0]) ...
        segments [segments]
    geometry generate from-edges ...
                            extrude (0,0,[membrane_length]) ...
                            segments [segments*2]

    structure shell import from-geometry 'Default' element-type dkt-cst
    structure node group 'middle' range position-z [-height/2.0] [height/2.0]
    structure node group 'top' range position-z [height/2.0] [membrane_length/2.0]
    structure node group 'bot' range position-z [-membrane_length/2.0] [-height/2.0]
    structure shell group 'middle' range position-z [-height/2.0] [height/2.0]
    structure shell property isotropic (1e8, 0.5) thick 1e-4 density 800

    model cycle 0

    fish define setLocalSystem
        loop foreach local s struct.node.list()
            local p = struct.node.pos(s)
            local nid = struct.node.id.component(s)
            local mvec = vector(0,0,comp.z(p))
            zdir = math.unit(p-mvec)
            ydir = vector(0,0,1)
            command
                structure node system-local z [zdir] y [ydir] ...
                    range component-id [nid]
            endcommand
        endloop
        command
            structure node fix system-local
        endcommand
    end
    @setLocalSystem
    structure damp local
    structure node fix velocity rotation

    [wall_thickness = diameter/2.0]
    wall-structure create
```

```
wall generate  name 'platenTop' cylinder  ...
    axis 0 0 1                              ...
    base 0 0 [halfLen]                      ...
    height [wall_thickness]                 ...
    radius [rad]                            ...
    resolution 0.27                         ...
    one-wall

wall generate name 'platenBottom' cylinder   ...
    axis 0 0 -1                              ...
    base 0 0 [-halfLen]                      ...
    height [wall_thickness]                  ...
    radius [rad]                             ...
    resolution 0.27                          ...
    one-wall

wall resolution full
wall attribute cutoff-angle 20

fish define granulometry
    global exptab = table.create('experimental')
    table(exptab,0.00030*magnification) = 0.002
    table(exptab,0.00060*magnification) = 0.0182
    table(exptab,0.00118*magnification) = 0.9865
    table(exptab,0.00260*magnification) = 0.9998
    table(exptab,0.00475*magnification) = 1
end
@granulometry

contact cmat default model linear property kn 1e10

contact cmat default type ball-ball model linearpbond ...
                                    property kn 1e10

ball distribute box [-rad] [rad] [-rad] [rad] [-halfLen] [halfLen] ...
    porosity [_porosity] ...
    number-bins 5                                                 ...
    bin 1                                                          ...
        radius [0.5*dmin] [0.5*table.x(exptab,1)]                  ...
        volume-fraction [table.y(exptab,1)]                        ...
    bin 2                                                          ...
        radius [0.5*table.x(exptab,1)] [0.5*table.x(exptab,2)]     ...
        volume-fraction [table.y(exptab,2)-table.y(exptab,1)]      ...
    bin 3                                                          ...
        radius [0.5*table.x(exptab,2)] [0.5*table.x(exptab,3)]     ...
        volume-fraction [table.y(exptab,3)-table.y(exptab,2)]      ...
    bin 4                                                          ...
        radius [0.5*table.x(exptab,3)] [0.5*table.x(exptab,4)]     ...
        volume-fraction [table.y(exptab,4)-table.y(exptab,3)]      ...
    bin 5                                                          ...
        radius [0.5*table.x(exptab,4)] [0.5*table.x(exptab,5)]     ...
        volume-fraction [table.y(exptab,5)-table.y(exptab,4)]      ...
    range cylinder end-1 (0,0,[-halfLen]) end-2 (0,0,[height]) ...
          radius [rad*0.95]
ball attribute density [density] damp [damping]

measure create id 1 radius [rad2] ...
                    bins 100 @dmin [table.x(exptab,5)]
measure dump id 1 table 'numerical'

contact cmat default ...
    model linearpbond ...
        method ...
            deformability emod [_emod] kratio [kratio]
model cycle 1000 calm 10
```

```
189.
190.     contact cmat default ...
191.         model linearpbond ...
192.             method ...
193.                 deformability emod [_emod] kratio [kratio]
194.     contact property fric [_fric]
195.     model solve calm 1000 ratio-average 1e-7 clock 10
196.
197.     contact method bond gap [_bond_gap] range contact type 'ball-ball'
198.     contact property ...
199.                 lin_mode 1 ...
200.                 lin_force 0 0 0 ...
201.                 pb_ten [_pb_ten] ...
202.                 pb_coh [_pb_coh] ...
203.                 pb_kn [_pb_kn] ...
204.                 pb_ks [_pb_ks] ...
205.                 pb_fa [_pb_fa]
206.
207.     fish define linear_parameters
208.         if _ln_kn > 1
209.             command
210.                 contact property kn [_ln_kn]
211.             endcommand
212.         endif
213.
214.         if _ln_ks > 1
215.             command
216.                 contact property ks [_ln_ks]
217.             endcommand
218.         endif
219.     end
220.     @linear_parameters
221.
222.     model calm
223.     ball fix velocity spin
224.     model cycle 2
225.     ball free velocity spin
226.
227.     structure node free velocity rotation range group 'middle'
228.     structure node free velocity-z range group 'top'
229.     structure node free velocity-z range group 'bot'
230.
231.     [platenTop = wall.find('platenTop')]
232.     [platenBottom = wall.find('platenBottom')]
233.     [failureStress = 0]
234.     [currentStress = 0]
235.     [failureStrain = 0]
236.     [area = math.pi()*rad^2.0]
237.     [platenTopLoc = comp.z(wall.pos(platenTop))]
238.     [platenBottomLoc = comp.z(wall.pos(platenBottom))]
239.
240.     fish define halt
241.         halt = 0
242.
243.         if strain > 20
244.             halt = 1
245.         endif
246.
247.         if strain > 5
248.
249.             if failureStress/stress > 2.5
250.                 halt = 1
251.             endif
252.
253.             if strain/failureStrain > 4
```

```
                halt = 1
            endif
        endif
    end

    fish define apply_pressure(beginIn,ending,increment)
        command
            ball attribute displacement (0,0,0)
            structure node initialize displacement (0,0,0)
        endcommand

        begin = -beginIn
        ending = -ending
        increment = -increment

        loop while (math.abs(begin) < math.abs(ending))
            begin = begin + increment
            command
                ;apply the confining stress
                structure shell apply [begin] range group 'middle'
                ;apply the same confining stress on the platens
                wall servo force (0,0,[begin*area]) activate true ...
                     range name 'platenTop'
                wall servo force (0,0,[-begin*area]) activate true ...
                     range name 'platenBottom'
                model cycle 200
                model calm
            endcommand
        endloop

        command
            model cycle 1000
            wall servo activate false
            wall attribute velocity (0,0,0) range name 'platenTop'
            wall attribute velocity (0,0,0) range name 'platenBottom'
        endcommand
    end

    model save 'beforeApplication'

    ;@apply_pressure(0, 0.5e6, [0.5e6/10.0])
    ;model save 'to_500kPa'

    ;model restore 'to_500kPa'
    @apply_pressure(0, 1e6, [1.0e6/10.0])
    model save 'to_1000kPa'

    ;model restore 'to_1000kPa'
    ;@apply_pressure(1.0e6, 1.5e6, [1.5e6/15.0])
    ;model save 'to_1500kPa'
    """.format(
        emod=emod,
        fric=fric,
        porosity=porosity,
        ln_kn=ln_kn,
        ln_ks=ln_ks,
        pb_ten=pb_ten,
        pb_coh=pb_coh,
        pb_kn=pb_kn,
        pb_ks=pb_ks,
        pb_fa=pb_fa
    ))

def simulation_start(pressure):
    for filename in [
```

```
        'generated_data/strain_data.txt',
        'generated_data/stress_data.txt'
    ]:
        with open(filename, 'w') as file:
            pass

    if pressure == 0.5e6:
        it.command("""
        model restore 'to_500kPa'
        """)

    elif pressure == 1e6:
        it.command("""
        model restore 'to_1000kPa'
        """)

    elif pressure == 1.5e6:
        it.command("""
        model restore 'to_1500kPa'
        """)

    else:
        print(f"Wrong pressure: {pressure}")
        return

    it.command("""
    fish define stress
        local topForce = math.abs(comp.z(wall.force.contact(platenTop)))
        local botForce = math.abs(comp.z(wall.force.contact(platenBottom)))

        currentStress = (topForce + botForce) ...
                        /(2.0 * area) - {pressure}
        stress = currentStress

        strain = (height - (comp.z(wall.pos(platenTop)) - ...
                comp.z(wall.pos(platenBottom)) - ...
                wall_thickness))/height * 100

        if failureStress <= currentStress
            failureStress = currentStress
            failureStrain = strain
        endif
    end

    wall attribute velocity-z [-strain_rate] range name 'platenTop'
    structure node initialize velocity-z [-strain_rate] range group 'top'
    wall attribute velocity-z [strain_rate] range name 'platenBottom'
    structure node initialize velocity-z [strain_rate] range group 'bot'

    ball attribute displacement (0,0,0)
    structure node initialize displacement (0,0,0)
    fish history @stress
    fish history @strain
    """.format(pressure=pressure))

    it.command("""
    model solve fish-halt halt clock 20
    model save 'lastModel'
    [io.out(string(failureStress) + 'Pa ')]
    [io.out('at' + string(failureStrain) + '% strain')]
    history export 1 file 'generated_data/stress_data.txt'
    history export 2 file 'generated_data/strain_data.txt'
    """)

def get_numerical_data():
```

```
    with open('generated_data/stress_data.txt', 'r') as f:
        stress_lines = f.readlines()[2:]  # Skip the first two lines
        stress = [float(line.split()[1]) / 1e6 for line in stress_lines]

    with open('generated_data/strain_data.txt', 'r') as f:
        strain_lines = f.readlines()[2:]  # Skip the first two lines
        strain = [float(line.split()[1]) for line in strain_lines]

    numerical_data = np.column_stack((strain, stress))
    return numerical_data

def new_params_check():
    global last_mod_time, params_file

    while True:

        it.command("pause 3")
        print("Waiting for new simulation_params...")

        # Check if simulation_params.json was updated
        try:
            mod_time = os.path.getmtime(params_file)

            # Continue waiting if simulation_params.json was generated before the run
            if mod_time < start_time:
                continue

            # Continue waiting if simulation_params.json is from previous trial
            if mod_time <= last_mod_time:
                continue

            # If simulation_params.json was generated after the run
            # and not from the previous trial
            break

        # Continue waiting if simulation_params.json was not generated
        except FileNotFoundError:
            continue

    last_mod_time = mod_time  # Update last modified time

    return True


start_time = datetime.now().timestamp()
params_file = "generated_data/params.json"
last_mod_time = 0
pressure_values = [0.5e6, 1.0e6, 1.5e6]

while True:
    if new_params_check():
        # Read new parameters
        with open(params_file, "r") as f:
            params = json.load(f)

        emod = params["ln_emod"]
        fric = params["fric"]
        porosity = params["porosity"]
        ln_kn = params["ln_kn"]
        ln_ks = params["ln_ks"]
        pb_ten = params["pb_ten"]
        pb_coh = params["pb_coh"]
        pb_kn = params["pb_kn"]
        pb_ks = params["pb_ks"]
        pb_fa = params["pb_fa"]
```

```
449.
450.         # Start simulation
451.         simulation_prep(
452.             emod,
453.             fric,
454.             porosity,
455.             ln_kn,
456.             ln_ks,
457.             pb_ten,
458.             pb_coh,
459.             pb_kn,
460.             pb_ks,
461.             pb_fa
462.         )
463.
464.         results = {}
465.         pressure = 1.0e6
466.         try:
467.             #for pressure in pressure_values:
468.             simulation_start(pressure)
469.
470.             numerical_data = get_numerical_data().tolist()
471.
472.             # Store results in a dictionary
473.             results[f"{pressure/1e6:.1f}MPa"] = numerical_data
474.
475.             # Save to JSON file
476.             with open("generated_data/simulation_results.json", "w") as f:
477.                 json.dump(results, f, indent=4)
478.
479.         except Exception as e:
480.             error_message = {"error": f"error: {e}"}
481.             with open("generated_data/simulation_results.json", "w") as f:
482.                 json.dump(error_message, f, indent=4)
483.
```

Appendix B – Optuna optimization code (Python)

```python
import pylab as plt
import numpy as np
import json
import time
import os
import optuna
from datetime import datetime

from my_functions import numerical, experimental, euclidean_distance, format_param

# File paths
params_file = "generated_data/params.json"
start_time = datetime.now().timestamp()
iteration = 0
last_mod_time = 0

# Define parameter bounds
def objective(trial):
    """Optimization function for PFC3D simulations"""

    global iteration, last_mod_time
    iteration = trial.number

    emod = trial.suggest_float("ln_emod", 1.0e7, 1.0e10, log=True)
    fric = trial.suggest_float("fric", 0.25, 1.0)
    porosity = trial.suggest_float("porosity", 0.3, 0.5)
    ln_kn = trial.suggest_float("ln_kn", 1.0e5, 5.0e6, log=True)
    ln_kratio = trial.suggest_float("ln_kratio", 0.05, 10.0, log=True)
    pb_ten = trial.suggest_float("pb_ten", 1.0e3, 1.0e8, log=True)
    pb_coh = trial.suggest_float("pb_coh", 1.0e3, 1.0e8, log=True)
    pb_kn = trial.suggest_float("pb_kn", 1.0e3, 1.0e8, log=True)
    pb_kratio = trial.suggest_float("pb_kratio", 0.05, 20.0, log=True)
    pb_fa = trial.suggest_float("pb_fa", 20.0, 45.0)

    # Save new parameters to JSON
    params = {
        "ln_emod": emod,
        "fric": fric,
        "porosity": porosity,
        "ln_kn": ln_kn,
        "ln_ks": ln_kn/ln_kratio,
        "pb_ten": pb_ten,
        "pb_coh": pb_coh,
        "pb_kn": pb_kn,
        "pb_ks": pb_kn/pb_kratio,
        "pb_fa": pb_fa
    }

    with open(params_file, "w") as f:
        json.dump(params, f)

    # Wait for simulations
    if simulation_check():
        pressure = 1e6
        deviation = deviation_calc(pressure)

        return deviation

def simulation_check():
    global last_mod_time
    print("Waiting for simulation...")

```

```
    while True:
        time.sleep(10)
        try:
            mod_time = os.path.getmtime("generated_data/simulation_results.json")

            if mod_time < start_time:
                continue

            if mod_time <= last_mod_time:
                continue

            print("Starting analysis...")
            break

        except FileNotFoundError:
            continue

    last_mod_time = mod_time  # Update modification time
    return True

def deviation_calc(pressure):
    experimental_data = experimental(pressure)
    numerical_data = numerical(pressure)

    # Aligning experimental data with numerical data by
    # interpolating the experimental stress values at the numerical strain values
    interpolated_stress = np.interp(
        numerical_data[:, 0],
        experimental_data[:, 0],
        experimental_data[:, 1]
    )
    experimental_data = np.column_stack((numerical_data[:, 0], interpolated_stress))

    # Calculating deviations using Euclidean distance (for failure points) and RMSE
    error1 = euclidean_distance(numerical_data, experimental_data)
    error2 = np.sqrt( np.mean(
        (numerical_data - experimental_data) ** 2
    ))

    # Final error calculation
    weight1 = 300  # Euclidian
    weight2 = 10  # RMSE
    deviation = weight1 * error1 + weight2 * error2

    # Plotting results
    plot_results(experimental_data, numerical_data, deviation, pressure)

    return deviation

def plot_results(experimental_data, numerical_data, deviation, pressure):
    with open(params_file, "r") as f:
        params = json.load(f)

        # Format keys with LaTeX
        formatted_params = {
            k: (
                format_param(k, v)
            )
            for k, v in params.items()
        }

        # Generate LaTeX-formatted parameter text for the side box
        param_text = "\n".join(formatted_params.values())

    # Creating the plot
```

```
    plt.figure(figsize=(8, 6))
    plt.subplots_adjust(right=0.75)  # Leaving space for text on the right
    plt.plot(experimental_data[:, 0], experimental_data[:, 1], label='Experimental')
    plt.plot(numerical_data[:, 0], numerical_data[:, 1], label='Numerical')
    plt.xlabel("Strain (%)")
    plt.ylabel("Stress (MPa)")
    plt.title(
        f'Iteration: {iteration}, Pressure: {pressure/1e6:.1f}MPa, Deviation: {deviation:.3f}'
    )
    plt.legend(loc="best")

    # Displaying current parameter values outside the plot area
    plt.gcf().text(0.77, 0.5, param_text,
                   fontsize=10, verticalalignment='center', bbox=dict(facecolor='white'))

    # Ensuring 'plots' folder exists
    if not os.path.exists('plots'):
        os.makedirs('plots')

    # Generate filename with timestamp
    timestamp = time.strftime("%Y-%m-%d %H-%M-%S")
    filename = f"plots/{timestamp}__iteration-{iteration}__pressure-{pressure/1e6:.1f}MPa.png"

    # Saving the plot
    plt.savefig(filename)
    print("Plot saved")
    plt.close()

# Run optimization
study = optuna.create_study(direction="minimize", study_name="characterization optimization",
                            storage="sqlite:///logs/optuna_progress.db", load_if_exists=True)
study.optimize(objective, n_trials=10000)

# Print best parameters
print("Best parameters:", study.best_params)
print("Least deviation:", study.best_value)

```

Appendix C – Python module for custom functions

```python
import numpy as np
import json
import csv
import optuna

def experimental(pressure):
    # strain - %
    # stress - kPa
    file_mapping = {
        5e5:    "7%_7-days_0.5Mpa.csv",
        1e6:    "7%_7-days_1.0Mpa.csv",
        1.5e6:  "7%_7-days_1.5Mpa.csv"
    }

    if pressure not in file_mapping:
        raise ValueError(f"Invalid pressure value: {pressure}")

    file_name = file_mapping[pressure]

    # Read CSV and store as 2D array
    data = []
    with open(file_name, mode="r") as file:
        reader = csv.reader(file)
        for row in reader:
            data.append([float(x) for x in row])  # Convert to float

    experimental_data = np.array(data)

    # Convert stress to MPa
    experimental_data[:, 1] /= 1000
    return experimental_data

def numerical(pressure):
    """
    Retrieves numerical simulation data corresponding to a given pressure.

    Args:
        pressure (float): The desired pressure level (e.g., 5e5, 1e6, 1.5e6 Pa).

    Returns:
        np.ndarray: The numerical stress-strain data for the given pressure.

    Raises:
        optuna.exceptions.TrialPruned: If the simulation contains an error.
    """

    data_mapping = {
        5e5:    "0.5MPa",
        1e6:    "1.0MPa",
        1.5e6:  "1.5MPa"
    }

    pressure_name = data_mapping[pressure]

    with open("generated_data/simulation_results.json", "r") as f:
        data = json.load(f)

    if "error" in data:
        raise optuna.exceptions.TrialPruned(f"Error in simulation: {data['error']}")

    numerical_data = np.array(data[pressure_name])
    return numerical data
```

```python

def euclidean_distance(numerical_data, experimental_data):
    """
    Computes the normalized Euclidean distance between the peak stress points
    of numerical and experimental datasets.

    The function finds the maximum stress points in both datasets, normalizes
    them using the respective strain and stress ranges, and then calculates
    the Euclidean distance in the normalized space.

    Args:
        numerical_data (np.ndarray): A 2D NumPy array of shape (n, 2) where
                                     the first column represents strain and
                                     the second column represents stress.
        experimental_data (np.ndarray): A 2D NumPy array of shape (m, 2) where
                                        the first column represents strain and
                                        the second column represents stress.

    Returns:
        float: The Euclidean distance between the normalized peak stress points
               of numerical and experimental data.
    """
    # Find indices of max stress for both datasets
    idx_num = np.argmax(numerical_data[:, 1])
    idx_exp = np.argmax(experimental_data[:, 1])

    # Extract max stress points (strain, stress)
    point_num = numerical_data[idx_num]
    point_exp = experimental_data[idx_exp]

    # Compute axis ranges
    strain_range = max(numerical_data[:, 0].max(), experimental_data[:, 0].max()) - \
                   min(numerical_data[:, 0].min(), experimental_data[:, 0].min())

    stress_range = max(numerical_data[:, 1].max(), experimental_data[:, 1].max()) - \
                   min(numerical_data[:, 1].min(), experimental_data[:, 1].min())

    # Avoid division by zero
    strain_range = strain_range if strain_range != 0 else 1
    stress_range = stress_range if stress_range != 0 else 1

    # Normalize the coordinates
    scaled_x_num = point_num[0] / strain_range
    scaled_y_num = point_num[1] / stress_range
    scaled_x_exp = point_exp[0] / strain_range
    scaled_y_exp = point_exp[1] / stress_range

    # Compute the scaled Euclidean distance
    scaled_distance = np.sqrt((scaled_x_exp - scaled_x_num) ** 2 + \
                              (scaled_y_exp - scaled_y_num) ** 2)
    return scaled_distance

def format_param(param_name, value=None):
    """
    Formats a parameter name and value into LaTeX notation with units.

    Args:
        param_name (str): The parameter's key name.
        value (float or int, optional): The numerical value to format. Defaults to None.


    Returns:
        str: A LaTeX-formatted string.
    """

```

```
    # LaTeX expressions for parameter names
    latex_expressions = {
        "pb_ten": r"\overline{\sigma}_c",  # Tensile strength
        "pb_coh": r"\overline{c}",  # Cohesion
        "pb_kn": r"\overline{k}_n",  # Normal stiffness
        "pb_ks": r"\overline{k}_s",  # Shear stiffness
        "pb_fa": r"\overline{\phi}",  # Friction angle
        "pb_emod": r"\overline{E}",  # Elastic modulus
        "fric": r"\mu",
        "damping": r"\zeta",
        "porosity": r"n",
        "bond_gap": r"R_{\text{bond}}",
        "pressure_build": r"N_{\text{pressure}}",
        "magnification": r"M_{\text{scale}}",
        "ln_kn": r"k_n",
        "ln_ks": r"k_s",
        "ln_emod": r"E",
    }

    # Get LaTeX name
    latex_name = latex_expressions.get(param_name, param_name)

    if value is None:
        return f"${latex_name}$"

    # Unit mapping based on parameter type
    units = {
        "pb_ten": "Pa",
        "pb_coh": "Pa",
        "pb_emod": "Pa",
        "ln_emod": "Pa",
        "pb_kn": "N/m",
        "pb_ks": "N/m",
        "ln_kn": "N/m",
        "ln_ks": "N/m",
        "pb_fa": "°",
        "bond_gap": "",
        "porosity": "",
        "fric": "",
        "damping": "",
        "pressure_build": "",
        "magnification": ""
    }

    # Format the value
    if param_name == "pressure_build" or value == 0:
        formatted_value = f"{value:.0f}"
    elif param_name in {"pb_fa", "fric", "damping", "bond_gap", "magnification"}:
        formatted_value = f"{value:.1f}"
    elif param_name in {"porosity"}:
        formatted_value = f"{value:.2f}"
    else:
        formatted_value = f"{value:.3e}"

    # Get unit (if available)
    unit = units.get(param_name, "")

    # Construct the formatted LaTeX text
    if unit:
        return f"${latex_name} = {formatted_value} \\text{{ {unit} }}$"
    else:
        return f"${latex_name} = {formatted_value}$"

```