A DISCRETE ELEMENT MODEL FOR THE NUMERICAL SIMULATION OF GRANULAR GEO-MATERIALS.

By

Agegnehu Gizaw

ADDIS ABABA INSTITUTE OF TECHNOLOGY
SCHOOL OF CIVIL AND ENVIRONMENTAL ENGINEERING
ADDIS ABABA,ETHIOPIA

Advisor
Dr.Ing Tensaye Gebremedhin

A thesis submitted to the School of Graduate Studies,
Addis Ababa University in Partial Fulfillment of the requirements
for the Degree of Master of Science in Geotechnical Engineering

November 16, 2021

ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

A DISCRETE ELEMENT MODEL FOR THE NUMERICAL SIMULATION OF GRANULAR GEO-MATERIALS.

By

Agegnehu Gizaw

Approved by Board of Examiners

1.	Tensaye GebreMedhin (Dr.-Ing)	_____	_____
	Advisor	Signature	Date
2.	_____	_____	_____
	Examiner(Internal)	Signature	Date
3.	_____	_____	_____
	Examiner(External)	Signature	Date
4.	_____	_____	_____
	Chairperson	Signature	Date

UNDERTAKING

I, the undersigned, declare that the research work entitled "Discrete Element model for the numerical simulation of granular geo-materials" is my original work. The work has not been presented as a thesis elsewhere. And all sources of materials used,have been properly acknowledged.

Agegnehu Gizaw	_____	_____
Researcher	Signature	Date

This is to certify that the above declaration made by the candidate is correct to the best of my knowledge.

Tensaye GebreMedhin (Dr.-Ing)	_____	_____
Advisor	Signature	Date

Abstract

This work presents computer simulation tools that use a coupled discrete-finite element methods for the modeling of granular materials. The simulation provides a new insight (perspective) in understanding the behavior of granular media at a microscopic level.It introduces and implements prevailing methods of computer simulation to examine the internal dynamics of granular material that are typically not amenable to direct observations in the laboratory and field investigations. The simulation techniques are implemented using an object-oriented programming approach via C++ and python in the open-source Linux package YADE.

The fundamental formulation of DEM ,FEM and the coupling scheme of discrete-finite element methods is presented. Along with Discrete Element soil sample and packing generation algorithms are discussed. Among the proposed packing algorithms, Multi-layer under compaction packing process is adopted since it mimics the natural layer-by-layer depositing process of granular materials . Accordingly, an algorithm is presented and coded in the platform (YADE). Then the generated DE soil sample is used to simulate a series of direct shear tests.

The microscopic view of Direct Shear Test (DST) is studied on the shear behavior of dense and loose sand. The output results of the numerical simulation is validated with laboratory DST. The deformation pattern, stress-strain relationship, and change are analyzed.The DEM assemblies with porosity =0.65 (dense sand) shows strain softening and volumetric dilation behavior. While strain hardening and volumetric contraction is observed for the assemblies with porosity =0.8 (loose sand) which is the typical shear behavior of Dense and loose sand in laboratory DST.Furthermore, the deformation pattern observed in the simulation is localized within the shear zone of the sample which is comparable to observations from the laboratory Direct Shear Test. when the shear stress arrives at a steady-state after large shear displacement, the soils inside the shear band reach a critical state while the entire sample doesn't. The critical properties of the sand in DST should be investigated from the shear band instead of the entire sample. This property is difficult to capture in laboratory DST however the simulations allow us to resolve the limitation.Finally, a technical manual for the development process of computer simulation tools to provide an efficient approach for solving coupled discrete-finite geotechnical problems is presented.

Acknowledgment

I wish to express my sincerest gratitude for my advisor Dr.Ing Tensaye Gebremedhin, who has shepherded me throughout this journey with his continuous guidance and encouragement. I enjoyed most of the time and importantly a great freedom in doing my research under his mentoring. I value the fruitful and joyful experience working with him which I believe will have a far-reaching influence on my own career to be a researcher.

Many thanks to my families, classmates and colleagues for their consistent moral and unfailing support. The success is theirs just as much as it is mine.

Above all special thanks to authors, researchers and programmers of the open-source community such as YADE , ESyS-Particle (Software developed by the Centre for Geoscience Computing at The University of Queensland), Escript, Python empowered with the NumPy/SciPy and matplotlib, packages such as ParaView, GenGeo and others. Even if I haven't met the communities in person, their work have great impact in motivating and guiding me throughout this thesis.

I thanks to LaTeX in which this thesis is organized edited up on.

Finally, I extend my thanks to all AAiT professors, staffs, librarians and others who directly or indirectly have contributed to the accomplishment of this thesis.

Publications

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Background

Different scholars have been studying the interaction between soil and geotechnical structures in the last few decades. Besides experimental studies, numerical methods have proven to be efficient in modeling the soil-structure interaction at a ***macroscopic*** level.

Classical continuum mechanics has been used to idealize soils and rocks, and numerical solution techniques such as the finite element method (FEM) have been successfully used to model these materials. The continuum mechanics models are phenomenological and are primarily concerned with the mathematical modeling of the observed phenomenon without giving detailed attention to the fundamental physical significance. This approach considers three completely independent assumptions: ***continuity, homogeneity, and isotropy***. This is an idealization of the materials for the representation of its mechanical behavior.

Considering the idealization of the material, continuum mechanics and the solution technique like the finite element method allows the analysis and modeling of various geotechnical engineering problems such as tunneling process (Mroueh and Shahrour 2002), deep excavation (L.Zdravkovic *et al.* 2005), and pile foundation (S.Karthigeyan *et al.* 2006). However, it is a challenging task to apply standard finite element methods to properly model the soil-structure interaction at the particle scale (***microscopic***) level. This is because the discrete nature makes the constitu-

tive relationship complex and needs an excessive number of parameters to be able to model the behavior accurately. FEM may not accurately capture engineering problems that involve particle movements and large deformation.

Granular materials consist of grains in contact and surrounding voids. The mechanical behavior of granular materials is, therefore, inherently ***discontinuous, heterogeneous***, and typically ***an-isotropic***. A different approach is the use of discontinuous or discrete models which directly treat the particles. The particulate nature is automatically simulated as discrete particles and forces are transferred through the contacts between particles. The material behavior is modeled realistically, considering the random distribution of particle's size and shape. In the case of cohesive materials, like rocks, the fracture can be obtained simply through the breakage of bonds between particles with adhesive contacts.

The most common discrete technique for the simulation of granular materials is the discrete element method (DEM). The method can handle a wide range of materials constitutive behaviors, contact laws, and arbitrary geometries. The DEM for modeling granular soils within the context of civil engineering was first introduced by (Cundall and Strack 1979a) . The particles were modeled as a random assembly of discrete discs. Many research studies have since been conducted to improve the simulation of angular grain shapes.

The DEM allows the use of different contact laws, linear and non-linear, depending on the process that will be simulated. This permits the simulation of a problem in different scales, and even the modeling of the behavior of the particles at their real size, involving the interaction of many particles in the system (refer to chapter 3). This technique is a very efficient tool for the analysis of different problems but does come with some drawbacks. The calculation of each particle-particle interaction, in some cases with complex contact laws, requires a high computational effort. Sometimes, a DEM simulation can take days or weeks. This restricts its application to simple problems with a small number of elements or simple shape particles. Another requirement of the DEM is the estimation of the contact model parameters that in some cases cannot be obtained directly.

The discrete element method (DEM) has proven to be promising for modeling geotechnical engineering problems, which involve granular material and large de-

formation (Herten and Pulsfort 1999),(Cui and O'Sullivan 2006) ,(Guerrero 2006) .Discrete models are more appropriate for problems with multiple discontinuities and particulate materials. The method has been gaining popularity in the analysis of granular materials, soils, and rocks. Many aspects of this method still require more profound investigation. Although DEM is considered efficient in modeling soil particles, it is challenging to properly model the behavior of structural elements using discrete particles due to the continuum behavior of the structure.

To take advantage of both the **FE** and **DE** methods, the continuum structures can be modeled using FE whereas the soil can be modeled using the DE method. The coupling of the finite and discrete element methods can efficiently model the behavior of the structure and the granular material. It is a promising approach for modeling granular soil-structure interaction. This approach has been used by several researchers to solve certain problems such as geosynthetic-reinforced earth structures (P.Villard *et al.* 2009), pile installation (Elmekati and Shamy 2009), and earth pressure on tunnel linings (Dang and Meguid 2011a).

1.2 Statement of the Problem

The primary advantage of computer simulations in scientific research is that the ability to look at internal dynamics that are typically not amenable to direct observation within the laboratory or field . Following its significance thus far several simulations of geotechnical engineering problems has been conducted. Numerical solutions like FEM, DEM and HYBRID techniques enable us to model the issues in computer programs and take the merit of computer simulations in scientific research .Especially coupled FE-DE method is a powerful numerical model to research the soil particle-structure interaction involving large deformation.In triaxial test simulation (Fakhimi 2009) used finite element to model the membrane and discrete element to model the soil sample. Likewise (P.Villard *et al.* 2009) proposed same approach to model earth structures reinforced by geosynthetic. (Dang and Meguid 2013a) also proposed a coupled FE-DE approach to model soil structure interaction problems involving large deformation . Although the coupled finite-discrete element approach has been utilized in geotechnical engineering to model certain soil-structure interaction problems, validation and calibration of the simulations with experimental data were limited .

Following the gap this thesis aims to advance within the use of the DEM technology, seeking to form contributions to enable a far better understanding of it. Different aspects of the DEM are going to be developed, like the estimation of parameters and particles packing techniques. These aspects are strongly related, because the behavior of the particles system depends on the initial configurations . Further aspects, such as improvement of the computational efficiency related to the simulations are going to be discussed.

A serious difficulty within the application of the DEM to real engineering problems is that the high computational cost in simulations involving an out sized number of particles. the foremost common thanks to solve this is often the utilization of parallel computing techniques, where multiple processors are used. These powerful simulation techniques are going to be implemented during this thesis using programming languages C++, python in open source Linux packages YADE (Šmilauer *et al.* n.d.) and esys-particle(Weatherley *et al.* n.d.). Validation of the simulation with experimental data are going to be performed. A coupling technique with contin-

uum based methods is introduced, and therefore the algorithms used for the contact resolution also are studied.

1.3 Objective

1.3.1 General Objectives

The overall objective of this thesis is to introduce and implement the powerful methods of computer simulation to investigate geotechnical engineering problems involving granular media and large deformation using coupled finite-discrete element procedure .This is achieved by addressing the following

- validate the use of the discrete element method in investigating geotechnical engineering problems involving granular media and large deformation.
- Develop computer simulation techniques to investigate the behavior of granular soil using coupled Finite-Discrete element method to examine the internal dynamics that are typically not amenable to direct observation in the laboratory or field.

1.3.2 Specific Objectives

The above General objectives achieved by addressing the following specific objectives.

- Develop soil specimen generation and packing algorithm in DEM simulation which replicate real soil deposition.
- Develop a python code to implement the packing algorithm by considering computational efficiency.
- Perform simulation of direct shear test using DEM .
- Validate the simulation results with experimental data.
- Review the theoretical formulation and the numerical implementation of a procedure for coupling the DEM with the FEM.

- Develop technical manual for the development process of computer simulation tools to provides an efficient approach for solving coupled FE-DE geotechnical problems

1.4 Research Methodology

The first and crucial step in engineering research is the idealization of physical problem to mathematical model .The physical problem under the study "usually involves an actual structure or structural components subjected to a certain loads" (*Finite Element procedure* 1996). The idealization to a mathematical model requires the establishment of the governing differential equations and boundary conditions, also called the **STRONG FORM** of a problem. The *strong form* consists of mathematical equations physically expressing the conditions that the exact solution of a problem must satisfy at any point inside the system under consideration.In idealization of the problem ,researchers usually makes certain assumptions on geometry,kinematics ,material law,loading , boundary conditions and etc ... Choosing appropriate mathematical model that are *effective* and *reliable* in predicting the behavior of the problem under investigation is the key step in engineering research.

Once the mathematical modeling is done ,several methods can be employed to come up with the solutions.Researchers use different approach to achieve the proposed objective of their study. The methodology (study design) they follow depends on nature of the problem ,knowledge and availability of resource(material and time).

Conducting an experiment is one of methods that can be applied to achieve the proposed objective of this study .Several experiments were conducted by different scholars to investigate the same problem ref(chapter 2). Beside the availability of resource, the method requires controlled and complicated laboratory setups .In addition , in experimental method the researcher only examine parameters which are perceptible to direct observations However in geotechnical engineering problems in general and in this thesis in particular there are parameters that are typically not amenable to direct observation in the laboratory or field . These parameters are called microscopic parameters .For instance it is impractical to measure the normal(k_n) and tangential(K_t) stiffness of a soil particles directly in laboratory test.However these parameter are essential in determining the over all(macroscopic) behavior of a soil sample . As a result in this thesis i employ numerical methods such as Discrete Element Method (DEM),Finite Element Method (FEM) and coupled Discrete-finite methods to achieve the desired objective of the thesis . In

addition object oriented programming approach is followed to implement the numerical simulations .laboratory test results from literature are used for validation of simulation result.

1.5 Scope of the Study and Limitation

Several geotechnical problems can be investigated using numerical methods such as finite element and discrete element. The coupling of these methods makes the solution even powerful. It could be used to solve problems related to tunneling, embankments, soil reinforcement, foundations laying on granular soil, slopes, etc. Even though the scope of the study is vast, It is difficult to cover all of the thematic areas in single research due to time and resource limitations. Hence, it is very crucial to limit the scope of the study based on the given resource and time.

This study will be merely to introduce and implement the powerful methods of computer simulation, to investigate geotechnical engineering problems related to granular soils involving large deformation. Coupled finite-discrete element numerical solution procedure is applied to tackle the problems. The study also uses suitable computer programs (*C++*) , scripts (*python*) and open source packages such as *esys-particle, YADE* that are implemented in *Linux* operating system .The thesis has inter-disciplinary nature since it combine computer science with geotechnical engineering. Hence the thematic area of the study can be categorized as *computer and geothechniques*.

CHAPTER 2

LITERATURE REVIEW

2.1 Discrete Element Modeling of Granular Soil

The discrete element method (DEM), distinguishes itself as much cheaper (economically cheap but computationally expensive) compared to experimental setups, and provides a large amount of information at the microscopic level, both geometric and mechanical and even energy transmission which should be very difficult to measure in laboratory.

The DEM proposed by (Cundall and Strack 1979b) may be traced back to the method of molecular dynamics(MD) which is widely used in material,chemical and physics science. The difference is the typical length scale in DEM is usually in millimeters which are much larger than molecular scale as in MD.Consequently, the rotational degree of freedom for the particles in DEM should be included.Also the interaction forces in the two methods are different due to the length scale involved. More recent DEM studies consider complex shaped particles as the effect of particle shape is conceived as important; while in MD there is no such concern. However, the numerical method is still a highly simplified model. Most tests are performed with circular/spherical particles, or clumps composed of circular/spherical particles to approximate the real shape. Besides the particle shape factor, the contact models implemented in DEM are somehow oversimplified.

Nevertheless, the method has proven itself to be promising as it has been demonstrated that DEM can reproduce laboratory tests qualitatively and even quantita-

tively as long as the parameters are well calibrated. For example, (Cundall and Strack 1979a) compared the DEM results with those from physical tests done by (A.Drescher and Jong 1972) using photo-elastic discs. The comparison indicates DEM can fairly replace the physical tests. More convincing tests have been done later on (Ishihara and Oda, 1998), (Masson and Martinez, 2001)(Thornton, 2000); (Thornton and Zhang, 2010) and many are under complicated loading path conditions such as true triaxial (Barreto *et al.* 2012) and rotational shearing (LI and YU 2010). Now, DEM is widely accepted and has significantly aided our understanding on the behavior of granular media. For example, Rothenburg and coworkers (Ouadfel and Rothenburg 2001); established a stress-force-fabric relationship based on DEM simulation with idealized ellipsoidal particles, which constitutes a step further towards the bridging between the micro-mechanics and the macro-mechanics of granular media.

The discrete element method has been used to study different geotechnical problems involving granular materials. Laboratory tests have been modeled using DEM to investigate the microscopic behavior of soil samples. The force distribution and shear band developing during a direct shear test using DEM were reported by Thornton 2003. (Cui and O'Sullivan 2006) employed DEM to investigate macroscopic and microscopic responses of granular soil samples under direct shear condition.(Park 2009) modeled rock joints under direct shear using bonded-particles. Triaxial tests of granular soil samples were modeled by (Ng *et al.* 2004). Similarily, simple shear test simulation using DEM was reported by (Jiang et al. 2003) and (Duriez et al. 2011). Numerical results of the above studies show a good agreement with experimental data which demonstrates the efficiency of DEM in simulating laboratory tests.

Large scale geotechnical problems have been also modeled using DEM. (Lobo Guerrero et al 2006) investigated the behavior of rail track ballast degradation during cyclic loading. The particle breakage process was visualized and the effect of crushing on the behavior of track ballast material was investigated. Bearing capacity of driven piles in crushable granular materials was studied by (Lobo Guerrero and Vallejo 2005) using breakable DE particles. (Deluzarche and Cambou 2006) employed the same approach to study rock fill dam behavior. Those simulations were capable of

simulating particle breakage process which is difficult to observe using experimental tests.

Earth pressure acting on vertical shafts was studied by (Herten and Pulsfort 1999).Spherical particles were used to model the soil domain. The circular shaft was assumed to behave as segments made from small flat walls. The shaft wall was gradually moved inward and lateral pressures acting on the shaft wall were recorded. Results of the numerical simulation were then compared with experimental data

A simplified DEM model was developed by (Maynar et al. 2005) to study the underground tunneling process. Input parameters for the numerical simulation were determined using triaxial test calibration. The excavation process was modeled and the tunnel face stability was analyzed. The thrust and torque evolution with respect to the movement of earth pressure balance machine were investigated.

(Gabrieli et al. 2009) studied the behaviour of a shallow foundation on a model slope. A particle upscale approach was proposed to reduce the number of DE particles required for the simulation. Displacement fields and contact force networks of the model were obtained at the microscopic level.

(Jenck et al.2009) employed DEM to model granular fill supported by piles. A simplified two-dimensional DE model was developed to investigate soil improvement using vertical rigid piles. Macro scale responses of the platform over piles such as soil arching, loads transfer to the piles and platform settlement were analyzed. The DE simulation was compared with experimental data .Parametric studies were performed to investigate the influence of microscopic parameters on the macroscopic response of the model.

2.2 Discrete Element Modeling of Soil-Structure Interaction

The modeling of soil-structure interaction using DEM has been reported by (Villard and Chareyre 2004), used two-dimensional DEM to model the failure of geosynthetic sheets anchored in trenches. The geosynthetic sheets were modeled using "dynamic spar elements" while back-fill soil was modeled using disk elements. Contact laws between dynamic spar elements and disks were introduced to assure their interaction. Pullout strengths of different anchorage shapes as well as deformation and failure mechanism of the system were investigated.

(McDowell 2006), and (Chen 2013) used DEM to model both the geogrid and the backfill soil. The geogrid was modeled using a set of spherical particles bonded together to form the geogrid shape. The interaction between the geogrid and the surrounding soil was obtained through the contact between discrete particles. The geogrid was then pulled out to investigate the peak mobilised resistance and associated displacement (McDowell 2006),wheras in (Chen 2013) the behavior of the geogrid-reinforced ballast under cyclic loading was investigated and the effectiveness of the geogrid reinforcement was examined

(J. Han *et al.* 2012)used DEM to model geogrid-reinforced embankment over piles. The 2D embankment was modeled using disk elements and the 2D geogrid was modeled using bonded particles. The changes of stresses, porosities and displacements within the embankment fill as well as the behavior of the geogrid were investigated.

In the above soil-structure interaction simulations, the structural elements were modeled using dynamic spar elements or bonded particles which do not represent the continuous nature of the structure. Moreover, due to the inflexibility of the bonded particles and dynamic spar elements, the real deformation as well as strains and stresses within the structure may not be accurately captured.

2.3 Finite Element Modeling of Soil-Structure Interaction

The finite element method has been widely used to model soil-structure interaction in many geotechnical engineering problems such as tunneling, deep excavation and pile foundation. Modeling the tunneling process using FEM has been performed by researchers including (Mroueh and Shahrour 2002),(Galli *et al.* 2004),(Meguid and Rowe 2006) and (Yoo 2013).Soil-wall interaction during excavation process has been studied by ((Faheem *et al.* 2004), (L.Zdravkovic *et al.* 2005) and (J.Finno *et al.* 2007)) . Similarly, FEM has been used to model soil-pile interaction ((Khodair and Hassiotis 2005),(Maheshwari *et al.* 2004),(S.Karthigeyan *et al.* 2006)). Soil-structure interaction is often assured using interface elements ((G.Beer 1985),(Langen and P.A.Vermeer 1991),(Karabatakis and Hatzigogos* 2002)). In these studies, the soil-structure interactions using interface elements were often considered at the macroscopic scale.

In geotechnical engineering problems such as those involving erosion voids next to tunnel lining ((Dang and Meguid 2011a) ,(Zienkiewicz and Huang, 1990)), earth pressure on cylindrical shaft ((V.G. n.d.),(Tobar and Meguid 2011)) and geogrid reinforced soil ((Agaiby and Jones 1995),(Palmeira 2009),(Michalowski 2004)), it is necessary to model the soil-structure interaction at the particle scale level to properly capture the soil particle-structure interaction. Voids due to erosion around tunnel linings often have irregular shapes and sizes and it is challenging to model the void development using FEM especially when the void size increases (Dang and Meguid 2011b)). The active earth pressure on a cylindrical shaft is generally reached with sufficient shaft wall movement. For the case of a shaft surrounded by granular soil, the required shaft movement to reach the full active condition was found to range from 2.5 to 4 percent of the shaft radius (Tran *et al.*,2012). The problem involves granular material and large deformation which makes it challenging to properly capture the earth pressure acting on the shaft wall using traditional FEM. In reinforced soil problems such as geogrid pullout test (Palmeira 2009), geogrid reinforced foundation (Michalowski 2004) and geogrid reinforced fill over void (Agaiby and Jones 1995), soil-geogrid interlocking effect is considered an important feature. However,

it is challenging to model the interlocking effect using FEM due to its particle based interaction. Moreover, the geogrid geometry is often simplified either as a truss structure (in 2D analysis) or a continuous sheet (in 3D analysis) which ignores the interlocking effect.

Although the above soil-structure interaction problems may be modeled using an adaptive re-meshing approach ((Zienkiewicz and Huang, 1990),(Zienkiewicz *et al*, 1995)) or a multiscale approach ((Hughes, 1995),(Garikipati and Hughes, 1998)), numerical simulations involving large soil deformation and unpredictable discontinuities using these approaches did not receive much research attention in the literature.

2.4 Coupling of Discrete-Finite Element Methods

To take advantage of both FEM and DEM, the coupling of the two numerical methods has been proposed. In this approach, the analyzed problem is divided into FE and DE domains. Several algorithms have been developed to assure the load transfer from the FE domain to DE domain and vice verse.

A procedure for combining finite and discrete elements to simulate the shot peening process was proposed by (K. Han and Peri 2000). Spherical shot was modeled using rigid discrete elements while the target material was modeled using deformable finite elements. The developed algorithm could capture the dynamic nature of the shot peening. However, the element size in the impact area should be no larger than d/10 where d is the diameter of the shot. This requires a large number of finite elements which results in high computational cost. A simulation of the shot peening process using a combined finite-discrete element approach was also reported by (Bhuvaraghan *et al.* ,2010).

An algorithm for coupling the finite and discrete element methods was reported by (Fakhimi 2009). The algorithm was capable of modeling deformable membrane in a laboratory triaxial test. The membrane was modeled using FE while the soil was modeled using DE. The membrane-soil interaction was based on the contact between DE and external face of the FE membrane. Both FE and DE computations were integrated explicitly using a central difference scheme.

(Xiao and b 2004) proposed a bridging domain method for coupling continuum models with molecular models. In this approach, the continuum and molecular domains were overlapped in a bridging sub-domain. Using the linearization of Halmitonian dynamics for both molecular and continuum models, this multi-scale method can avoid spurious wave reflections at the molecular/continuum interface. A multiple-time-step algorithm was also proposed within this framework. Similar techniques to combine the finite and discrete element domains were reported by (Dhia ,1998) and (Dhia and Rateau 2005).

Although the above coupling approaches have shown their efficiency in analyzing certain engineering problems, their applications in geotechnical engineering are still very limited. (P.Villard *et al.* 2009) proposed a coupled FE-DE approach to model earth structures reinforced by geosynthetic. Interface elements were proposed to

assure the interaction between the FE and DE domains. The framework was used to model the interaction between a geosynthetic sheet and surrounding soil. The geosynthetic sheet was modeled using FE while the soil was modeled using DE. (Elmekati and Shamy 2009) used a similar approach to model a rigid pile in contact with granular soil. The near-field zone surrounding the pile was modeled using DE whereas FE was used to model far-field zones. The interaction between the FE and DE domains was assured using a wall set of polygons having the same geometry of finite element surfaces at the interface.

(Dang and Meguid 2013b) proposed a coupled FE-DE approach to model soil-structure interaction problems involving large deformation. The domain involving the large deformation was modeled using DE while FE was used to model the rest of the domain. Interface elements at the boundary of the two domains were introduced to transmit interacting forces between the DE and FE domains. Explicit time integration with different damping schemes were applied to each domain in order to relax the system and to reach the convergence condition. Since a relatively coarse FE mesh is required, the algorithm reduces the computational time. The framework was then used to simulate a soft ground tunneling problem involving soil loss near an existing lining.

CHAPTER 3

RESEARCH METHODOLOGY

3.1 Introduction

Many physical problems encountered in solid mechanics and geotechnical engineering can be idealized and mathematically modeled using differential equations. However apart from some very specific cases it is generally not possible to write down the solution to these problems in closed form, meaning it is difficult to come up with direct analytical solutions to such problems. Hence to under stand the behavior of the solution ,mostly it is often necessary to construct an approximation via numerical methods and computer algorithms .

This research uses numerical methods such as DEM,FEM and coupled DEM/FEM to achieve the desired objective of the study .These methodology are selected based on the nature of the problem and their application . Numerical methods use exact algorithms to present numerical solutions to mathematical problems, Hence this chapter will briefly discus the numerical methods such as DEM,FEM coupling of DEM/FEM with their respective Algorithms to implement in the simulations.

3.2 The Discrete Element Method

The discrete (or distinct) element method (DEM) was originally developed by (Cundall and Strack 1979a) for the analysis of rock mechanics problems. The basic formulation of the DEM using spherical or cylindrical particles was later proposed by (Cundall and Strack 1979b) to investigate the constitutive laws for soil. (Cundall and Hart 1992) showed that DEM is better at modeling a discontinuous material than other numerical tools such as the finite element method.

The DEM assumes that the material can be represented by an assembly of rigid particles interacting among themselves. The overall behavior of the system is determined by the cohesive/frictional contact laws. The contact law can be seen as the formulation of the material model on the microscopic level. Cohesive bonds can be broken, which allows to simulate fracture of material and its propagation.

3.2.1 Equations of Motion

The translational and rotational motion of rigid spherical or cylindrical particles is described by means of Newton-Euler equations of rigid body dynamics.For the *i-th* element we have

$$\mathbf{m}_i \ddot{\mathbf{u}}_i = \mathbf{F}_i \tag{3.1}$$

$$\mathbf{I}_i \dot{\omega}_i = \mathbf{T}_i \tag{3.2}$$

where $\mathbf{u}_i$ is the displacement of the particle center in a fixed (inertial) coordinate frame $\mathbf{X}$, ω_i the angular velocity, $\mathbf{m}_i$ the element (particle) mass,$\mathbf{I}_i$ the moment of inertia,$\mathbf{F}_i$ the resultant force, and $\mathbf{T}_i$ the resultant moment about the central axes.

Vectors $\mathbf{F}_i$ and $\mathbf{T}_i$ are sums of all forces and moments applied to the *i-th* element.

$$\mathbf{F}_i = \sum_{c=1}^{n_c} \mathbf{F}_i^c + \mathbf{F}_i^{ext} + \mathbf{F}_i^{damp} \tag{3.3}$$

$$\mathbf{T}_i = \sum_{c=1}^{n_c} (\mathbf{r}_i^c \times \mathbf{F}_i^c + \mathbf{q}_i^c) + \mathbf{T}_i^{ext} + \mathbf{T}_i^{damp} \tag{3.4}$$

where $\mathbf{F}_i^{ext}$ and $\mathbf{F}_i^{ext}$ are external load,$\mathbf{F}_i^c$ the contact force for the interaction with neighboring spheres and other obstacles,$\mathbf{F}_i^{damp}$ and $\mathbf{T}_i^{damp}$ are the force and torque resulting from damping in the system (will be discussed in section 3.2.4, $\mathbf{r}_i^c$ is the

vector connecting the center of the particles of the *i-th* element with the contact point c, nc its number of particles being in contact and $\mathbf{q}_i^c$ are torques due to rolling or torsion (not related with the tangential forces).

The form of rotational equation (3.2) is valid for spheres and cylinders (in 2D) and is simplified with respect to a general form for an arbitrary rigid body with the rotational inertial properties represented by the second order tensor.

3.2.2 Contact Forces

Once contact between a pair of elements has been detected, the forces occurring at the contact point are calculated. The interaction between the two interacting bodies can be represented by the contact forces F_i andF_j, which by the Newton's third law satisfy the following relation:

$$\mathbf{F}_i = -\mathbf{F}_j \tag{3.5}$$

We take $\mathbf{F} = \mathbf{F}_i$ and decompose $\mathbf{F}$ into the normal and tangential components, $\mathbf{F}_n$ and $\mathbf{F}_t$, respectively Figure 3.1

$$\mathbf{F} = \mathbf{F}_n + \mathbf{F}_t = f_n\mathbf{n} + \mathbf{F}_t \tag{3.6}$$

where $\mathbf{n}$ is the unit vector normal to the particle surface at the contact point (this implies that it lies along the line connecting the centers of the two particles) and directed outwards from the particle i

$$\mathbf{n} = \frac{\mathbf{x}_j - \mathbf{x}_i}{\parallel \mathbf{x}_j - \mathbf{x}_i \parallel} \tag{3.7}$$

The contact forces f_n and $\mathbf{F}_t$ are obtained using a constitutive model formulated for the contact between two rigid spheres (or discs in 2D) Figure 3.2 . The contact interface in our formulation is characterized by the normal and tangential stiffness k_n and k_t, respectively, the Coulomb friction coefficient μ, and the contact damping coefficientc_n.

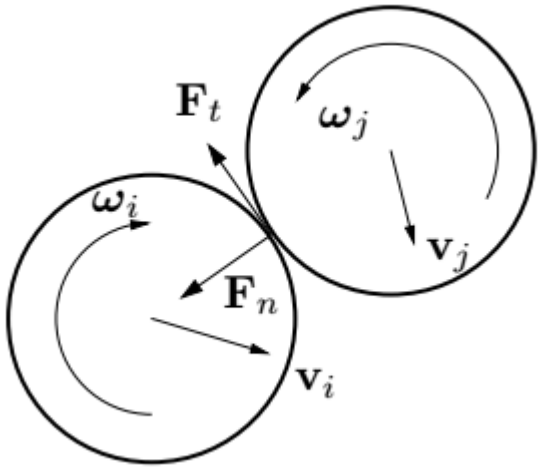


Figure 3.1: Decomposition of the contact force into the normal and tangential components.

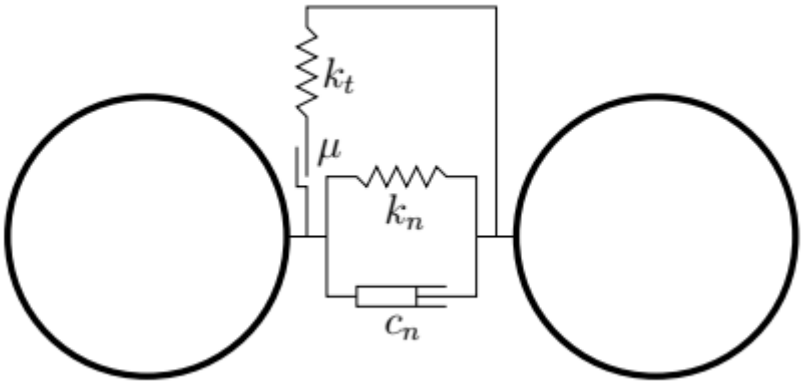


Figure 3.2: Rheological model of the contact.

Normal contact force

The normal contact force f_n is decomposed into the *elastic* part f_{ne} and the damping contact force f_{nd}

$$f_n = f_{ne} + f_{nd} \tag{3.8}$$

The damping part is used to decrease oscillations of the contact forces and to dissipate kinetic energy.

The elastic part of the normal contact force f_{ne} is proportional to the normal stiffness k_n and to the penetration of the two particle surfaces u_{rn}, i.e.

$$f_{ne} = k_n u_{rn} \tag{3.9}$$

The penetration is calculated as

$$u_{rn} = (\mathbf{x}_j - \mathbf{x}_i).\mathbf{n} - (r_j + r_i) \tag{3.10}$$

where $\mathbf{x}_i, \mathbf{x}_j$ are the center of the particles,$\mathbf{n}$ the normal unit vector between the particles Equation 3.7, and r_i, r_j their radii. If no cohesion is allowed, no tensile normal contact forces are allowed and hence

$$f_{ne} \leq 0 \tag{3.11}$$

If $u_{rn} < 0$ equation 3.9 holds, otherwise $f_{ne} = 0$. The contact with cohesion will be considered later on.

The contact damping force is assumed to be of viscous type and given by

$$f_{nd} = c_n \upsilon_{rn} \tag{3.12}$$

where υ_{rn} is the normal relative velocity of the centers of the two particles in contact defined by

$$\upsilon_{rn} = (\dot{\mathbf{u}}_j - \dot{\mathbf{u}}_i).\mathbf{n} \tag{3.13}$$

The damping coefficient c_n can be taken as a fraction α of the critical damping C_{cr} for the system of two rigid bodies with masses$m_i and m_j$, connected with a spring of stiffness k_n (with

$$c_n = \alpha C_{cr} = 2\sqrt{m_{ij} k_n} \tag{3.14}$$

with $0 \leq \alpha \leq 1$, and where m_{ij} is the reduced mass of the contact

$$m_{ij} = \frac{m_i m_j}{m_i + m_j} \tag{3.15}$$

The fractionα it is related with the *coefficient of restitution* c_r, which is a fractional value representing the ratio of speeds after and before of an impact, through

$$\alpha = \frac{-ln c_r}{\sqrt{\pi^2 + ln^2 c_r}} \tag{3.16}$$

Tangential frictional contact

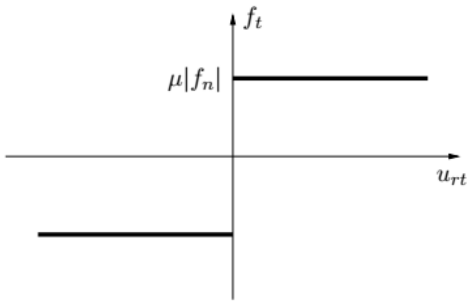
In the absence of cohesion (if the particles were not bonded at all or the initial cohesive bond has been broken) the tangential reaction $\mathbf{F}_t$ appears by friction opposing the relative motion at the contact point. The relative tangential velocity at the contact point $\mathbf{v}_{rt}$ is calculated from the following relationship .

$$\mathbf{v}_{rt} = \mathbf{v}_r - (\mathbf{v}_r.\mathbf{n})\mathbf{n} \tag{3.17}$$

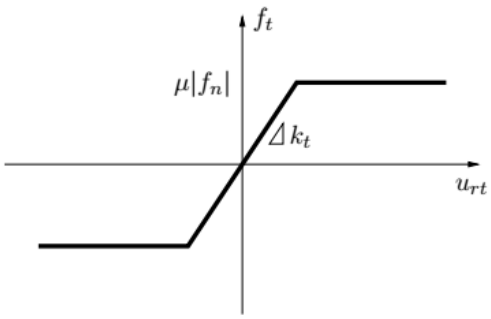
with

$$\mathbf{v}_r = (\dot{\mathbf{u}}_j + \omega_j \times \mathbf{r}_{cj}) - (\dot{\mathbf{u}}_i + \omega_i \times \mathbf{r}_{ci}) \tag{3.18}$$

where $\dot{\mathbf{u}}_i, \dot{\mathbf{u}}_j$, and ω_i, ω_j are the translational and rotational velocities of the particles, and $\mathbf{r}_{ci}$ and $\mathbf{r}_{cj}$ are the vectors connecting particle centers with contact points.



(a) classical Coulomb law.



(b) Regularized Coulomb law.

Figure 3.3: Friction force vs. relative tangential displacement.

The relationship between the friction force f_t and the relative tangential displacement v_{rt} for the classical Coulomb model (for a constant normal force f_n) is

shown in Figure 3.3 a. This relationship would produce non physical oscillations of the friction force in the numerical solution due to possible changes of the direction of sliding velocity. To prevent this, the Coulomb friction model must be regularized. The regularization procedure chosen involves decomposition of the tangential relative velocity into *reversible* and *irreversible* parts,$\mathbf{v}_{rt}^{r}$ and $\mathbf{v}_{rt}^{ir}$, respectively as:

$$\mathbf{v}_{rt} = \mathbf{v}_{rt}^{r} + \mathbf{v}_{rt}^{ir} \tag{3.19}$$

This is equivalent to formulating the frictional contact as a problem analogous to that of *elastoplasticity*, which can be seen clearly from the friction force-tangential displacement relationship in Figure 3.3 b. This analogy allows us to calculate the friction force employing the standard radial return algorithm. First a trial state is calculated.

$$\mathbf{F}_{t}^{trial} = \mathbf{F}_{t}^{n-1} - k_t \mathbf{v}_{rt} \triangle t \tag{3.20}$$

and then the slip condition is checked:

$$\phi^{trial} = \| \mathbf{F}_{t}^{trial} \| - \mu \mid f_n \mid \tag{3.21}$$

If $\phi^{trial} \leq 0$ we have stick contact and the friction force is assigned the trial value

$$\mathbf{F}_{t}^{n} = \mathbf{F}_{t}^{trial} \tag{3.22}$$

otherwise (slip contact) a return mapping is performed giving

$$\mathbf{F}_{t}^{n} = \mu \mid f_n \mid \frac{\mathbf{F}_{t}^{trial}}{\| \mathbf{F}_{t}^{trial} \|} \tag{3.23}$$

3.2.3 Constitutive Models

In the literature it is possible found different constitutive models for the contact between particles.The elastic response in the contact models can be separated into linear and nonlinear models. All of them can be represented by the normal and tangential stiffness, k_n and k_t respectively, defined in equations (3.9) and (3.20)

Hertz-Mindlin Contact Model

The Hertz contact model is the most classical nonlinear model used in particle collisions . From the Hertz theory for an elastic sphere i the normal stiffness may be written as:

$$k_n = \Big(\frac{2}{3}\frac{G\sqrt{2r'}}{(1-\nu)}\Big)\sqrt{\upsilon_{rn}} \tag{3.24}$$

where r' is called *contact radius*, defined as:

$$r' = \frac{2r_i r_j}{r_i + r_j} \tag{3.25}$$

As a complement to the Hertz model, which considers just the normal collision, Mindlin and Deresiewicz extended the theory for the stiffness in the tangential direction as:

$$k_t = \Big(\frac{2(G^2 3(1-\nu)r')^{\frac{1}{3}}}{2-\nu}\Big) \mid f_{ne} \mid^{\frac{1}{3}} \tag{3.26}$$

The Hertz-Mindlin contact model has been extensively used for granular dynamic simulations . A more general version of this model considers a modified contact damping coefficient (3.12) in order to avoid the viscous force when $u_{rm} = 0$

$$f_{nd} = c_n \upsilon_{rn} \upsilon_{rn} \tag{3.27}$$

The Hertz theory does not work in the case of bonded particles, and it is just reserved for contacts under compression.

Cundall and Strack Contact model

Cundall and Strack (Cundall and Strack 1979b) defines a linear stiffness proportional to the particle size.This model assume that the elastic connection along the line passing through the centers of any two particles in contact is formed by two springs in series. Each spring which its own stiffness K_n^i.

$$K_n^i = 4E_c r_i \tag{3.28}$$

where E_c is called *particle elastic modulus* and with the serial connection between particles we have

$$k_n = \frac{K_n^i K_n^j}{K_n^i + K_n^j} \tag{3.29}$$

Considering two particles of with the same mechanical properties, and radius r_i and r_j, the contact stiffness in normal and tangential direction are defined as :

$$k_n = 2E_c r' \tag{3.30}$$

$$k_t = \frac{k_n}{k} \tag{3.31}$$

where k is the stiffness ratio $\frac{k_n}{k_t}$

The most particular aspect of this linear model presents an scaled stiffness, directly related with the effective contact radius. This model allows bonded contacts for the simulation of cohesive materials.In this thesis Cundall and Strack (Cundall and Strack 1979b) contact model is used .

3.2.4 Background Damping

The contact damping previously described is a function of the relative velocity of the contacting bodies. It is sometimes necessary to apply damping for non-contacting particles to dissipate their energy. We have considered two types of such damping of viscous and non-viscous type, referred here as background damping. In both cases damping terms $\mathbf{F}_i^{damp}$ and $\mathbf{T}_i^{damp}$ are added to equations of motion 3.1 and 3.2.

In the case of the viscous damping, the value of the damping force is proportional to the magnitude of the velocity

$$\mathbf{F}_i^{damp} = -\alpha^t m_i \dot{\mathbf{u}}_i \tag{3.32}$$

$$\mathbf{T}_i^{damp} = -\alpha^r I_i \omega_i \tag{3.33}$$

For the non-viscous damping, the damping force is proportional to the magnitude to the resultant force and resultant moment in the direction of the velocity.

$$\mathbf{F}_i^{damp} = -\alpha^t \parallel \mathbf{F}_i' \parallel \frac{\dot{\mathbf{u}}_i}{\parallel \dot{\mathbf{u}}_i \parallel} \tag{3.34}$$

$$\mathbf{T}_i^{damp} = -\alpha^r \parallel \mathbf{T}'_i \parallel \frac{\omega_i}{\parallel \omega_i \parallel} \tag{3.35}$$

where α^t, α^r are damping constants, and $\mathbf{F}'_i$ $\mathbf{F}'_i$ are defined as

$$\mathbf{F}'_i = \sum_{c=1}^{n_c} \mathbf{F}_{ic} + \mathbf{F}_i^{ext} \tag{3.36}$$

$$\mathbf{T}'_i = \sum_{c=1}^{n_c} (\mathbf{l}_i^c \times \mathbf{F}_i^c + \mathbf{q}_i^c) + \mathbf{T}_i^{ext} \tag{3.37}$$

It can be seen from equations 3.32 3.35. that both the non-viscous and viscous damping terms are opposite to the velocity and the difference lays in the evaluation of the damping force.A quasi-static state of equilibrium for the assembly of particles can be achieved by application of adequate damping. The damping applied in such problems should be high enough to obtain a non-oscillatory overall response. This is necessary to dissipate kinetic energy of the assembly of particles.

3.2.5 Time Integration Scheme

Time integration operator for the translational motion at the n-th time step is as follows:

$$\ddot{\mathbf{u}}_i^n = \frac{\mathbf{F}_i^n}{m_i} \tag{3.38}$$

$$\dot{\mathbf{u}}_i^{n+\frac{1}{2}} = \dot{\mathbf{u}}_i^{n-\frac{1}{2}} + \ddot{\mathbf{u}}_i^n \triangle t \tag{3.39}$$

$$\mathbf{u}_i^{n+1} = \mathbf{u}_i^n + \dot{\mathbf{u}}_i^{n+\frac{1}{2}} \triangle t \tag{3.40}$$

The first two steps in the integration scheme for rotational motion are identical to those givens by Equations 3.38 and 3.39:

$$\dot{\omega}_i^n = \frac{\mathbf{T}_i^n}{I_i} \tag{3.41}$$

$$\omega_i^{n+\frac{1}{2}} = \omega_i^{n-\frac{1}{2}} + \dot{\omega}_i^n \triangle t \tag{3.42}$$

For the rotational plane (2D) motion the rotation angle θ_i can be obtained similarly as the displacement vector u_i :

$$\theta_i^{n+1} = \theta_i^n + \omega_i^{n+\frac{1}{2}} \triangle t \tag{3.43}$$

The vector of incremental rotation is obtained as :

$$\triangle\theta_i = \omega_i^{n+\frac{1}{2}} \triangle t \tag{3.44}$$

It must be remarked that knowledge of the rotational configuration is not always necessary. If tangential forces are calculated incrementally, then knowledge of the vector of incremental rotation $\triangle\theta$ is sufficient. This saves considerable computational cost of the time integration scheme.

3.2.6 Numerical Stability

Explicit integration in time yields high computational efficiency and it enables the solution of large models. The known disadvantage of the explicit integration scheme is its conditional numerical stability imposing the limitation on the time step $\triangle t$:

$$\triangle t \leq \triangle t_{cr} \tag{3.45}$$

where $\triangle t_{cr}$ is a critical time step determined by the highest natural frequency of the system ω_{max}

$$\triangle t_{cr} = \frac{2}{\omega_{max}} \tag{3.46}$$

If damping exists, the critical time increment is given by

$$\triangle t_{cr} = \frac{2}{\omega_{max}}\left(\sqrt{1+\xi^2} - \xi\right) \tag{3.47}$$

where $\xi$is the fraction of the critical damping corresponding to the highest frequency ω_{max}. Exact calculation of the highest frequency ω_{max} would require solution of the eigenvalue problem defined for the whole system of connected rigid particles.

In the algorithm implemented an approximate solution procedure is employed. An eigenvalue problem can be defined separately for every rigid particle . The maximum frequency is estimated as the maximum of natural frequencies of massspring

systems defined for all the particles with one translational and one rotational degree of freedom.

$$\omega_{max} = max\omega_i \tag{3.48}$$

and the natural frequency for each mass-spring system (contact) is defined as

$$\omega_i = \sqrt{\frac{k}{m_i}} \tag{3.49}$$

with k the spring stiffness and m_i the mass of particle i. Now it is possible to rewrite the critical time step as:

$$\triangle t_{cr} = min2\sqrt{\frac{m_i}{k}} \tag{3.50}$$

The effective time step is considered as a fraction of the critical time step

$$\triangle t = \beta \triangle t_{cr} \tag{3.51}$$

with

$$0 \leq \beta \leq 1 \tag{3.52}$$

3.2.7 DEM Algorithms

"The DEM Architecture contain two main component of classes ,The data component and functional component.The former only store data with out providing functionality while the latter define functions operating on the data.A simulation in DEM contain both the data and function component .In function component the following sequence of actions run repeatedly .Each of the actions are called engines ,these sequences of engines is known as *simulation loop*" (Šmilauer *et al.* n.d.) Refer appendix c for Architecture overview.

- Reset forces acting on bodies .
- Approximate and real contact detection.
- Determine physical properties of interactions.

- Calculate forces acting on particles based on the constitutive laws used.
- Update total forces acting on particles.
- Update particles positions.

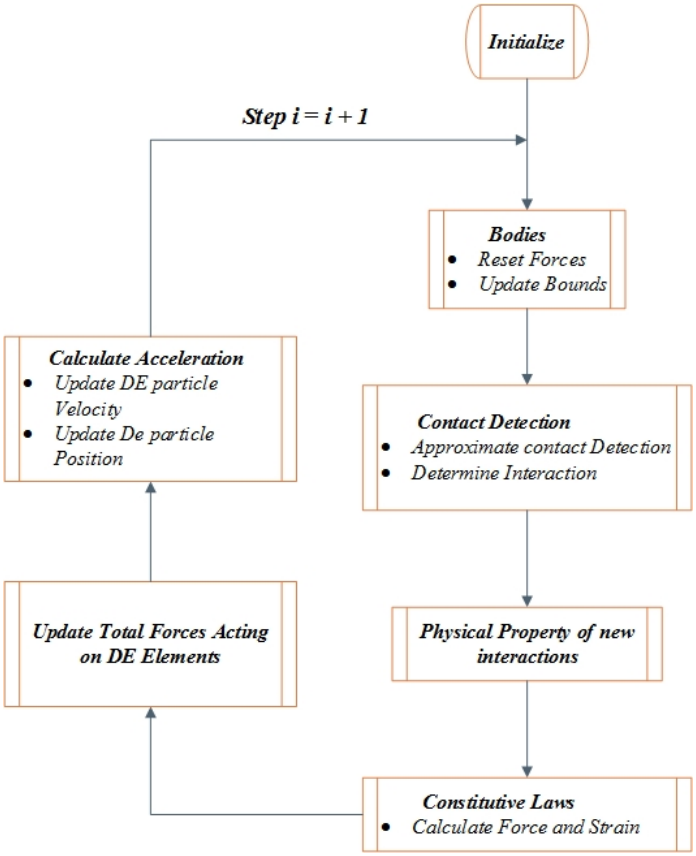


Figure 3.4: Flow chart for Discrete element method

3.3 The Finite Element Method

As shown in the previous section, the DEM is a good alternative to continuum based methods like Finite Element Methods for the simulation of problems characterized by discontinuities and material failure dominated by fracture.However the method is in short supply to characterize continuum domains.The Finite Element Method is a well equipped numerical method to tackle problems which involve continuum media. Hence in this section the formulation of the finite element method is summarized and the Algorithm to implement in the simulation is developed.

3.3.1 Non-Linear Transient Dynamic Formulation

In the non-linear transient dynamic formulation the equation of motion for a solid material can be written in as:

$$-\rho\frac{\partial^2 u_i}{\partial t^2} + \frac{\partial^2 \sigma_{ij}}{\partial x^2} + b_i = 0 \tag{3.53}$$

where ρ is density, t is time, σ_{ij} are the stress and bi are the body forces. Equation (3.53) is completed with the boundary conditions in displacements u_i and the equilibrium in surface traction .

$$u_i - \tilde{u_i} = 0 on \Gamma_u \tag{3.54}$$

$$\sigma_{ij} - \tilde{t_i} = 0 on \Gamma_t \tag{3.55}$$

In the above $\tilde{u_i}$and $\tilde{t_i}$ are prescribed displacements and traction over the boundaries Γ_u and Γ_t, respectively.

Employing a standard split of stresses into deviatoric and volumetric (pressure) parts, s_{ij} and p, respectively

$$\sigma_{ij} = s_{ij} = p\delta_{ij} \tag{3.56}$$

where δ_{ij} is the Kronecker delta function.

The linear elastic constitutive equations for the deviatoric stress s_{ij} are written as

$$s_{ij} = 2G(\varepsilon_{ij} - \frac{1}{3}\varepsilon_v\delta_{ij}) \tag{3.57}$$

where G is the shear modulus

$$\varepsilon_{ij} = \frac{1}{2}(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}) and \varepsilon_v = \varepsilon_{ij} \tag{3.58}$$

Note that the Einstein's summation convention is used. The pressure equation is written as

$$\frac{p}{K} - \frac{\partial u_i}{\partial x_i} = 0 \tag{3.59}$$

where K is the bulk modulus.

The governing equations of the mixed displacement-pressure formulation can be obtained then in the form

$$-\rho\frac{\triangle u_i}{\triangle t}+\frac{\partial s_{ij}}{\partial x_j}+\frac{\partial p}{\partial x_i}+b_i=0 \tag{3.60a}$$

$$\frac{\triangle u_i}{\triangle t}-u_i^{n+\frac{1}{2}}=0 \tag{3.60b}$$

$$\frac{\triangle p}{K}-\frac{\partial(\triangle u_i)}{\partial x_i}=0 \tag{3.60c}$$

where $\triangle u_i = u_i^{t+\frac{1}{2}}, \triangle u_i = u_i^{t+1}-u_i^{t+1}, \triangle p = p^{t+1}-p^t$ are the increments of velocity, displacement and pressure, respectively

By Introducing the concept of orthogonal sub-scale approach developed by (Codina 2000) (Codina and Blasco 2000) Equation 3.53 is split now as

$$\pi_i+\frac{\partial p}{\partial x_i}=0 \tag{3.61}$$

$$\pi_i=-\rho\frac{\partial u_i}{\partial t}+\frac{\partial s_{ij}}{\partial x_i}+b_i \tag{3.62}$$

The term π_i is the part of equation 3.53not containing the pressure gradient and may be considered as the negative of a projection of the pressure gradient. In a discrete setting, the term π_i can be considered belonging to a sub-scale space orthogonal to that of the pressure gradient terms.

The weighted residual form of the governing equations can be written in the form

$$\int_\Omega \delta u_i \rho\frac{\triangle u_i}{\triangle t}d\Omega+\int_\Omega \delta\epsilon_{ij}\sigma_{ij}d\Omega-\int_\Omega \delta u_i b_i d\Omega-\int_{\Gamma_t}\delta u_i\tilde{t_i}d\Gamma_t \tag{3.63}$$

$$\int_\Omega \delta u_i[\frac{\triangle u_i}{\triangle t}-u_i]d\Omega=0 \tag{3.64}$$

$$\int_\Omega q(\frac{\triangle p}{K}-\frac{\partial(\triangle u_i)}{\partial x_i})d\Omega+\int_\Omega\frac{\partial q}{\partial x_i}(\frac{\partial p}{\partial x_i}+\pi_i)=0 \tag{3.65}$$

$$\int_\Omega \omega(\pi_i+\frac{\partial p}{\partial x_i})d\Omega=0 \tag{3.66}$$

3.3.2 Finite Element Discretization

We chooseC^0 continuous linear interpolations of the displacements, pressure and pressure gradient projection π_i over three-node triangles (2D) and four-node tetrahedron (3D). The linear interpolations are written as :

$$u_i = \sum_{j=1}^{n} \mathbf{N}_j \bar{u_i}^j \tag{3.67}$$

$$p = \sum_{j=1}^{n} \mathbf{N}_j \bar{p}^j \tag{3.68}$$

$$\pi_i = \sum_{j=1}^{n} \mathbf{N}_j \bar{\pi_i}^j \tag{3.69}$$

where n = 3 for 2D 4 for 3D problems and $\tilde{\cdot}$ denotes nodal variables. N_j are the linear shape functions

Substituting the approximations into Equations 3.63 and 3.66 gives the following system of discretized equations:

$$\mathbf{M}\frac{\triangle\tilde{\mathbf{v}}}{\triangle t} - R_u + f_d = 0 \tag{3.70}$$

$$\frac{\triangle\tilde{\mathbf{u}}}{\triangle t} - \tilde{\mathbf{v}} = 0 \tag{3.71}$$

$$\mathbf{C}^T\triangle\tilde{\mathbf{u}} - M_p\triangle\tilde{\mathbf{p}} - L\tilde{\mathbf{p}} - Q\tilde{\pi} = 0 \tag{3.72}$$

$$\mathbf{Q}^T\tilde{\mathbf{p}} + \tilde{\mathbf{G}}\tilde{\pi} = 0 \tag{3.73}$$

where the element contributions are given by

$$\mathbf{L} = \int_\Omega (\nabla\mathbf{N})^T\tau\nabla\mathbf{N}d\Omega, \tilde{\mathbf{G}} = \int_\Omega \mathbf{N}^T\tau\mathbf{N}d\Omega, f_d = \int_\Omega \mathbf{B}^T\sigma d\Omega, \mathbf{Q} = \int_\Omega (\nabla\mathbf{N})^T\tau\mathbf{N}d\Omega \tag{3.74}$$

where $\mathbf{B}$ is the linear stress-strain operator matrix, and

$$\tau = \begin{bmatrix} \tau_1 & 0 & 0 \\ 0 & \tau_2 & 0 \\ 0 & 0 & \tau_3 \end{bmatrix} \tag{3.75}$$

The matrices M, M_p, C, R_v are defined as :

$$\mathbf{M} = \int_\Omega \rho \mathbf{N}_v^T \mathbf{N}_v d\Omega, \mathbf{M}_p = \int_\Omega \frac{1}{\mathbf{K}} N_P^T N_p d\Omega, R_v = \int_\Gamma \mathbf{N}_p^T \hat{t} d\Gamma + \int_\Omega \mathbf{N}_p^T \mathbf{b} d\Omega, \mathbf{C} = \int_\Omega \mathbf{N}_v^T \nabla \mathbf{N}_p d\Omega \tag{3.76}$$

In the case of explicit solution, the two matrices **M and Mp** are usually diagonalized. In the discretization procedure the same interpolation has been assumed for all the discretized fields: $N_v = N_p = N_\pi = N$.

We can then define a four steps semi-implicit time integration algorithm as follows:

Step 1 Compute the nodal velocities $\tilde{\mathbf{v}}^{n+\frac{1}{2}}$

$$\tilde{\mathbf{v}}^{n+\frac{1}{2}} = \tilde{\mathbf{v}}^{n-\frac{1}{2}} + \triangle t \mathbf{M}^{-1}(\mathbf{R}_v^n - \mathbf{f}_d^n) \tag{3.77}$$

Step 2 Compute the nodal displacements $\tilde{\mathbf{u}}^{n+1}$

$$\tilde{\mathbf{u}}^{n+1} = \tilde{\mathbf{u}}^{n-1} + \triangle t \mathbf{V}^{n+\bar{\frac{1}{2}}} \tag{3.78}$$

Step 3 Compute the nodal pressures $\tilde{\mathbf{p}}^{n+1}$

$$\tilde{\mathbf{p}}^{n+1} = [\mathbf{M}_p - \mathbf{L}]^- 1[\triangle t \mathbf{C}^T]\mathbf{V}^{n+\bar{\frac{1}{2}}} + \mathbf{M}_p \tilde{p}^n - \mathbf{Q}\tilde{\pi}^n \tag{3.79}$$

Step 4 Compute the nodal projected pressure gradients $\tilde{\pi}^{n+1}$

$$\tilde{\pi}^{n+1} = -\mathbf{G}_{-1}\mathbf{Q}^T \tilde{p}^{n+1} \tag{3.80}$$

In above matrices,$M, Mp, L, C, Q, andG$ are evaluated at t^{n+1}. Note that the steps (1), (2) and (4) are fully explicit. A fully explicit algorithm can be obtained by computing $\tilde{p}^{n+1}$ from step (3) in Equation (**??**) as follow:

$$\tilde{\mathbf{p}}^{n+1} = \mathbf{M}_p^{-1}[\triangle t \mathbf{C}^T \tilde{\mathbf{v}}^{n+\frac{1}{2}} + (\mathbf{M}^p - \mathbf{L})\tilde{p}^n - \mathbf{Q}\tilde{\pi}^n] \tag{3.81}$$

Obviously, the explicit solution is efficient if diagonal form of the matrices $Mp, Mand\tilde{G}$ a are used. The explicit solution is possible if certain compressibility is assumed, i.e.$K \neq 0$. If $K \longrightarrow \infty$ pressure must be obtained using the implicit scheme given by Equation (3.79)

3.3.3 FEM Algorithms

FE simulation consists of the following engines:

- Reset forces acting on FE nodes.
- Determine physical properties of FE elements (ex: nodal masses, element stiffness and restraints).
- Update geometrical properties of FE elements.
- Determine external loads acting on FE nodes.
- Determine FE critical time-step.
- Set FE time-step equal to n times (n ¿= 1) DE time-step (for coupled DE/FE problems):
- Determine Damping coefficient.
- Check convergence.
- Calculate stresses and strains and at Gauss points and update element loads.
- Calculate nodal velocities and update nodal positions
- Record outputs

3.4 Coupled Discrete-Finite Element Formulation

In most case the failure of the material in DEM modeling is localized at the portion of the DE domain.The Remaining domain can be represented as continuum domain and can be treated with the finite element method.This section presents the theoretical formulation and the numerical implementation of a procedure for coupling the DEM

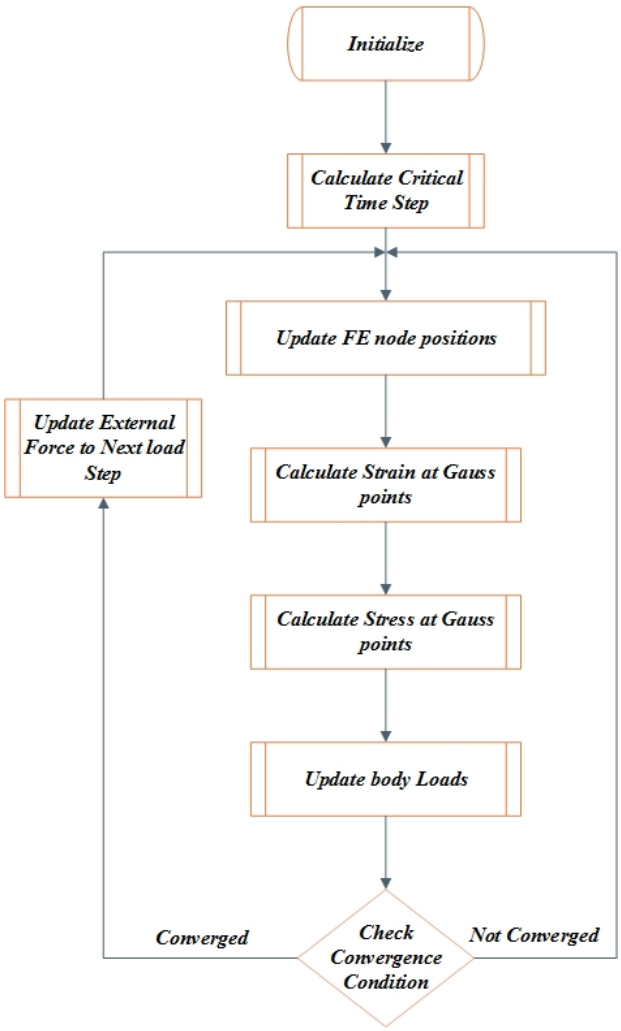


Figure 3.5: Flow chart for Finite element simulation

with the FEM. The idea considers the use of the DEM in the region, or subdomain, where failure will be produced and the FEM in the rest of the domain. This allows taking advantage of the best functions of each method in each sub-domain and makes more efficient the numerical simulation.Finaly the algorithm for the coupling scheme is suggested.

The general algorithm for the transient dynamic problem involving discrete elements and finite elements includes the following steps (O nate and Rojek 2004)

Step 1 Compute nodal velocities

Discrete Element

$$\dot{u}_i^{n+\frac{1}{2}} = \dot{u}_i^{n-\frac{1}{2}} + \ddot{u}_i^n \triangle t \qquad with \qquad \ddot{u}_i^n = \frac{F_i^n}{m_i} \tag{3.82}$$

$$\omega_i^{n+\frac{1}{2}} = \omega_i^{n-\frac{1}{2}} + \dot{\omega}_i^n \triangle t \qquad with \qquad \dot{\omega}_i^n = \frac{T_i^n}{I_i} \tag{3.83}$$

Finite Element

$$\dot{u}_i^{n+\frac{1}{2}} = \dot{u}_i^{n-\frac{1}{2}} + \triangle t M_d^{-1}(f^n - g^n) \tag{3.84}$$

Step 2 Compute nodal Displacement

Discrete Element

$$u_i^{n+1} = u_i^n + \dot{u}^{n+\frac{1}{2}} \triangle t \tag{3.85}$$

$$\triangle \theta_i^{n+1} = \omega_i^{n+\frac{1}{2}} \triangle t \tag{3.86}$$

Finite Element

$$u_i^{n+1} = u_i^n + \dot{u}^{n+\frac{1}{2}} \triangle t \tag{3.87}$$

Step 3 Compute nodal pressure

Finite Element

$$\tilde{p}^{n+1} = [C + L - S]^{-1}[\triangle t G^T \dot{\tilde{u}}^{n+\frac{1}{2}} + C\tilde{p}^n] \tag{3.88}$$

Step 4 update the nodal coordinate

Discrete and Finite Element

$$x_i^{n+1} = x_i^n + \triangle \tilde{u} = x_i^n + (\tilde{u}^{n+1} - \tilde{u}^n) \tag{3.89}$$

Step 5 update Residual Force vector

Go to Step 1

3.4.1 Coupled FEM-DEM Algorithms

This Algorithm is customized version of the above steps for implementation in YADE

Engines which handles the interaction between DE particles and interface elements consist of :

- Determine bounding volume of interfaces:

- Determine physical properties of interfaces:
- Update geometrical properties of interfaces:
- Determine potential interactions between DE particles and interfaces:

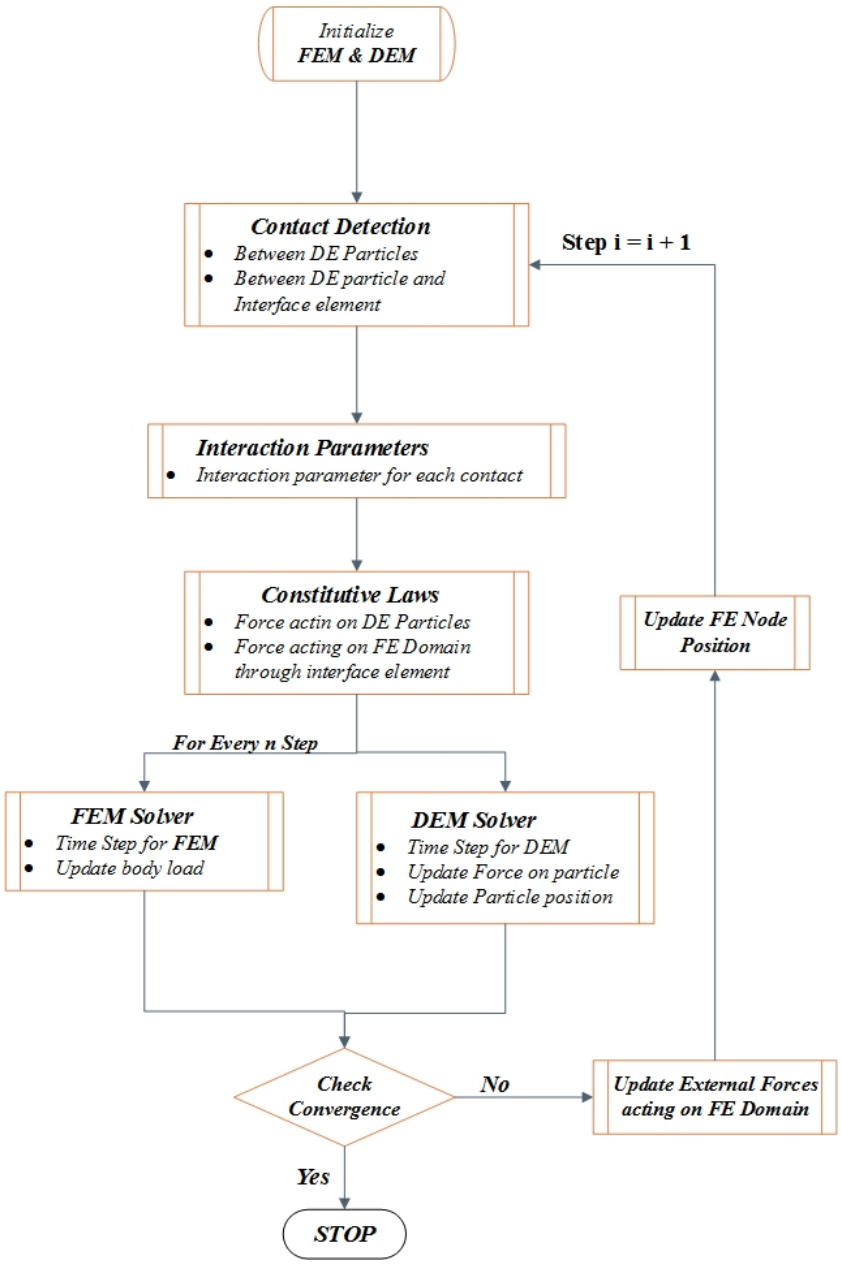


Figure 3.6: Flow chart for FE-DE simulation

3.5 Programming Approach

Now that we are equipped with the numerical methods and their algorithms ,the next step is to implement or coded them through a programming languages .In this stage selecting an effective programming approach and languages is a crucial step .

Basically there are two types of computer programming approaches ,procedural and object oriented programming.*Procedural Oriented Programming* (**POP**)is derived from structural programming based on the concepts of **functions**,**procedure**,**routines**. It is easy to access and change the data in procedural oriented programming. On the other hand, *Object Oriented Programming* **(OOP)** allows decomposition of a problem into a number of units called objects and then build the data and functions around these objects. It emphasis more on the data than procedure or functions. Table (3.1) summarizes the difference between POP and OOP .

3.5.1 Object-Oriented Programming

Object Oriented means directed towards objects. In other words, it means functionally directed towards modeling objects. This is one of the many techniques used for modeling complex systems by describing a collection of interacting objects via their data and behavior.

Object Oriented programming (OOP), is a way of programming that focuses on using *objects* and *classes* to design and build applications. Major pillars of Object Oriented Programming (OOP) are **Inheritance**, **Polymorphism**, **Abstraction**, and **Encapsulation**.Object Oriented Analysis(OOA) is the process of examining a problem, system or task and identifying the objects and interactions between them.

OOP offers the following advantages:

- Provides a clear program structure, which makes it easy to map real world problems and their solutions.
- Facilitates easy maintenance and modification of existing code.
- Enhances program modularity because each object exists independently and new features can be added easily without disturbing the existing ones

	Procedural Oriented Programming	**Object Oriented Programming**
Based on	Focuses on data and functions	Is based on real world scenarios .whole program is divided into small parts called **Object**
Re-usability	limited code Reuse	Code Reuse
Approach	Top down approach	Object focus design
Access Specifiers	Not Any	public,private and protected
Data movement	Data can move freely from function to function in the system	objects can move and communicate with each other through members function
Data Access	Most function uses global data for sharing that can be accessed freely with in the system	Data cannot move freely from method to method ,it can be kept in public or private so we can control the access of data
Data hiding	Specific way to hide data so its less secure	Provides data hiding its more secure
Overloading	Not possible	Method and operator overloading
Example languages	C,VB,FORTRAN,Pascal	C++,python,Java
Abstraction	use abstraction at procedure level	Use abstraction at Class and object level

Table 3.1: Difference between object oriented and procedural oriented programming languages

- Presents a good framework for code libraries where supplied components can be easily adapted and modified by the programmer.
- Imparts code re-usability
- Encourages a clean and pragmatic way to programming and enables writing bigger and complex programs.

3.5.2 Object-oriented Python and C++

Object Oriented Programming (OOP) is based on the concept of ***objects*** and***data***. In order for a programming language to be object oriented, it should have a way to entertain working with ***classes*** and ***objects***.The language should have a means to implement the fundamental object-oriented principles such as ***inheritance***, ***abstraction***, ***encapsulation*** and ***polymorphism***.

Python and C++ are programming languages the are designed with an object oriented approach. The fundamental principles of OOP are integrated in these languages .Hence this research uses these languages for coding the algorithms stated in the above sections and harvest the merits of OOP.

3.6 Input Parameters

The input parameters that are necessary for the numerical simulation and validation are shown in table (3.2)

Types of elements	**Parameter**
Finite Element	• Young modulus (**E** MPa) • Poissons Ratio (ν)
Interface Element	• Material Modulus (**E** MPa) • Ratio ($\frac{K_T}{K_N}$) • Coefficient of friction ($\tan\phi$)
Discrete Element	• Density ($\frac{kg}{m^3}$) • Material Modulus (**E** MPa) • Ratio ($\frac{K_T}{K_N}$) • Coefficient of friction ($\tan\phi$) • Rolling resistance coefficient (β_r) • Dimensionless coefficient (η_r) • Damping coefficient

Table 3.2: Input Parameters

CHAPTER 4

MICROSCOPIC INSIGHT TO DIRECT SHEAR TEST AN APPLICATION USING OPEN-SOURCE PLATFORM YADE

To determine the input parameters for the numerical model, calibration is first conducted using the results of direct shear tests. Numerical simulations of direct shear tests are performed and microscopic parameters for the DE simulation are identified through parametric study and by comparing the numerical results with the experimental data .The numerical simulation and the model calibrations are based on (GUO 2014) ,(Tran 2013). YADE (Šmilauer *et al.* n.d.) and esys-particle (Weatherley *et al.* n.d.) is used as platform .

4.1 Physical Direct Shear Test

The Direct shear test is the conventional laboratory test used by geotechnical engineers to measure the shear strength of soils. A schematic diagram of the apparatus and the shearing action is demonstrated in Figure 6.1.The essential feature of the apparatus is a rectangular box divided horizontally into two halves and containing a rectangular prism of soil. While the prism is subjected to a constant vertical compressive force, an increasing horizontal force is applied to the upper half of the box, thus causing the prism to swear along the dividing plane of the box. The test is normally carried on a number of identical specimens using different vertical

stresses so that a graph of shearing resistance against vertical stress can be plotted. The change in volume, density and porosity is also measured by tracing the vertical movement of the top platen.

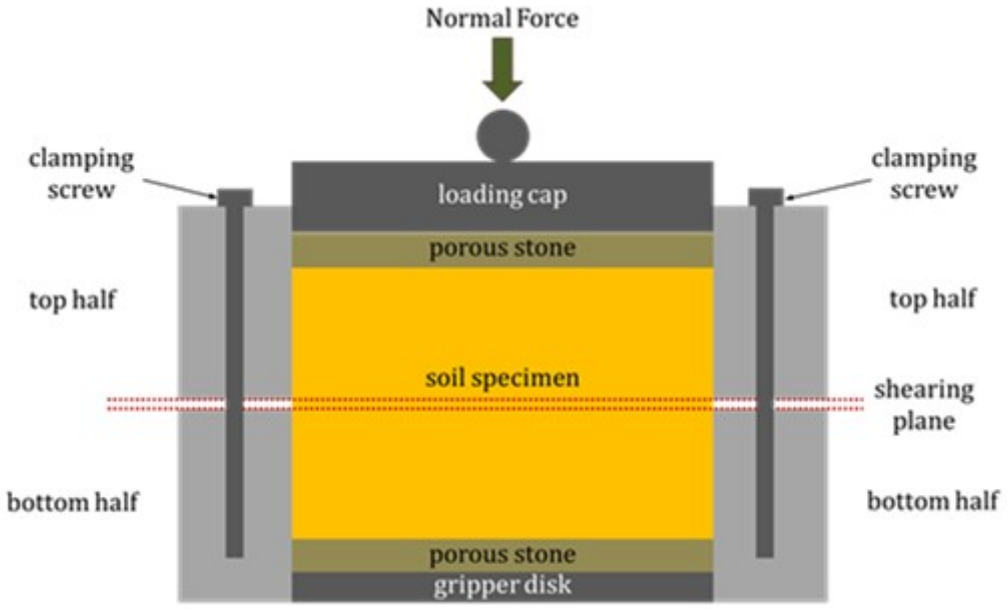


Figure 4.1: Schematic diagram of shear box

The direct shear test apparatus used in the current study comprises of a shear box (60 mm x 60 mm) split horizontally into two halves.To apply direct shear to the sample, one part of the box was moved with a constant velocity controlled by servo controller while the other part was kept stationary. Three different normal stresses, 50 K Pa, 100 K Pa and 200 K Pa were used in this study using vertical loads applied on top of the shear box. The initial sample height was about 25 mm with the height to width ratio 1: 2.4.

4.2 3D Implementation of Direct Shear Test (DST) in YADE

The first step in the simulation of DST is to create a DE soil that replicate the real soil sample used in the laboratory test . Although there are several available methods that can be used to construct DE specimens,no agreement has been reached regarding the most suitable approach to generate soil specimens.The simulator have to adopt methods that provide best replication of the real packing process while

keeping the computational cost acceptable .Based on replication of the real packing (Actual packing) and computational cost the following techniques are often used by researchers .

4.2.1 DEM Soil Sample Generation Methods

- *Gravity Method* : The DE specimens are generated in layers and Gravity is applied to the particles for the packing process. The number of layer and volume of particle for each layer is calculated Based on the target void ratio of the final soil specimen.

- *Compression Method* : This packing method is proposed by (Cundall and Strack 1979b)) it uses pressurized boundaries to maintain equilibrium condition which violate the initial condition in real granular soil (sand) deposition .

- *Triangulation Based Approach* :This method uses the concept of triangulation, and employs Triangle in 2D and Tetrahedral in 3D to generate and pack the particles. The method uses simple algorithm and efficient in terms of computational cost for the preparation of preliminary particle assembly(Labra *et al.* 2012).However it lucks control of particle size distribution which is very crucial to replicate real soil behavior.

- *Radius Expansion Method* : This approach tends to generate DE specimens with *Isotropic* Stress State (Barreto *et al.* 2012) .

The gravitational approach by (Jiang, Konrad, *et al.* 2003) appears to be a suitable Method since it mimics the natural layer-by-layer depositing process of sands.Hence this thesis uses this packing method by accounting computational cost and replication of real soil deposition .

Multi-Layer Under-compaction Packing Process (based on (Jiang, Konrad, *et al.* 2003)

The gravitational packing technique used in this study is a multi-layer packing method. This packing technique originated from the one proposed by Ladd (1978)

for real specimen preparation and is similar to the Multi-layer with Undercompaction Method proposed by (Jiang, Konrad, *et al.* 2003). It follow the subsequent packing procedure:

Based on the required porosity of the final soil sample, initially the number of layer and the volume of particle is estimated. The first layer of non-contacting particle which is called clouds of particles is then generated through accounting the particle size distribution. To avoid particle overlapping, the height of the box have to be larger than the target height. Then Gravity is applied to each generated particles hence they experience free fall and come in contact with each other.. To increase the density of the packing, the particles are compressed using servo mechanism which helps smaller particles to move into voids between larger particles. When the system reaches equilibrium (i.e. the unbalanced force is less than a value 0.01),The first layer generation will be completed . For the second layer, the height of the box is increased and the second cloud of non- contacting particles is generated in the area above the existing particles. Then the same procedure is followed until the final sample is formed. The proposed Multi-layer approach helps increase the density of the packing while keeping the packing pattern realistic

Computational Cost

Along with the fabric structure of the sample, particle size distribution also have effect on the characteristics of DE specimens. It is essential that particle generation follows the predefined particle size distribution which has a great influence on the behavior of the discrete element system. However, the true replication of grain size is usually restricted by the high computational cost caused by the large number of particles. Since the volume of a particle with radius r is proportional to r^3, a large number of small particles are needed to generate a very small volume. This leads to an extremely high number of particles required to fill up the soil domain which in turn radically increases the simulation cost. In addition, the high computational cost is also caused by the decrease in critical time step needed for stable analysis which is proportional to the mass of the particles. As a result , particles smaller than $D5$ (particle diameter corresponding to $5percent$ passing) are neglected in this study to reduce the computational cost.On the other hand in most simulation in geotechnical

Parameter	**Value**
Material Modulus (**E** MPa)	100
Inter particle friction (ν)	0.5
Density ($\frac{kg}{m^3}$)	2655
Damping coefficient	0.7
Normal Contact Stiffness K_N	$2\mathbf{E}r$ where r is radius of particle
Tangential Contact Stiffness K_T	($\frac{K_N}{1.5}$)

Table 4.1: input parameters used for Multi-layer under compaction packing algorithm

engineering problems ,researchers use particle up-scaling to reduce computational cost.However the effect of the up-scaling should be carefully considered .In this study, the scale factors (ratio of a numerical particle size to its real particle size) are chosen as 4 from literature for the direct shear test .

Application of Packing Technique in YADE

The characteristics of a DE specimen depends on the particle shape ,size , packing structure, particle size distribution,contact model and other different parameters .Different techniques are available in YADE to generate DE soil sample that replicate the real soil.These packing Techniques are briefly described in Appendices C. Accordingly using multi-layer under compaction Algorithm ,DEM sample where generated with two distinct initial void ratio $e_{in} = 0.65$ and $e_{in} = 0.84$.The packing Algorithm mimics the natural layer by layer deposition process of sand. The algorithm allow us to generate uniform sample with target void ratio. The following algorithm (figure 4.2)is used to generate the sample in YADE.

An example of the generated packing sample is shown in figure 4.3a .To show the deformation during the shear process the sample is painted in different colors along x axis Figure 6.3 b.

Table 4.1 shows the parameter used for the packing algorithm .The python code for the packing algorithm is presented in appendices D

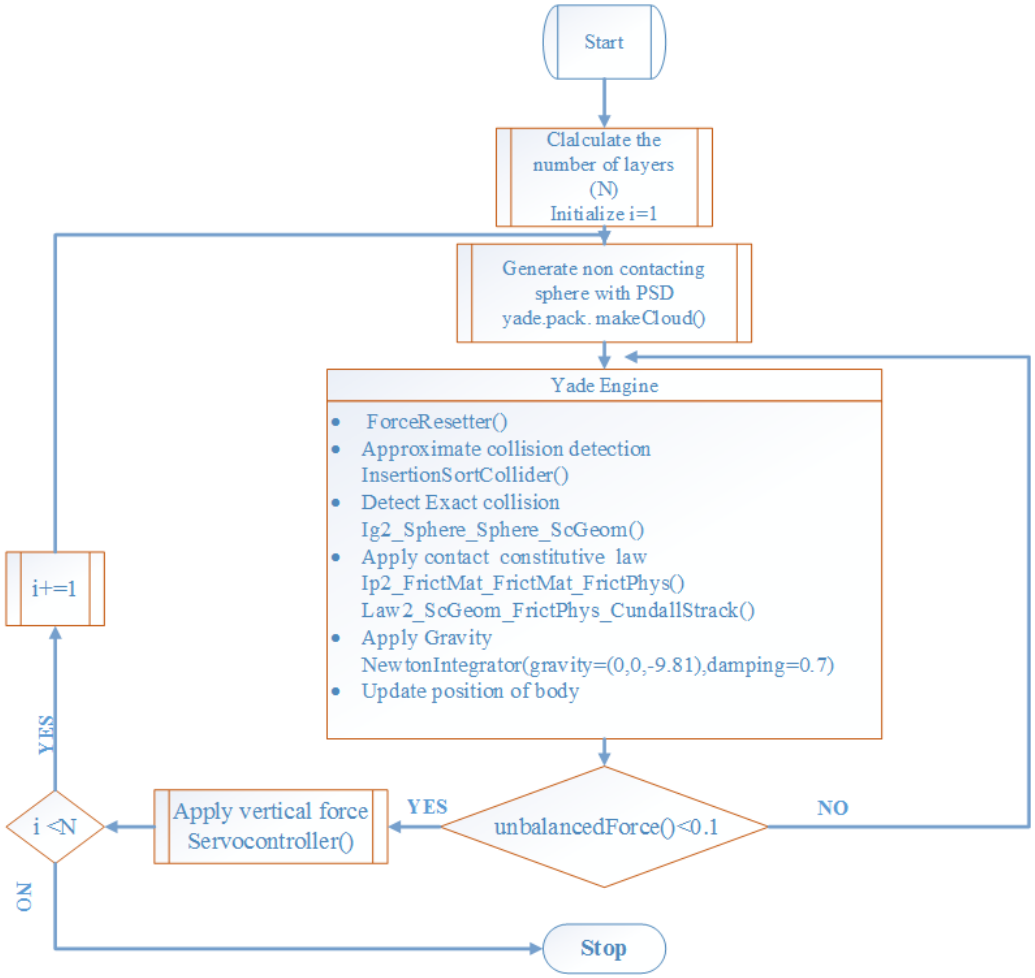
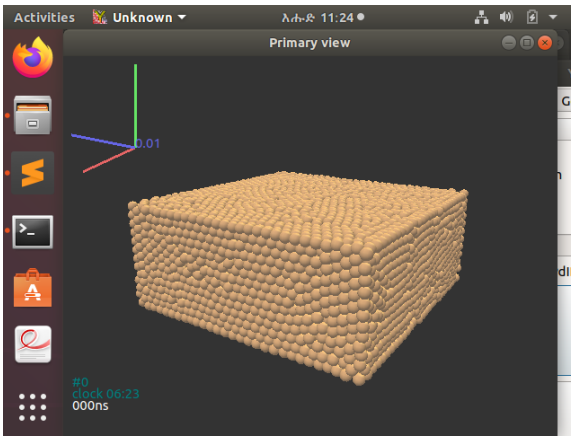


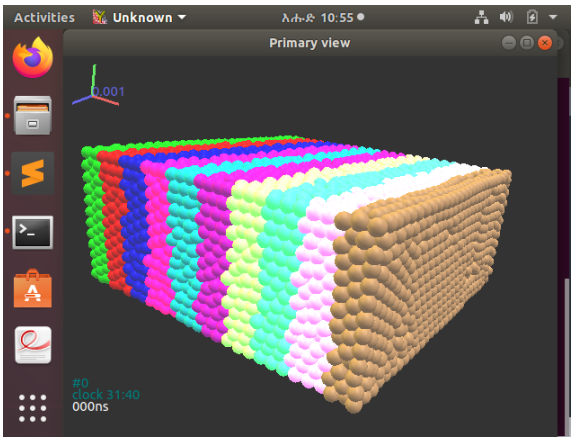
Figure 4.2: Multi-layer under compaction packing Algorithm for YADE

4.2.2 Simulation Scheme

The numerically simulated shear box consists of two parts and each part comprises 5 rigid boundaries: one horizontal boundary and four vertical boundaries .It has the same dimensions as the actual one to replicate the testing conditions A specimen is generated using the gravitational method illustrated in the previous Section.After the sample generation is completed, the specimen is subjected to three different vertical stresses of 50 kPa, 100 kPa and 200 kPa. And then sheared in the x-direction,to maintain a quasi-static condition,a servo controller is implemented in the python code (see appendis D). Figure 4.4 shows The Generated Shear boxes in YADE



(a) Generated packing sample



(b) Generated packing sample with color

Figure 4.3: The Generated soil sample in YADE

4.3 Simulation Results and Discussions

In This section , the output result of the numerical simulation is analyzed through comparing with laboratory DST . The deformation pattern, stress–strain relationship and volumetric change is analyzed as follows.

4.3.1 Deformation Pattern

The deformation pattern of the DEM sample observed in the simulation is shown in figure (4.5 a) The deformation is at the shear strain of 15 percent with $\sigma_v = 100kpa$ .The shear deformation is high along the a shear plane and it became less and less when we go to the upper and lower planes . The deformation pattern observed from laboratory test under the same shear strain is shown in figure (4.5b) the comparison between the numerical result and laboratory test shows similar deformation patterns.

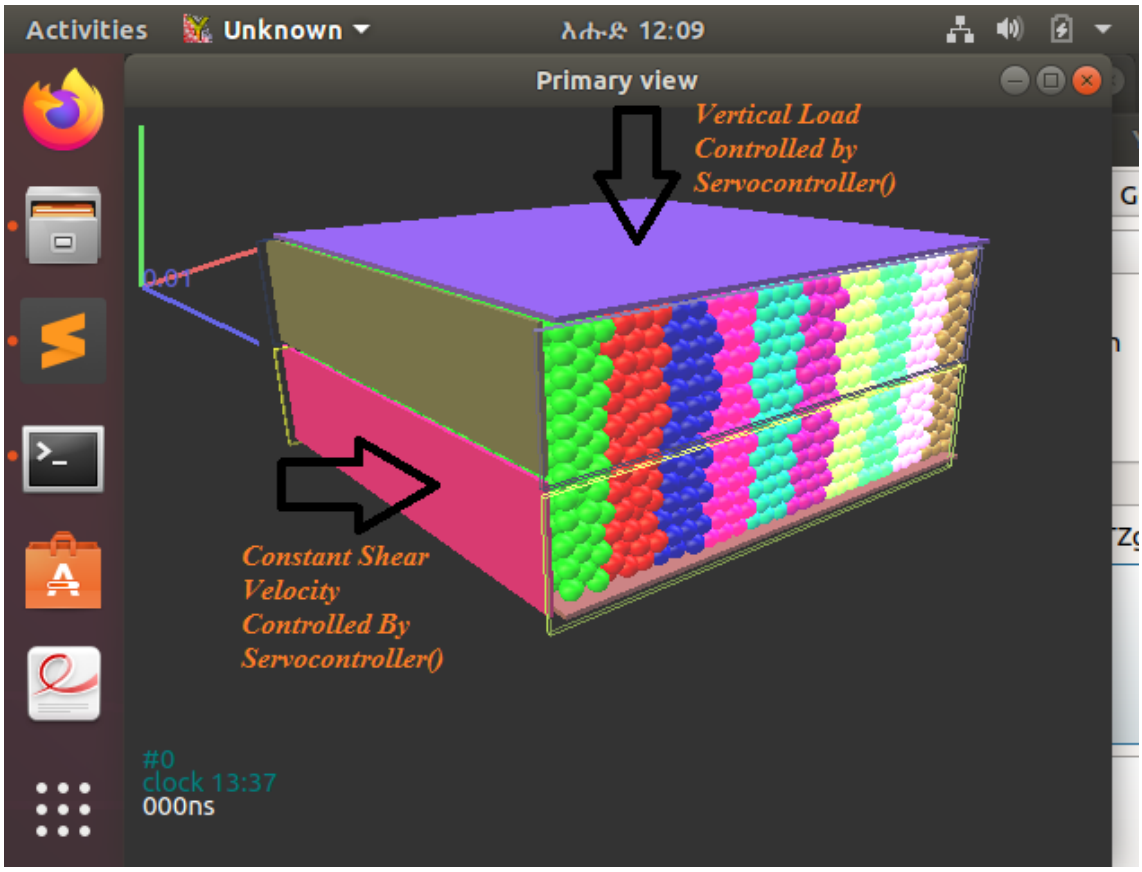
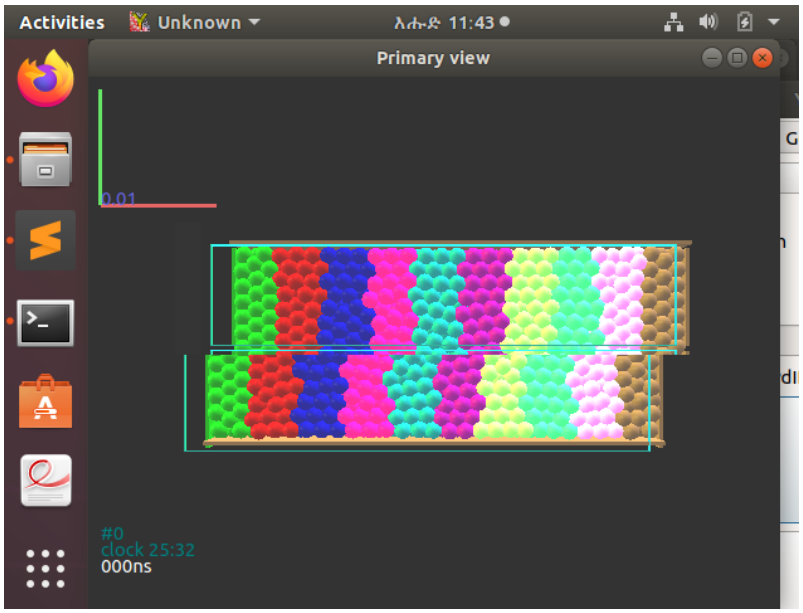


Figure 4.4: The Generated Shear box in YADE

This shows the shear deformation is localized along the shear plane. The soil near the shear plane reached their maximum shear strength while the soils far from the shear plane are not.In other word the soil at the shear plane reaches its critical state while the whole sample is not. This scenario of DST make it difficult to measure the critical state behavior of the granular material in the laboratory DST. Different scholars prefer Triaxial test to study the critical state behavior of the soil. This limitation of the laboratory DST can be resolved in the DEM simulation. The numerical simulation allow us to measure the critical state of the sample through dividing the sample to different zones. Behavior of the soil near the shear zone and far from the shear zone can be studied separately. Here the biggest challenge is the computational cost. There are different Algorithms and Methods that can applied to overcome the computational cost issues. We will discuss these methods in bellow.



(a) Deformation pattern from the Simulated DST

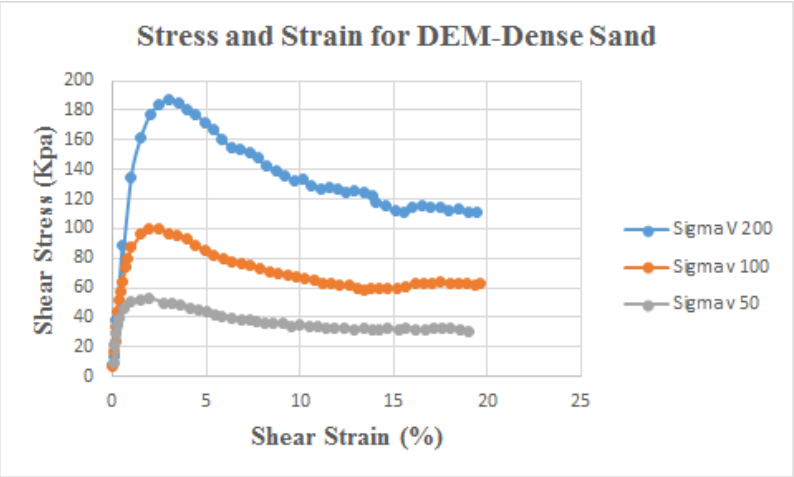
(b) Deformation Pattern from laboratory DST

Figure 4.5: Deformation pattern of DEM Direct Shear Test and Laboratory Test

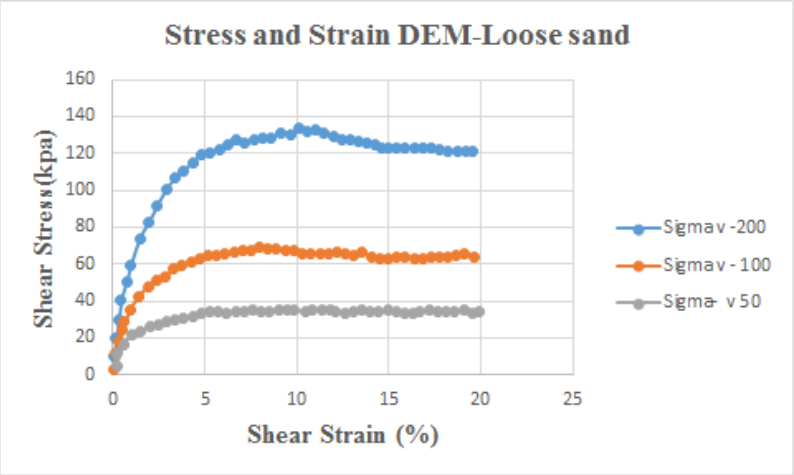
4.3.2 Stress Strain

Figure 4.6 a and 4.6 b presented the the stress–strain relationships obtained from DEM samples with initial void ratio =0.65 (Dense sand) and void ratio = 0.8 (Loose Sand) with vertical stress 200,100,50 Kpa . . The shear stress are obtained from the stress on and displacement of the boundary walls as in laboratory DST.the result shows the DEM assemblie with initial void ratio 0.65 exhibits strain softening while the sample with initial void ratio 0.8 shows strain hardening . A larger σ_v leads to larger shear stress until they reach to steady state .

In the steady state the shear stress of initial porosity 0.65 and 0.8 are the same under each σ_v refer Figure 4.7.



(a) Dense Sand



(b) Loose Sand

Figure 4.6: Stress and Strain relation ship of the numerical simulated DST

The shear stress vs vertical stress for dense and loose sand is presented in figure 4.8 a and 4.8 b respectively.

4.3.3 Volumetric Change

The change of void ratio e with $\sigma_v 200, 100, and 50 kpa$ obtained from the DEM Sample is shown in figure 4.9 a,4.9 b and 4.9 c respectively ,as shown in the figures the assemblies with $e_{ini} = 0.65$ exhibits volumetric dilation while the assemblies with $e_{ini} = 0.8$ shows volumetric contraction.

A larger σ_v leads to more volumetric change .In addition the void ratios of the loose sample $e_{ini} = 0.8$ are larger than that of the dense sample $e_{ini} = 0.65$ Refer figure (4.10). This is due to the deformation of the DEM sample in DST is highly

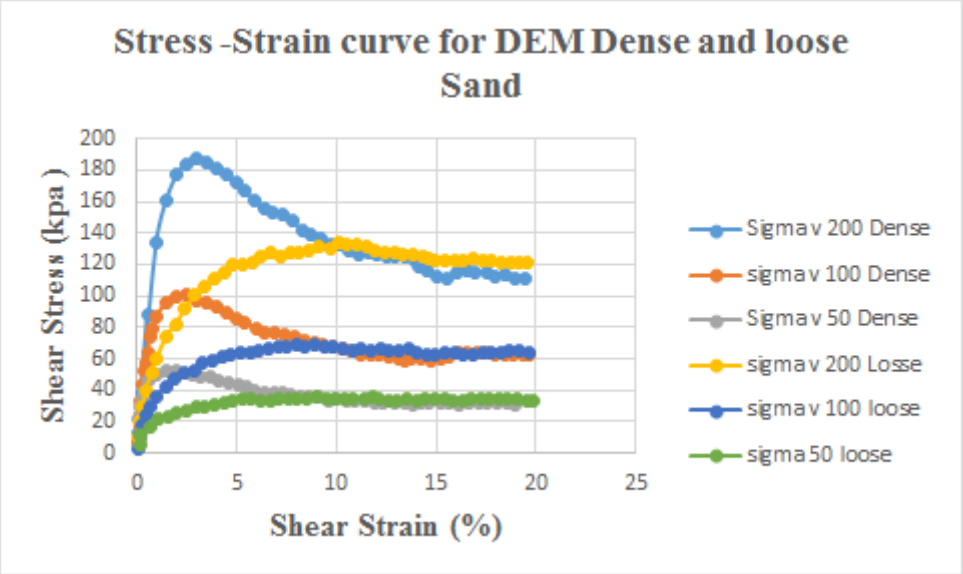


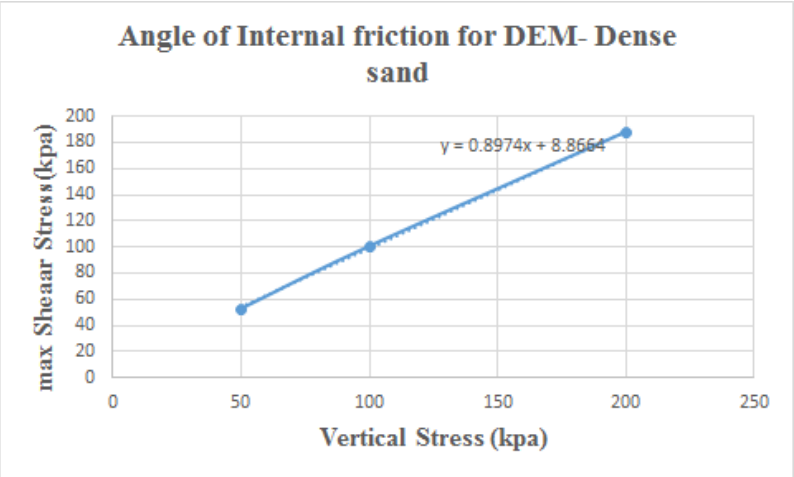
Figure 4.7: Stress and Strain relation ship of the numerical simulated DST in Dense and Loose Sand

non uniform.

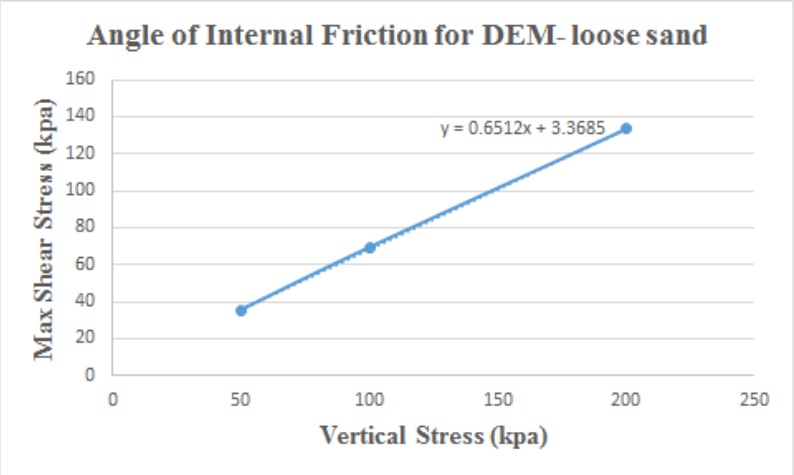
The highly non-uniform deformation as illustrated in figure 4.5 make it difficult to investigate critical state behavior of soil in laboratory direct shear test since the strain localization (local deformation) can't be measured in the localized zone.This behavior of soil had been widely explored by Triaxial compression tests. However in DEM simulation the limitation can be resolved through examining the evolution of e in the shear zone and the whole sample separately.

To monitor the behavior of the material ,Two concentric measuring spherical zones where sets in the center of the sample .The radius of the these zones where 0.45 h for the measuring sphere 1 and 0.20 h measuring sphere 2 where h is the height of the sample . Similar procedure where employed (Jiang, Liu, *et al.* 2018). Measuring sphere 1 represent the whole sample except soil near to the boundary walls. Measuring Sphere 2 used to capture the feature of the shear zone. See figure 4.11

In figure 4.12a ,4.12b,and 4.12C shows the void ration evolution under $\sigma_v = 200, 100, 50kpa$ with different e_{ini} respectively .The data is obtained from the measuring sphere 1 which represent the whole sample except soils having contact with the boundary wall of DST box refer figure 4.10.In this case the void ratio of the assemblies tends to reach different stable values under the same σ_v .



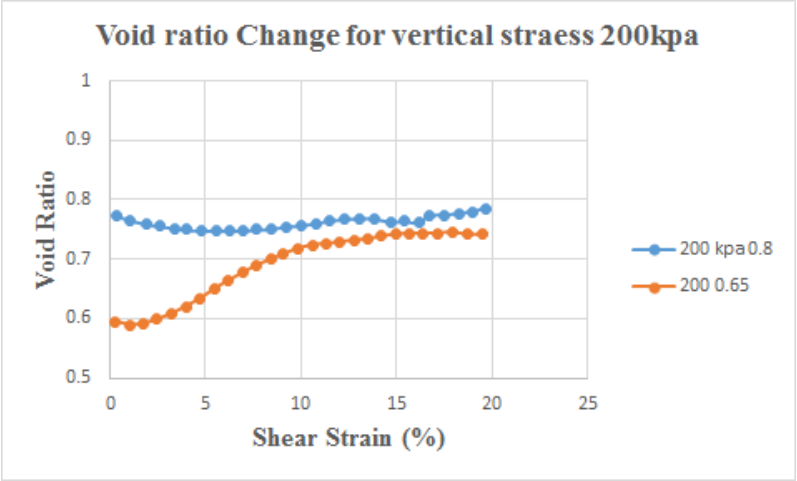
(a) Dense Sand $\phi = 42^o$



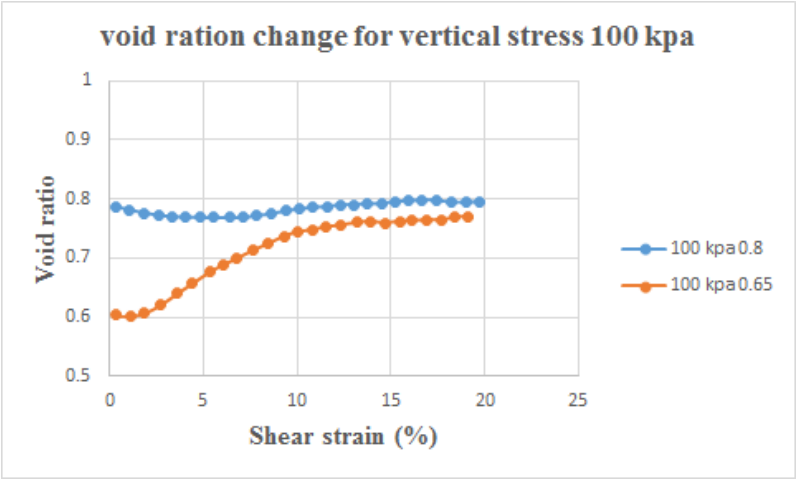
(b) Loose Sand $\phi = 33^o$

Figure 4.8: Shear Stress VS vertical Stress of the numerical simulated DST

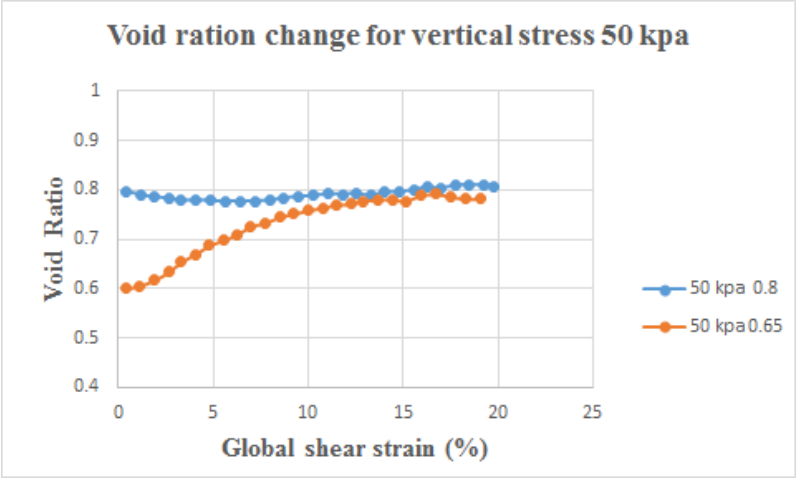
In figure 4.14a,4.14b,and 4.14c shows the void ration evolution under $\sigma_v = 200, 100, 50kpa$ with different e_{ini} respectively .The data is obtained from the measuring sphere 2 which represent the soil grains at the failure plane (shear Band)refer figure 6.15.In this case the void ratio under the same σ_v develops similar stable value after global shear strain of 15 percent.



(a) vertical stress ($\sigma_v = 200kpa$)



(b) vertical stress ($\sigma_v = 100kpa$)



(c) vertical stress ($\sigma_v = 50kpa$)

Figure 4.9: The evolution of void ratio under different σ_v

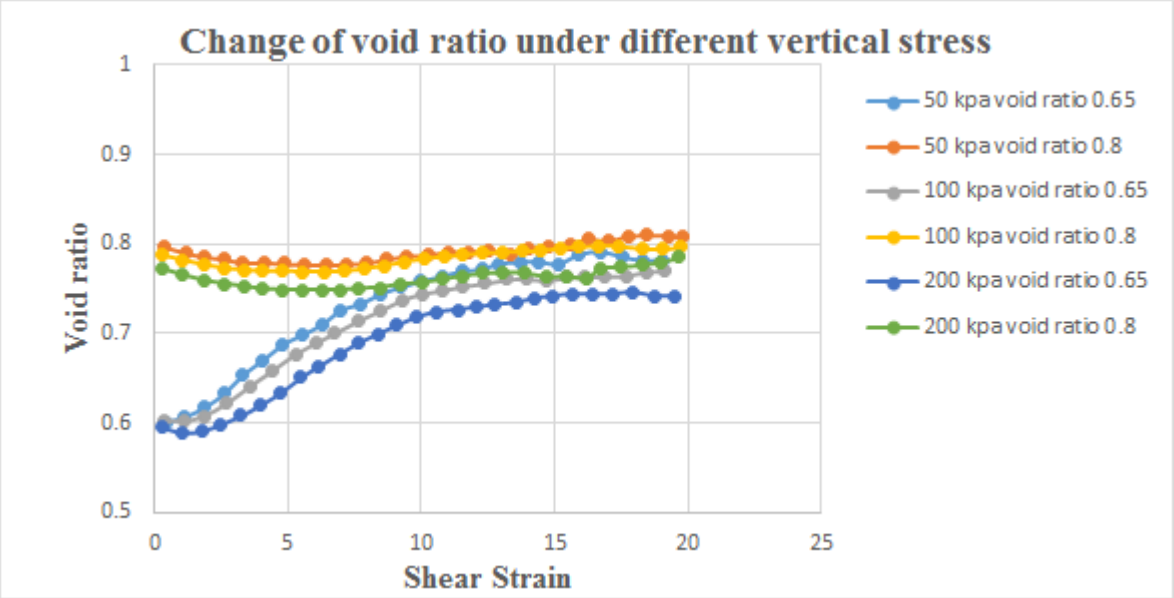


Figure 4.10: The evolution of void ratio of the numerical simulated DST

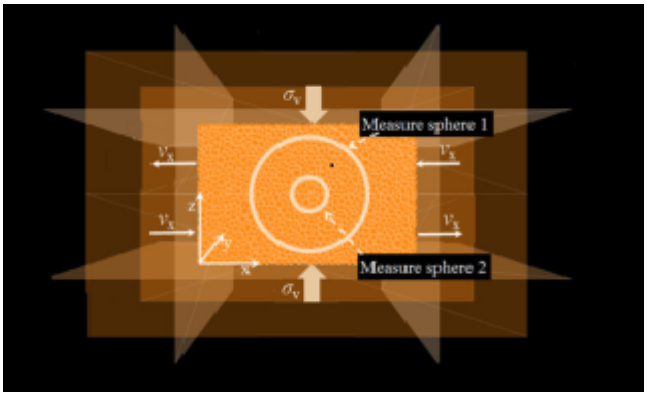
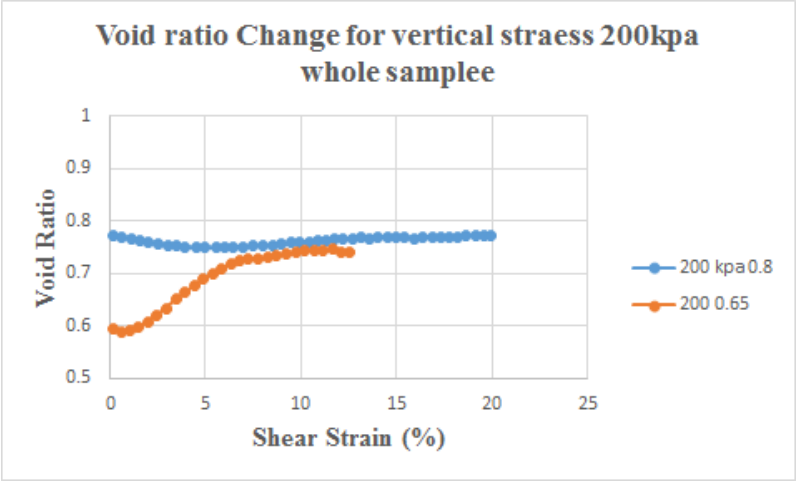
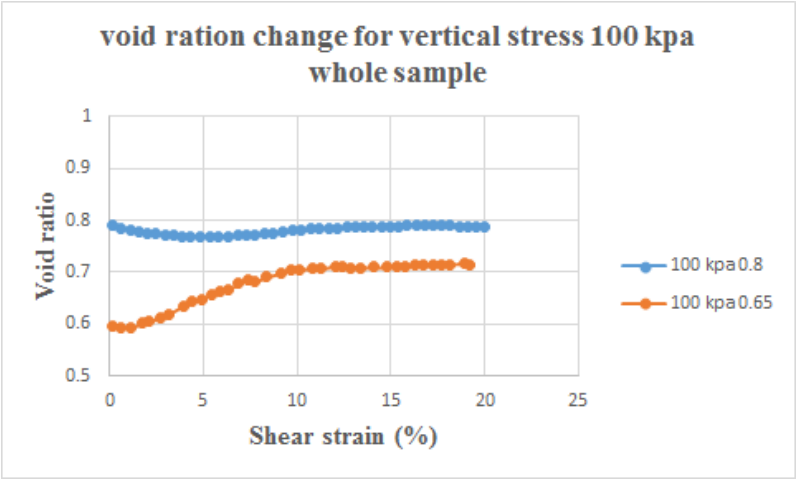


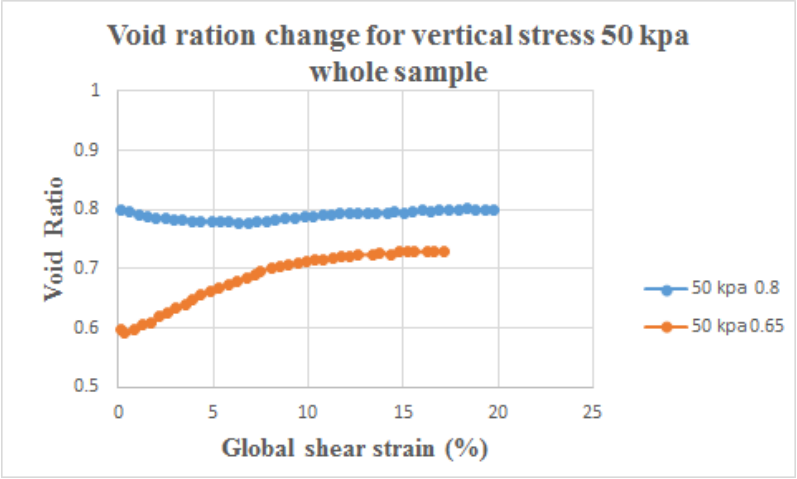
Figure 4.11: The Two Concentric Measuring spheres



(a) For vertical stress ($\sigma_v = 200kpa$)



(b) for under vertical stress ($\sigma_v = 100kpa$)



(c) for vertical stress ($\sigma_v = 50kpa$)

Figure 4.12: The evolution of void ratio measuring sphere 1 under different σ_v

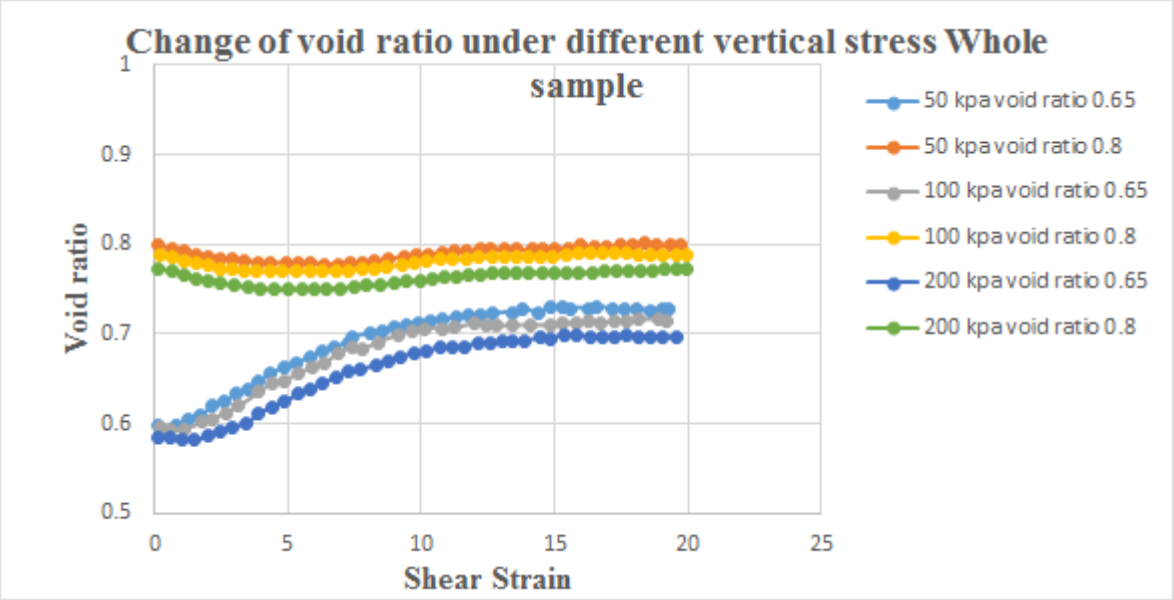
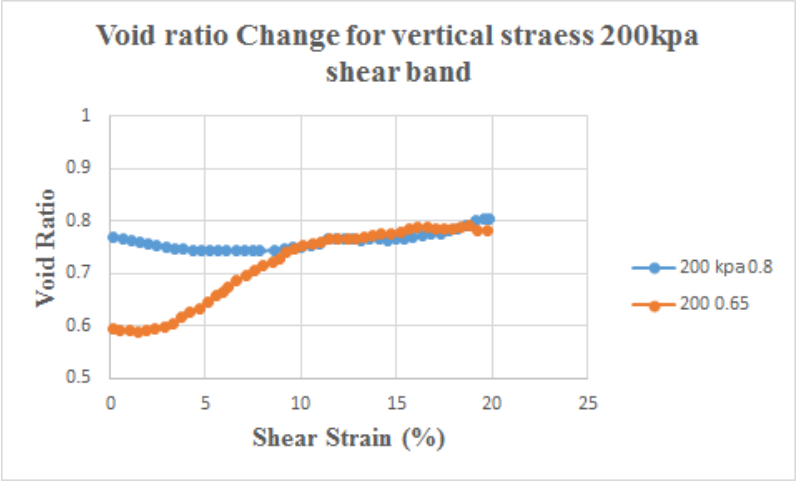
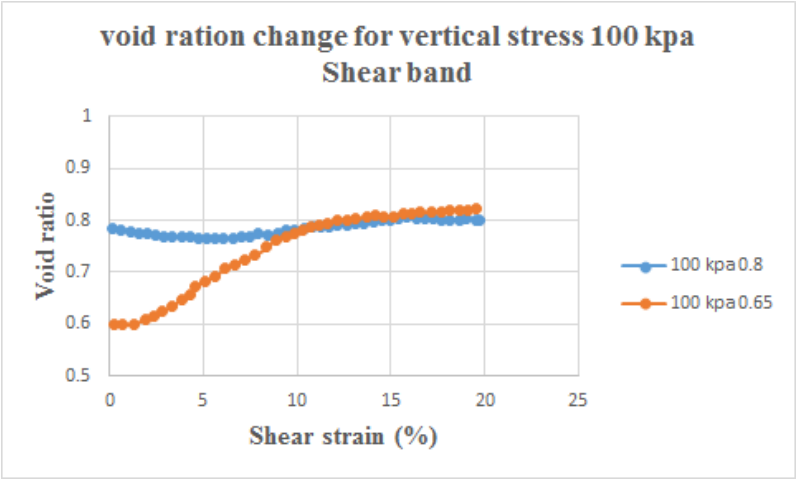


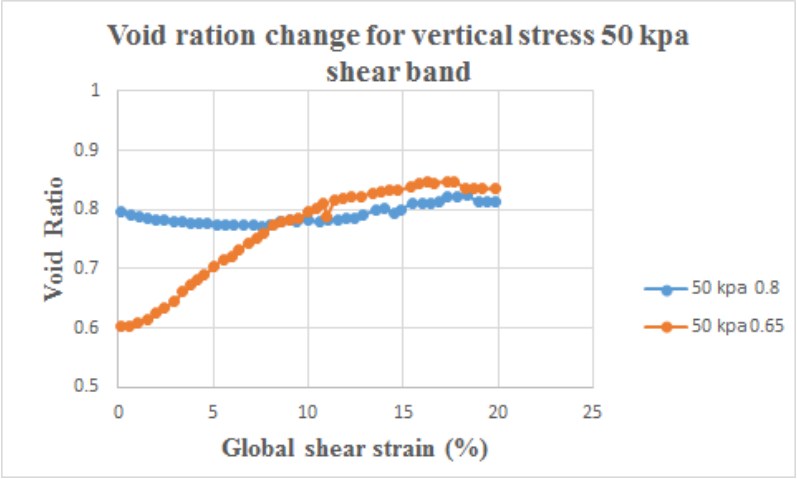
Figure 4.13: The evolution of void ratio DE samples in measuring sphere 1 (Whole sample)



(a) For($\sigma_v = 200kpa$)



(b) for ($\sigma_v = 100kpa$)



(c) for ($\sigma_v = 50kpa$)

Figure 4.14: The evolution of void ratio measuring sphere 2 under different σ_v

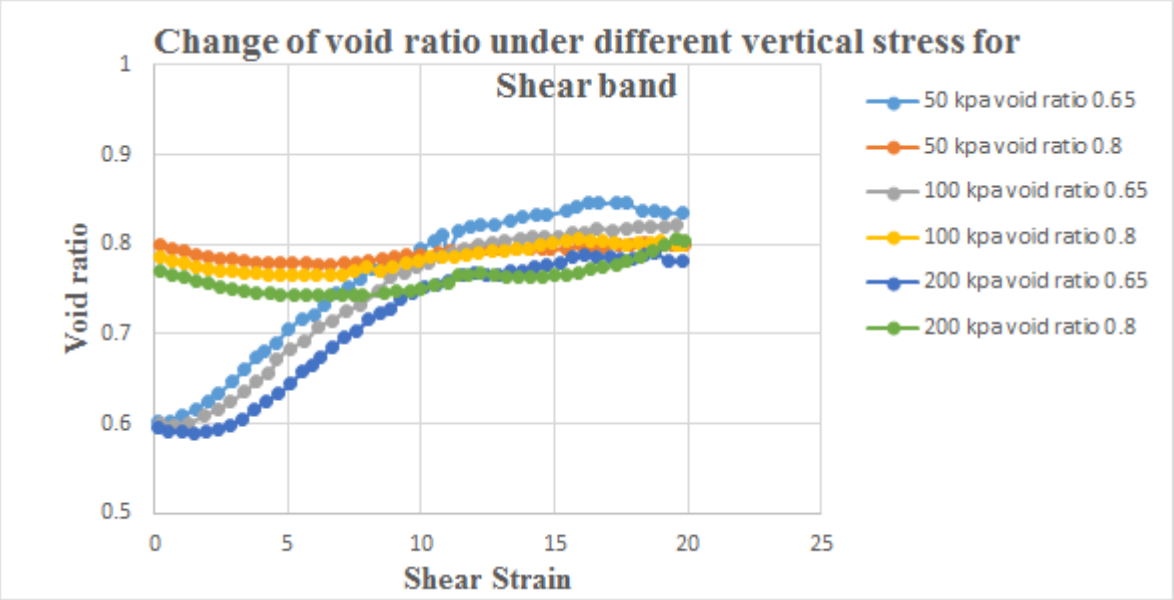


Figure 4.15: The evolution of void ratio DE samples in measuring sphere 2 (Shear zone)

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

The DEM was used to simulate a series of direct shear tests and the microscopic view of DST was studied on the shear behavior of dense and loose sand .The result was compared to the laboratory tests, and based on the results ,it can be concluded that the Discrete Element Method has strong capability to mimic the characteristics of sands in direct shear tests.And the microscopic characteristics of a sand grains are better investigated or examined in the DEM simulation than the laboratory DST since DEM allow us to monitor the microscopic parameters that are not amenable in lab or field investigations.

from the simulation result the following conclusion is arrived.

1. The deformation observed in the simulation is localized in the shear zone of the sample which is similar to observations from the laboratory Direct Shear Test.

2. The DEM assemblies shows strain softening and volumetric dilation in dense sand with initial porosity = 0.65 and, strain hardening and volumetric contraction in loose sand with initial porosity 0.8,Which represent the typical shear behavior of Dense and loose sand in laboratory DST. This result demonstrate the strong proficiency of DEM to capture the behavior of pure sand in direct shear test.

3. The critical properties of sand in DST should be investigated from the shear band rather than the whole sample. The non-uniform deformation of sands in DST make it a challenging task to investigate the critical property of sands since the deformation, volume change and stress value can be only read for the whole sample in lab DST procedure (Not mentioning the human error and apparatus calibration problems) . However the DEM simulation resolve this limitation through examining the sand properties in the shear band and the whole sample separately. The DEM simulation allow us to investigate the behavior of granular materials at microscopic level.

4. When the shear stress of the over all soil sample arrives at a stable state after large shear displacement ,the soils inside the shear zone reach a their critical values .Even though This properties can be noticed in laboratory DST procedures, it can not be measured through the conventional lab setups .Mostly the critical state behavior is investigated using triaxial test.However the DEM simulation can employ measuring zones that microscopical investigate the critical state behavior of the soils in shear zone and the whole sample simultaneously . Hence the DEM simulation enhance the shortcomings of the laboratory DST setups and procedures.

5. The DEM simulation is flexible to change the size of the shear box based on the Grain size distribution .

5.2 Recommendations

The behavior of discrete element samples depends not only on the microscopic parameters but also on particle shapes, particle size distribution, contact models and packing technique. While the adopted packing method and particle size distribution are considered realistic, spherical particle shapes and the contact model are somewhat unrealistic .In addition the numerical simulation in this study was based on a classical contact model (Cundall and Strack 1979a) which overlooks the angularity effects of DEM assembly. This contact model is most appropriate for sand particles with low aspect ratio. Elongated particles can be modeled as particle agglomerates, polygonal/polyhedral particles and ellipsoids to account for further particle scale details.Cohesive material can also be modeled with bonded particles .further more coupling with the finite element method and computational fluid dynamics should be evaluated in future works.

Appendices

APPENDIX A

INTRODUCTION

This appendix presents a technical manual for the development process of computer simulation tools to provides an efficient approach for solving coupled FE-DE geotechnical problems.The simulation is implemented in YADE, an open source code package for discrete element (DE) analysis . Since the package("YADE") is written in C++ with object oriented programming approach, and implemented in Linux operating system with python scripts ,Knowledge of Linux operating system, C++ and Python programming languages is strongly recommended.This appendix will introduce the basic knowledge and skills required to employ the computer simulation tools used for solving scientific research problems in general and geotechnical engineering in particular . I have assumed that the user of this appendix will have basic to middle scale computer literacy .

The appendix is organized in four parts . *Appendix A* covers the installation and configuration of the pre-required operating systems and software packages . The installation of ubuntu (linux Debian Distro) as stand alone operating system and as second layer which is on top of other operating system like Windows using *virtualization* is presented . Then the installation and configuration of *YADE* in ubuntu will be discussed . Finally installation of *paraview* which is a Data visualization software for 3D post-processing will be presented .

In *appendix B* the basic shell commands and linux directory structures is reviewed .The tutorials are selected from a standard linux tutorials which are crucial and more often used in this research . And *Appendix C* will briefly describe the

Architecture overview , commands and approaches used in YADE . Most of the commands used in the YADE module are also listed in the YADE's documentation (https://yade-dem.org/doc/).Finaly ,all the developed python codes for this research is attached in the last Appendix .Although the codes are designed and employed for this thesis , Researchers can amend,customize or improve the codes for further investigations which incorporate concepts such as computational fluid dynamics (fluid coupling),Deformable spheres ,particle angularity effect and other scientific considerations .(refer the recommendation).

A.1 Installation

A.1.1 Linux Installation(ubuntu)

What is GNU/Linux

Linux is an operating system: a series of programs that let you interact with your computer and run other application programs. An operating system consists of various fundamental programs which are needed by your computer so that it can communicate and receive instructions from users; read and write data to hard disks,tapes, and printers; control the use of memory; and run other software. The most important part of an operating system is the kernel. In a GNU/Linux system, Linux is the kernel component. The rest of the system consists of other programs, many of which were written by or for the GNU Project.Because the Linux kernel alone does not form a working operating system, we prefer to use the term "GNU/Linux" to refer to systems that many people casually refer to as "Linux".

Linux is modeled on the Unix operating system. From the start, Linux was designed to be a multi-tasking, multi-user system. These facts are enough to make Linux different from other well-known operating systems. However, Linux is even more different than you might imagine. In contrast to other operating systems, nobody owns Linux. Much of its development is done by unpaid volunteers.

Development of what later became GNU/Linux began in 1984, when the Free Software Foundation(http://www.fsf.org/) began development of a free Unix-like operating system called GNU .The GNU Project (http://www.gnu.org/) has devel-

oped a comprehensive set of free software tools for use with $Unix^{TM}$ and Unix-like operating systems such as Linux. These tools enable users to perform tasks ranging from the mundane (such as copying or removing files from the system) to the arcane(such as writing and compiling programs or doing sophisticated editing in a variety of document formats).

While many groups and individuals have contributed to Linux,the largest single contributor is stil the Free Software Foundation, which created not only most of the tools used in Linux, but also the philosophy and the community that made Linux possible.

What is Ubuntu

Ubuntu is a complete Linux operating system, freely available with both community and professional support. The Ubuntu community is built on the ideas enshrined in the Ubuntu Manifesto: that soft-ware should be available free of charge, that software tools should be usable by people in their local language and despite any disabilities, and that people should have the freedom to customize and alter their software in whatever way they see fit.

Ubuntu is suitable for both desktop and server use. The current Ubuntu release supports Intel x86(IBM-compatible PC), AMD64 (x86-64), ARMv7, ARMv8 (ARM64), IBM POWER8/POWER9(ppc64el), IBM Z zEC12/zEC13/z14 and IBM LinuxONE Rockhopper I+II/Emporer I+II (s390x).

Ubuntu includes thousands of pieces of software, starting with the Linux kernel version 4.15 and GNOME 3.28, and covering every standard desktop application from word processing and spread-sheet applications to internet access applications, web server software, email software, programming languages and tools for scientific researches such as C++,python ,yade Esys-escript ... and many more .

How to install Ubuntu

Prerequisites :-

System requirements (recommended):

- 2 GHz dual-core processor

- 4GB memory
- 25GB available disk space for storage (less if installing the minimal version)
- DVD drive or USB port

Ubuntu can be installed in two ways .The first one is the ***clean installation*** . It is the conventional way of installing the operating system . In this process Ubuntu will be installed as a primary operating system . Here's a road map for the steps you will take during the clean installation process.

1. Back up any existing data or documents on the hard disk where you plan to install
2. . Gather information about your computer and any needed documentation, before starting the installation.
3. Locate and/or download the installer software and any specialized driver or firmware files your machine requires.
4. Set up boot media such as CDs/DVDs/USB sticks or provide a network boot infrastructure from which the installer can be booted
5. Boot the installation system
6. Select the installation language.
7. Activate the ethernet network connection, if available.
8. If necessary, resize existing partitions on your target hard disk to make space for the installation.
9. Create and mount the partitions on which Ubuntu will be installed
10. Watch the automatic download/install/setup of the base system.
11. Install a boot loader which can start up Ubuntu and/or your existing system.
12. Load the newly installed system for the first time

for step by step installation of this type please visit (https://ubuntu.com/tutorials/install-ubuntu-desktop)

The second option of installing Ubuntu is using ***Virtual Machine*** . In this case the operating system is installed on top of other primary operating systems using virtualization soft-wares such as virtualBox(for this thesis i have used virtualBox6.1),Vmware Work station,hyper v and others .Ubuntu will be installed as a guest operating system on top of windows , Linux, Macintosh, and Solaris . Here is the step to be taken for this type of installation process :-

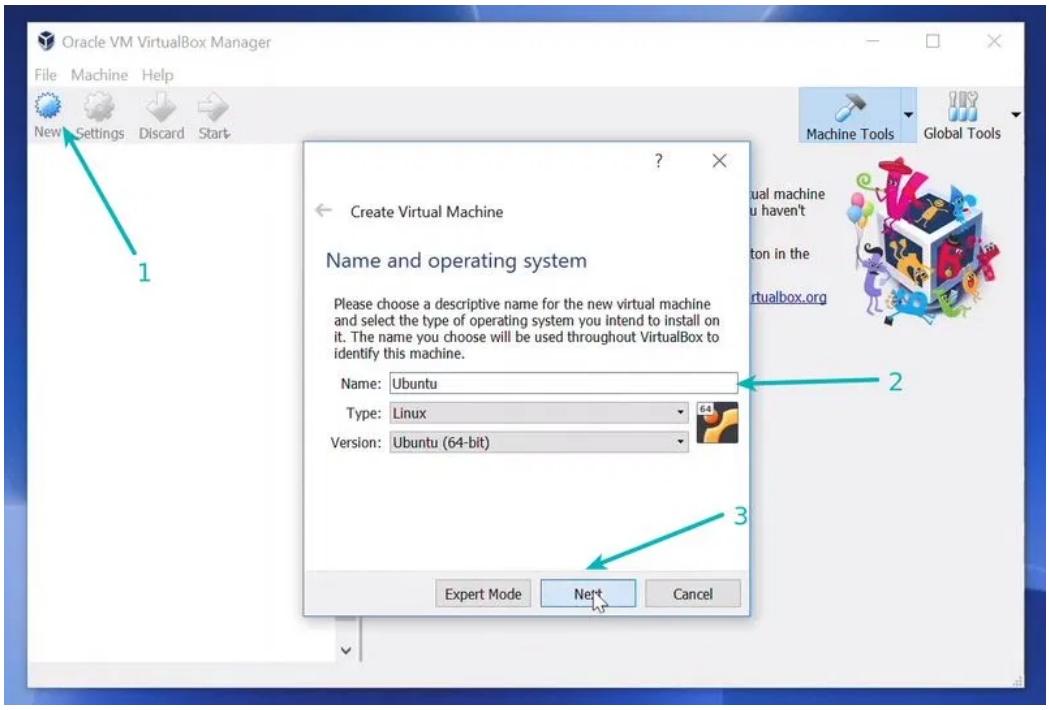
1. ***Download and install VirtualBox*** :- Go to the website of Oracle VirtualBox and get the latest stable version from here(https://www.virtualbox.org/). Installing VirtualBox is not rocket science. Just double-click on the downloaded.exe file and follow the instructions on the screen. It is like installing any regular software on Windows.

2. ***Download the Ubuntu ISO*** :- Next, you need to download the ISO file of the Linux distribution. You can get this image from the official website of the Linux distribution you are trying to use.

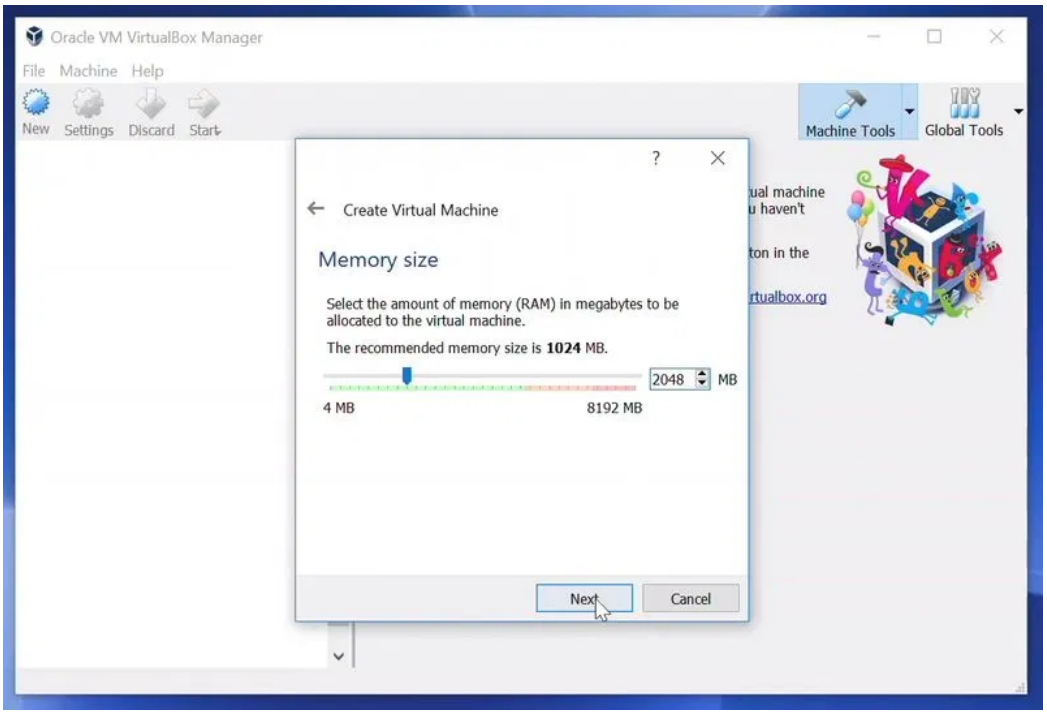
   I am using Ubuntu 18.04 for this research , and you can download ISO images for Ubuntu from (https://www.ubuntu.com/desktop).

3. ***Install Ubuntu using VirtualBox*** :- You have installed VirtualBox and you have downloaded the ISO for Ubuntu. You are now set to install Ubuntu in VirtualBox.

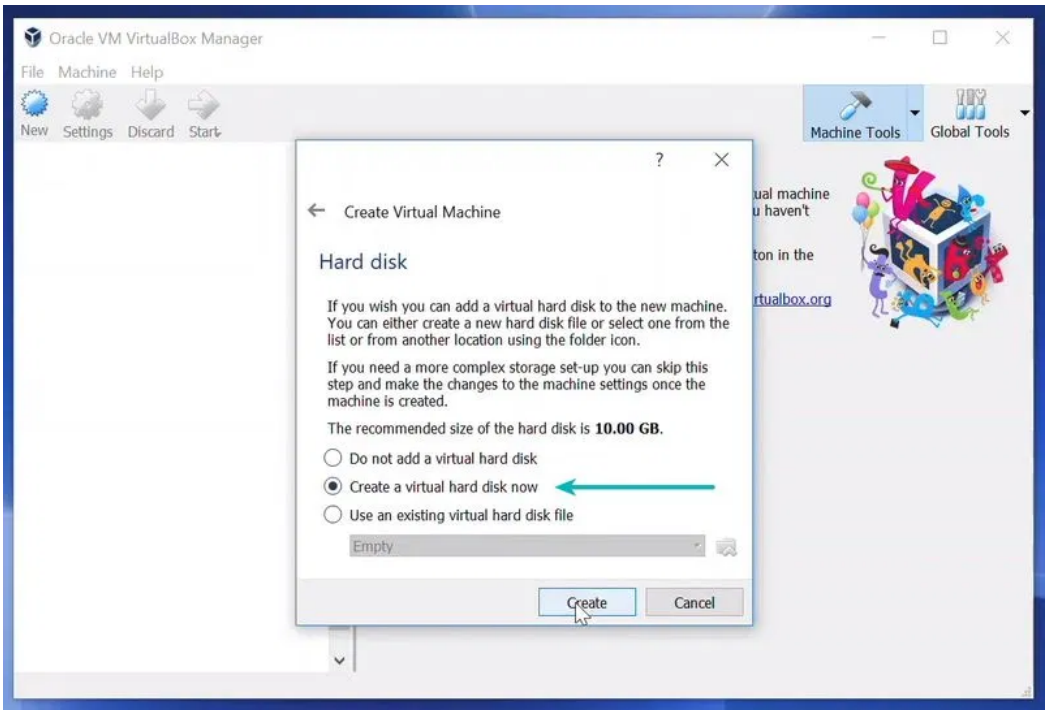
   - Start VirtualBox, and click on the New symbol. Give the virtual OS a relevant name



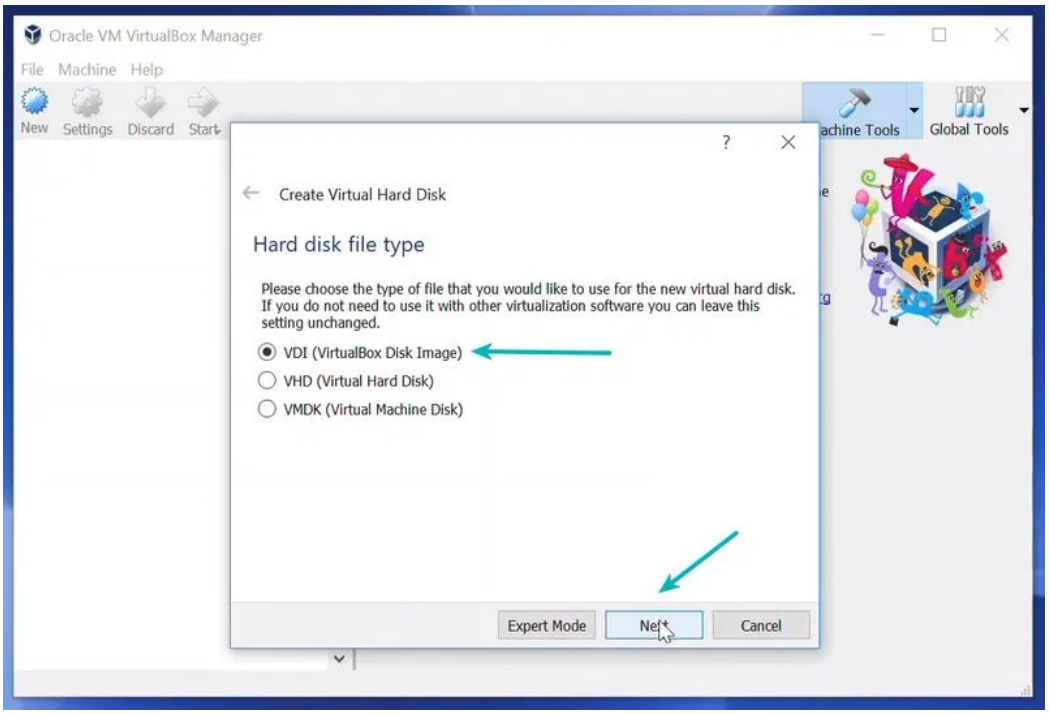
- Allocate RAM to the virtual OS. My system has 8GB of RAM and I decided to allocate 2GB of it. You can use more RAM if your system has enough extra.



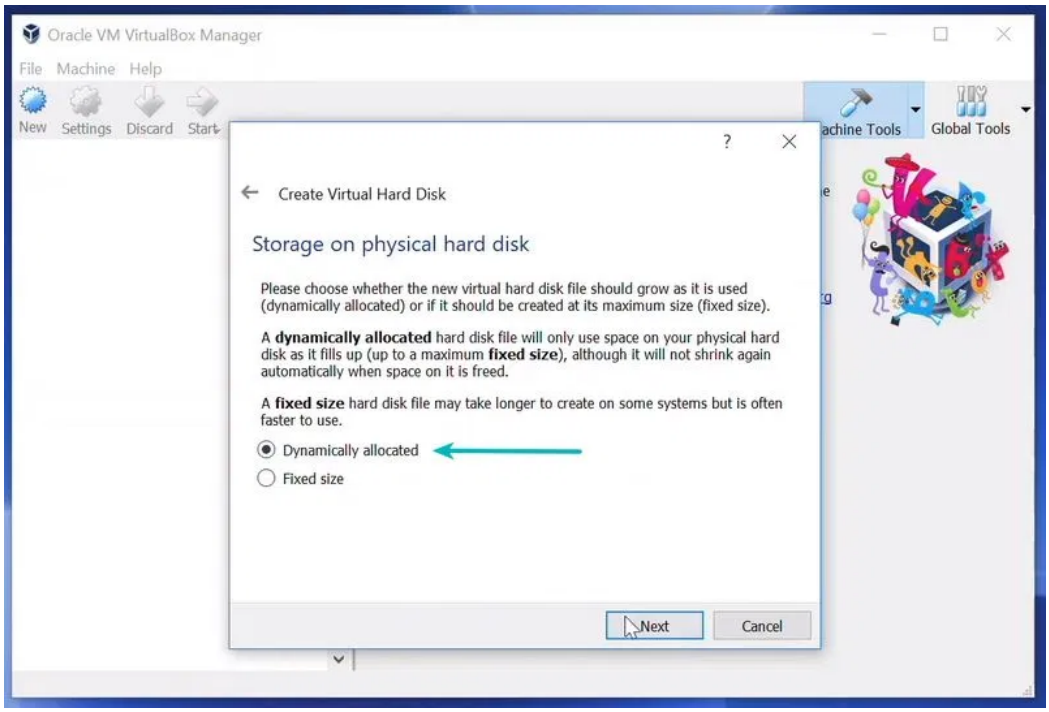
- Create a virtual disk. This serves as the hard disk of the virtual Linux system. It is where the virtual system will store its files.



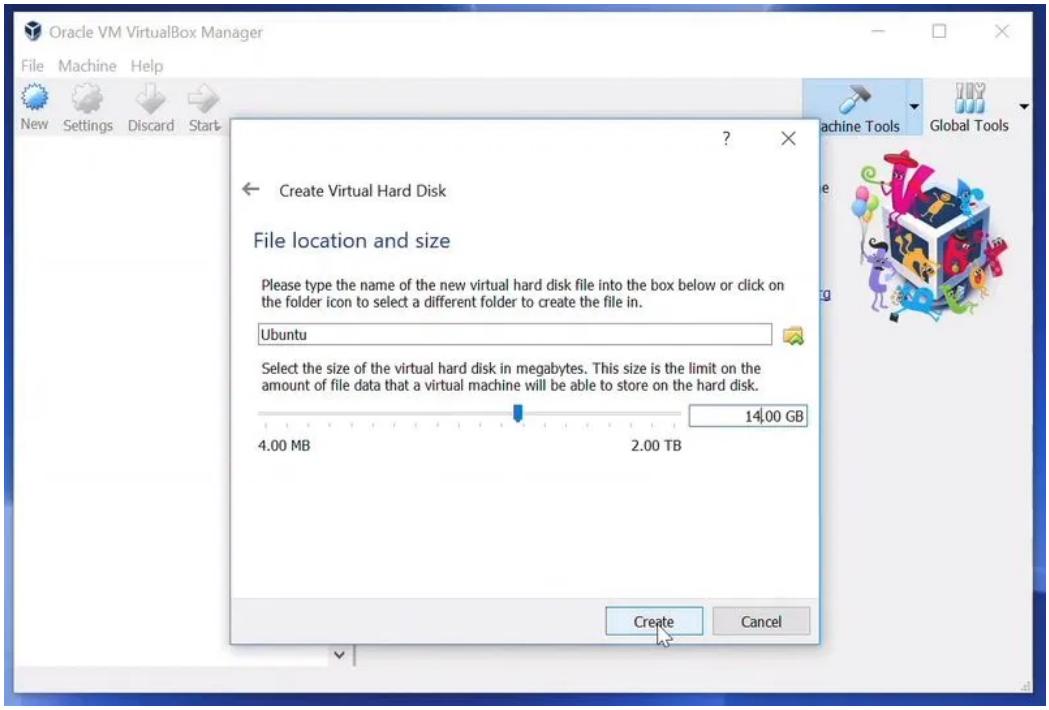
- I recommend using the VDI file type here.



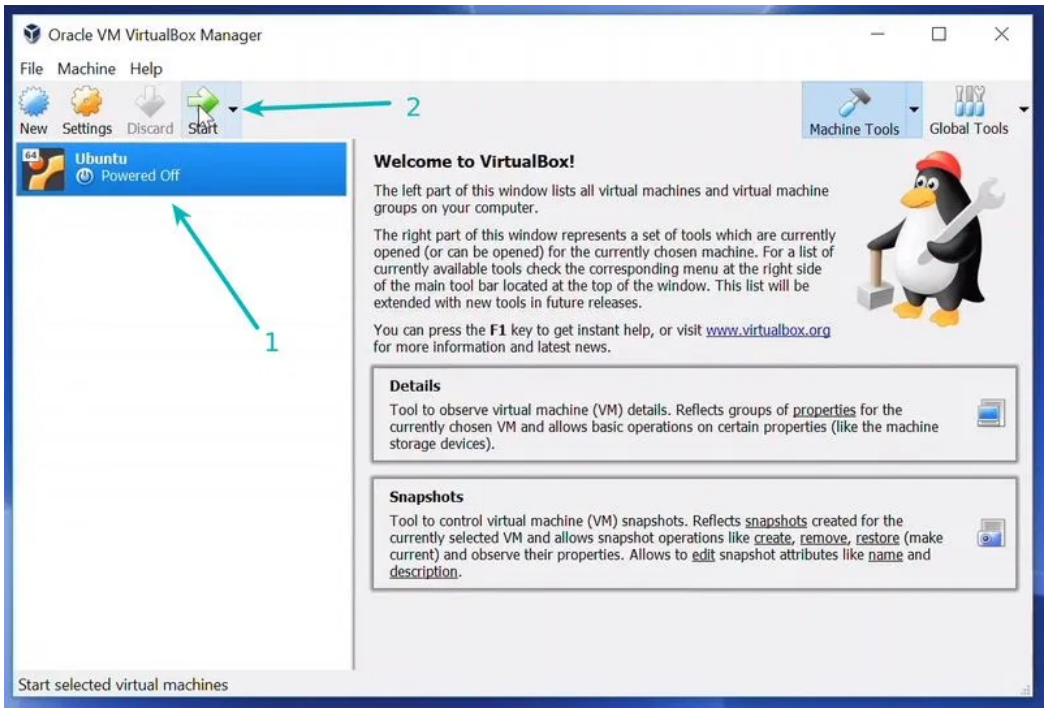
- You can choose either the “Dynamically allocated” or the “Fixed size” option for creating the virtual hard disk



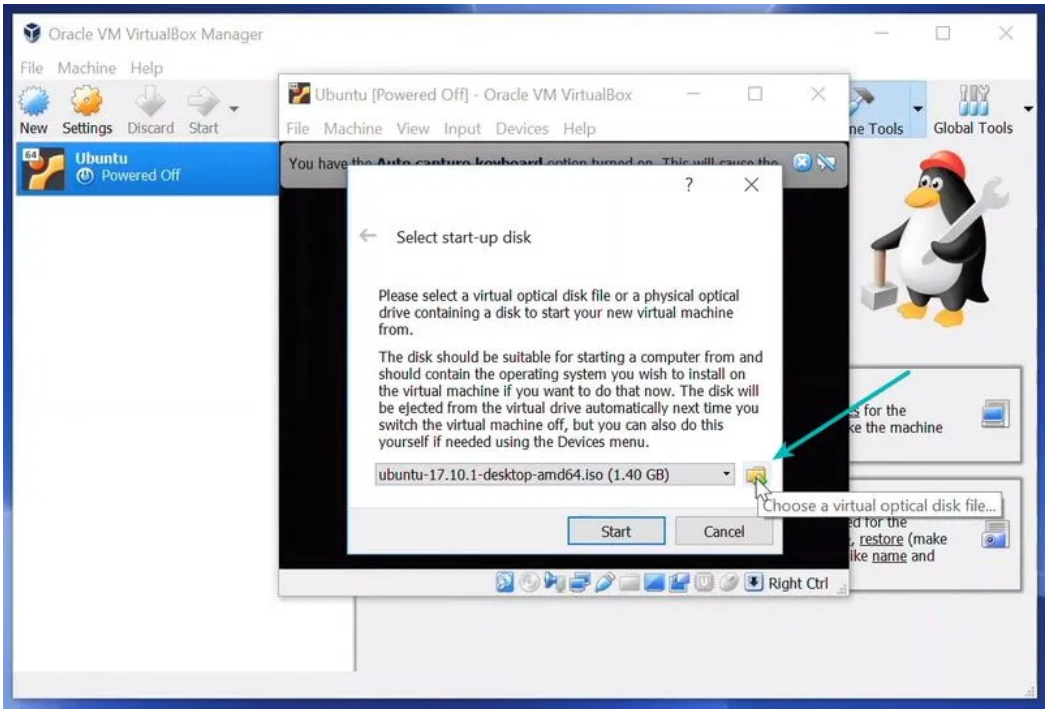
- The recommended size is 10 GB. However, I suggest giving it more space if possible. 15-20 GB is preferable.



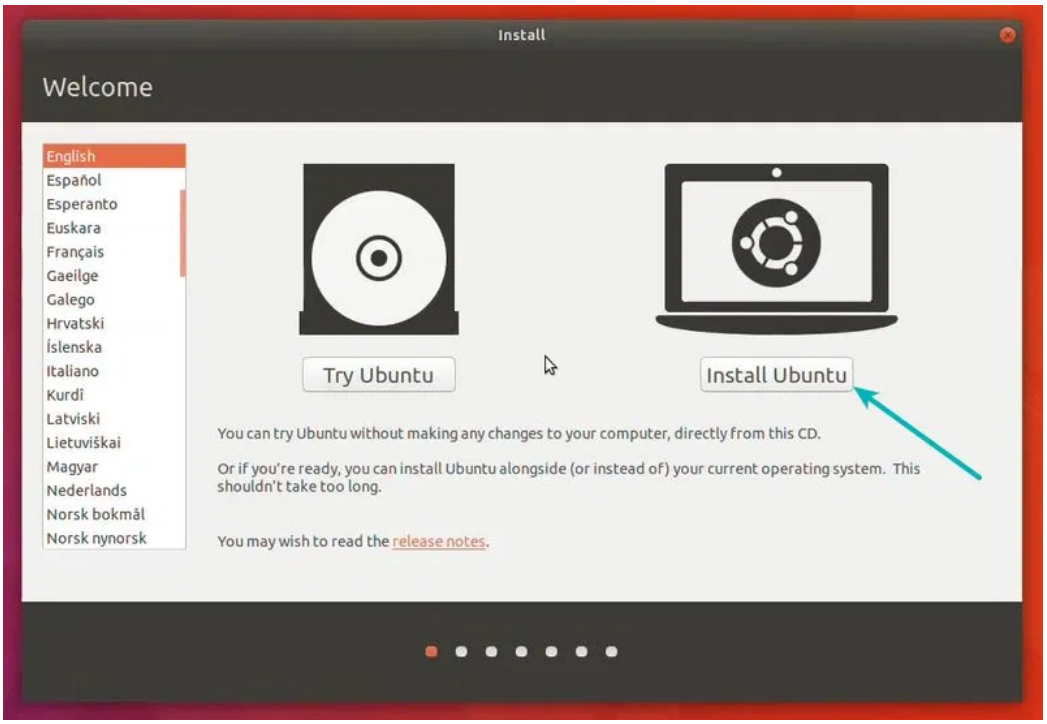
- Once everything is in place, it's time to boot that ISO and install Linux as a virtual operating system.



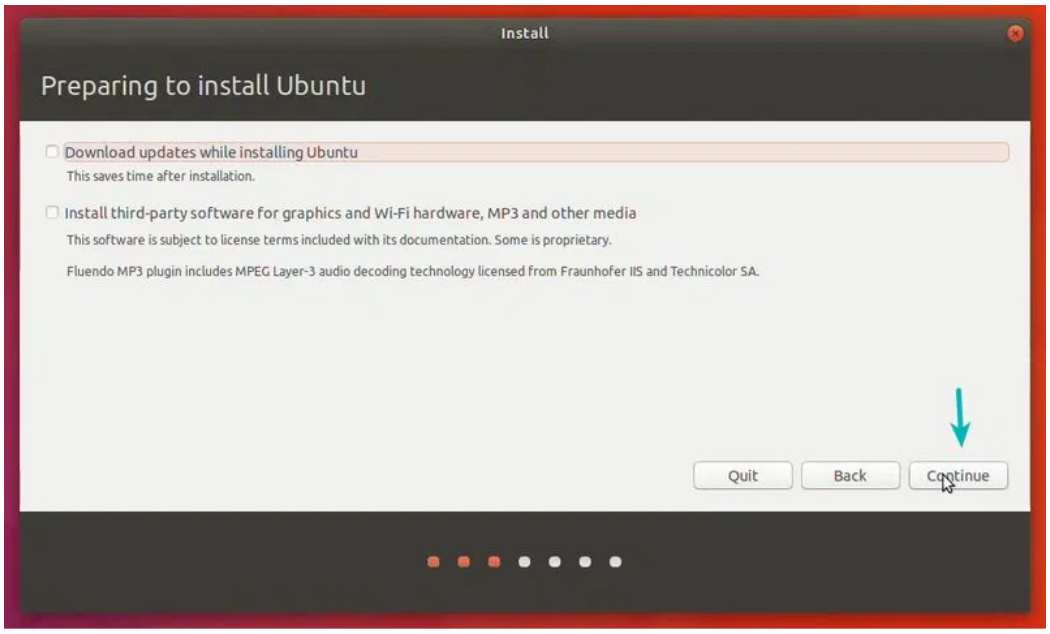
- If VirtualBox doesn't detect the Linux ISO, browse to its location by clicking the folder icon as shown in the picture below



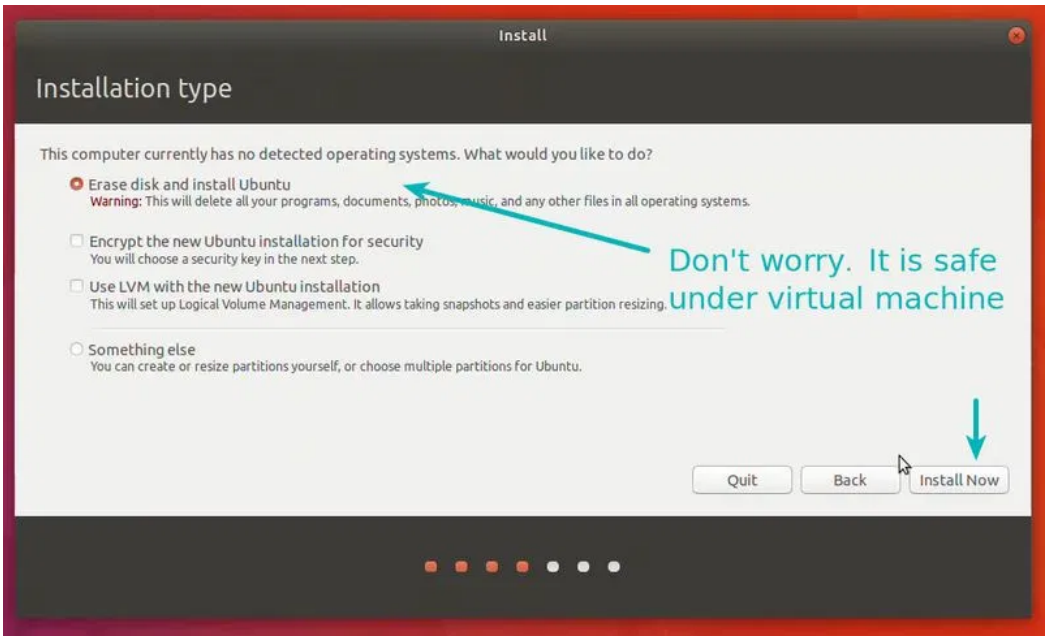
- Soon you'll find yourself inside Linux. You should be presented with the option to install it. Things from here are Ubuntu-specific. Other Linux distributions may have slightly different looking steps, but it won't be complicated at all.



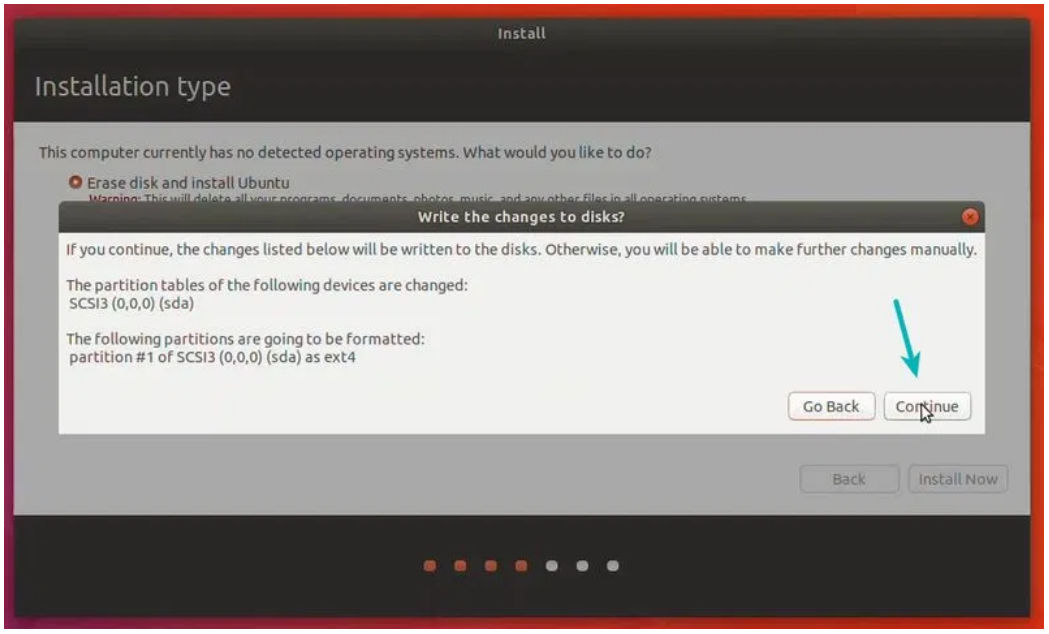
- You can skip to Continue.



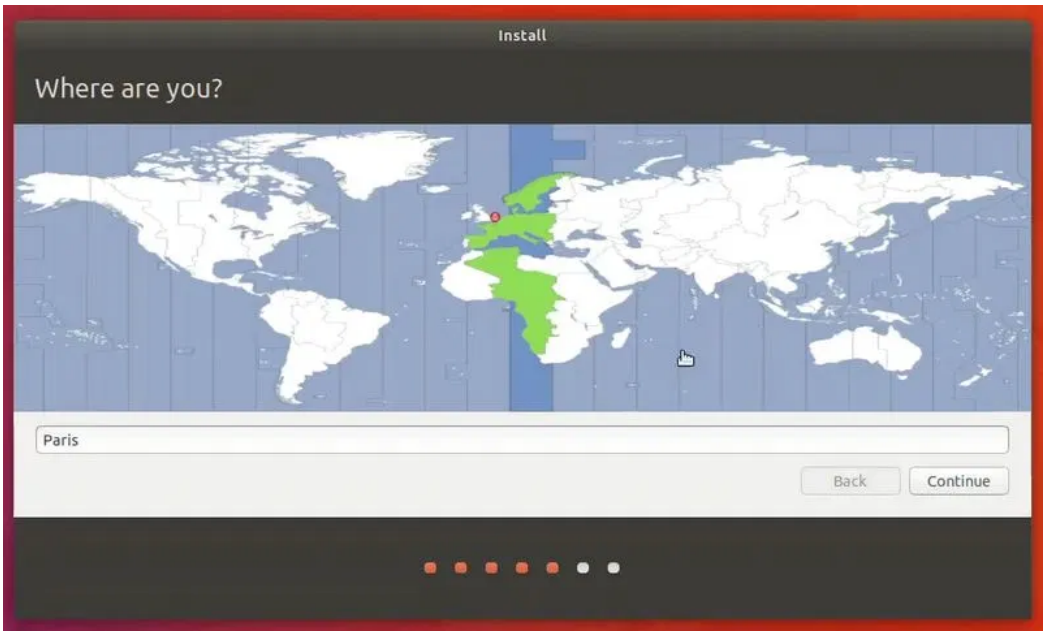
- Select 'Erase disk and install Ubuntu'. Don't worry. It won't delete anything on your Windows operating system. You are using the virtual disk space of 15-20GB that we created in previous steps. It won't impact the real operating system.



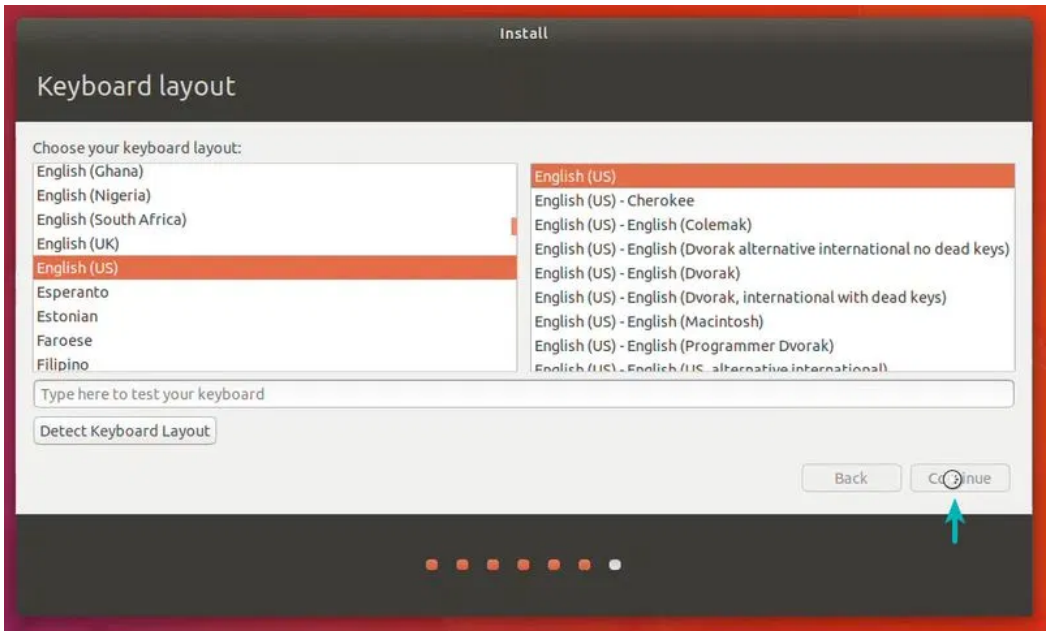
- Just click on Continue.



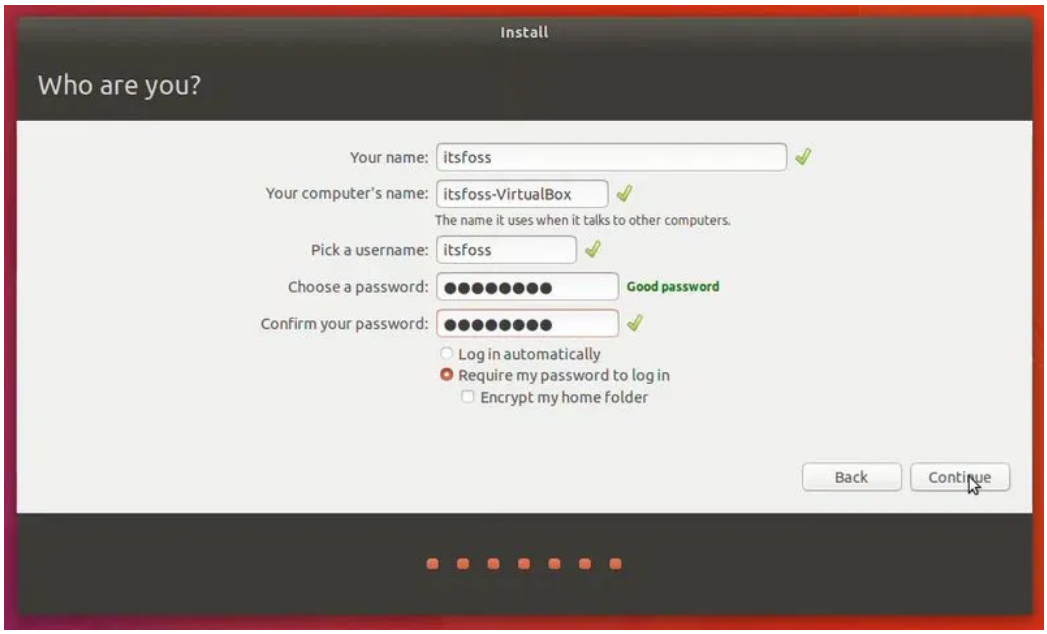
- Things are pretty straightforward from here.



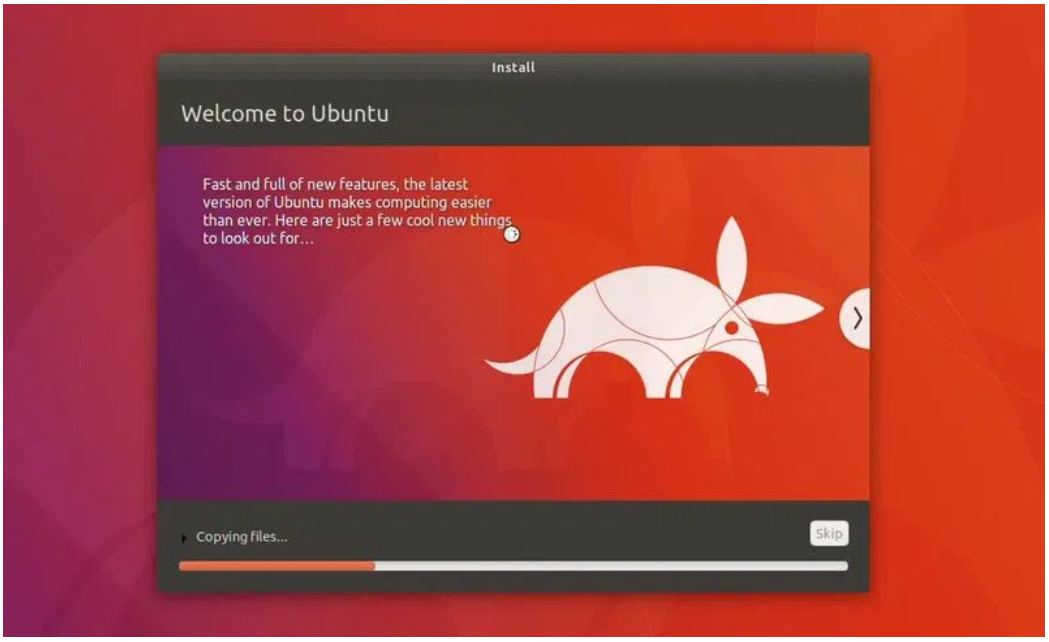
- Self explanatory.



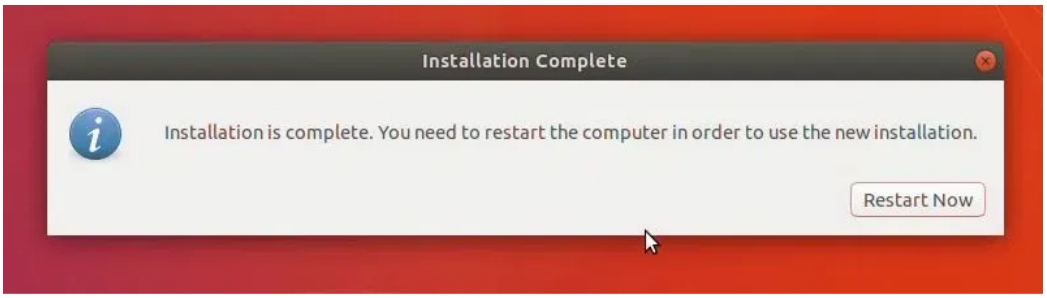
- Try to choose a password that you can remember. You can also reset the password in Ubuntu if you forget it.



- You are almost done. It may take 10-15 minutes to complete the installation.



- Once the installation finishes, restart the virtual system.



- If it gets stuck on the screen below, you may close the VirtualBox.

- And that's all. From now on, just click on the installed Linux virtual machine. You'll be able to use it directly. The installation is a one time only process. You can even delete the Linux ISO that you downloaded earlier.

I strongly recommend using VirtualBox Guest Additions on Ubuntu for it provides better compatibility and you would be able to use copy-paste and drag-drop between Linux and Windows.

A.1.2 YADE Installation

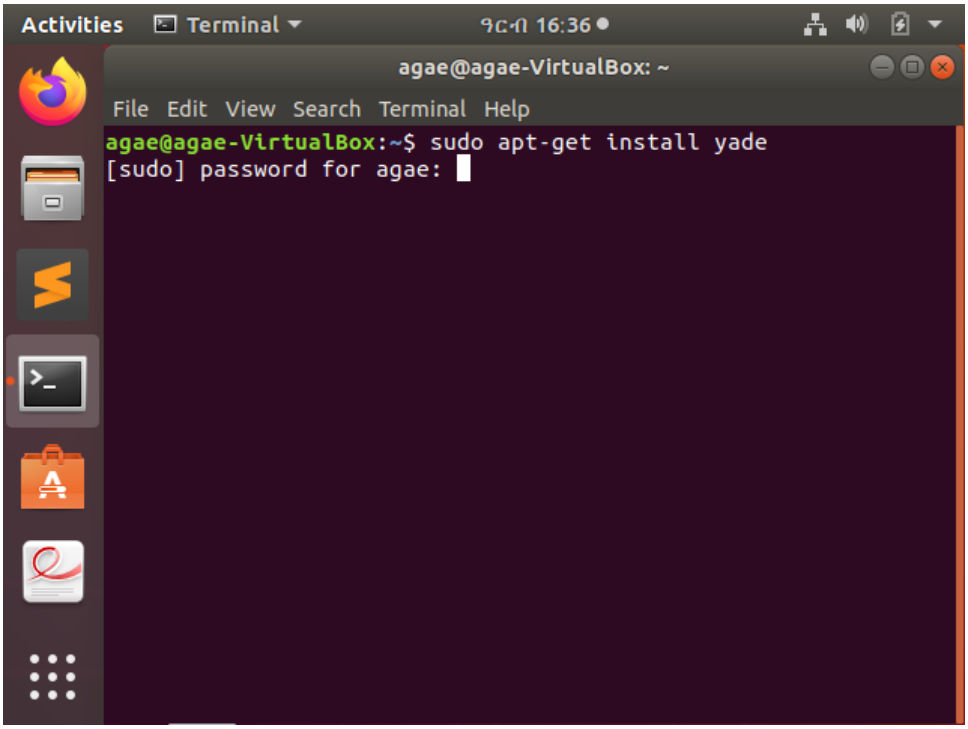
After installing ubuntu ,installation of Yade can be done in Two ways. ***Packages***(pre-compiled binaries)installation And ***source code*** installation .

Package installation

This installation method is the easiest method to install stable and daily packages .However if we wanted to modify the source code this installation technique did not have the flexibility to do so .

Since 2011, all Ubuntu (starting from 11.10, Oneiric) and Debian (starting from Wheezy) versions have *Yade* in their main repositories. There are only stable releases in place. To install *Yade*, start Ubuntu , open Terminal and run the following command :-

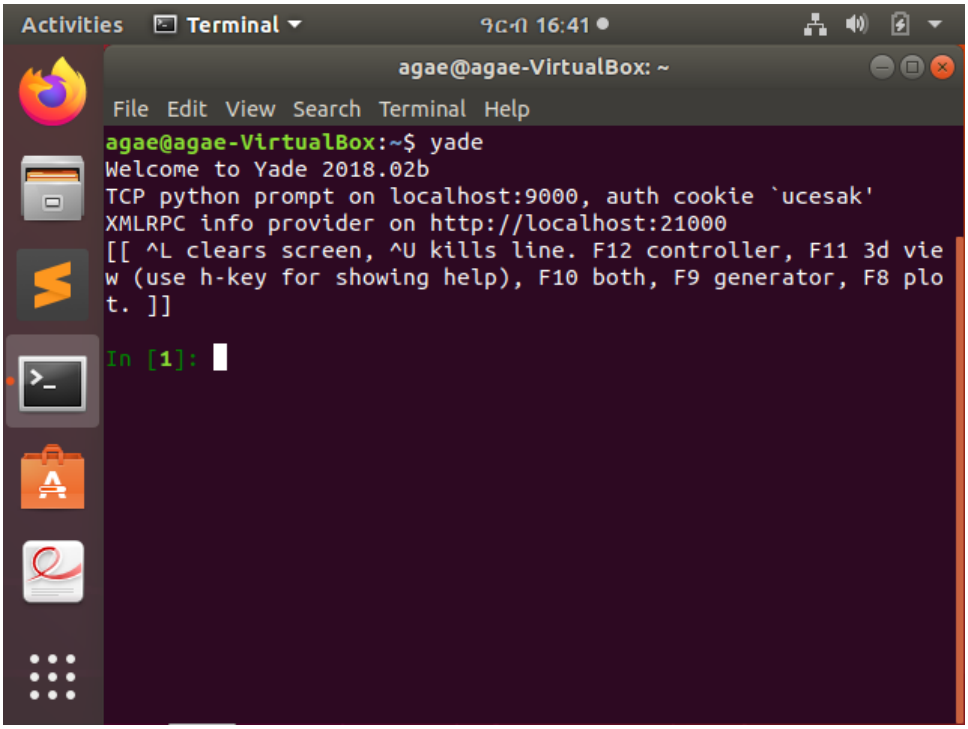
```
1  sudo apt-get install yade
```



you need root credential to install yade

Then *yade* can be started with the following command

```
yade
```



for Parametric simulation use the following command to start *yade*

```
yade-batch
```

Source code installation

Installation from source code is reasonable, when you want to add or modify constitutive laws, engines,functions etc. In this method of installation two types of versions are available. These are ***Release version*** and ***Development* (trunk)**version . Installing the latest trunk version allows one to use newly added features, which are not yet available in packaged versions.You should download the development version (called *trunk*) if you want to modify the source code, as you might encounter problems that will be fixed by the developers. *Release* versions will not be updated (except for updates due to critical and easy-to-fix bugs), but generally they are more stable than the trunk.

1. *Releases* can be downloaded from (https://launchpad.net/yade/+download), as compressed archive. Uncompressing the archive gives you a directory with the sources.

2. The development version (trunk) can be obtained from the code repository(https://gitlab.com/ya
dev) at GitLab.

Yade relies on a number of external software to run; they are checked before the compilation starts. Some of them are only optional.

- *cmake* build system(www.cmake.org)
- *gcc*compiler (g++)(https://gcc.gnu.org/); other compilers will not work; you need g++¿=4.2 for openMP support
- boost 1.47 or later
- Qt library
- freeglut3
- libQGLViewer
- python, numpy, ipython, sphinx, mpi4py
- matplotlib
- eigen algebra library (minimal required version 3.2.1)
- gdb debugger
- sqlite3 database engine
- Loki library
- VTK library (optional but recommended)
- SuiteSparse sparse algebra library (fluid coupling, optional, requires eigen¿=3.1)
- Metis matrix preconditioning (fluid coupling, optional)

The following commands have to be executed in the command line of your corresponding distribution. in our case Ubuntu 18.04 Just copy and paste to the terminal. Note, to execute these commands you need root privileges.

```
sudo apt install cmake git freeglut3-dev libloki-dev libboost-all-dev
fakeroot dpkg-dev build-essential g++ python3-dev python3-ipython
python3-matplotlib libsqlite3-dev python3-numpy python3-tk gnuplot
libgts-dev python3-pygraphviz libvtk6-dev libeigen3-dev python3-xlib
python3-pyqt5 pyqt5-dev-tools python3-mpi4py python3-pyqt5.qtwebkit
gtk2-engines-pixbuf python3-pyqt5.qtsvg libqglviewer-dev-qt5
python3-pil libjs-jquery python3-sphinx python3-git libxmu-dev
libxi-dev libcgal-dev help2man libbz2-dev zlib1g-dev libopenblas-dev
libsuitesparse-dev libmetis-dev python3-bibtexparser python3-future
coinor-clp coinor-libclp-dev python3-mpmath libmpfr-dev libmpfrc++-dev
```

A.1.3 Paraview Installation (Post-processing)

ParaView is an open-source, multi-platform application designed to visualize data sets of size varying from small to very large. The goals of the ParaView project include the following:

- Develop an open-source, multi-platform visualization application.
- Support distributed computation models to process large data sets.
- Create an open, flexible, and intuitive user interface.
- Develop an extensible architecture based on open standards.

ParaView runs on distributed and shared memory parallel as well as single processor systems and has been successfully tested on Windows, Mac OS X, Linux and various Unix workstations, clusters and supercomputers. Under the hood, Paraview uses the Visualization Toolkit(VTK) as the data processing and rendering engine and has a user interface written using Qt.

install *paraview* by entering the following commands in the terminal:

```
sudo apt-get update
sudo apt-get install paraview
```

To start paraview run a command *paraview* in command line

APPENDIX B

HAND ON TUTORIAL ON LINUX

B.1 Shell Basics

B.1.1 Directory Tree

Directory tree is hierarchical way to organize files in operating systems. A typical tree in Linux looks like this:

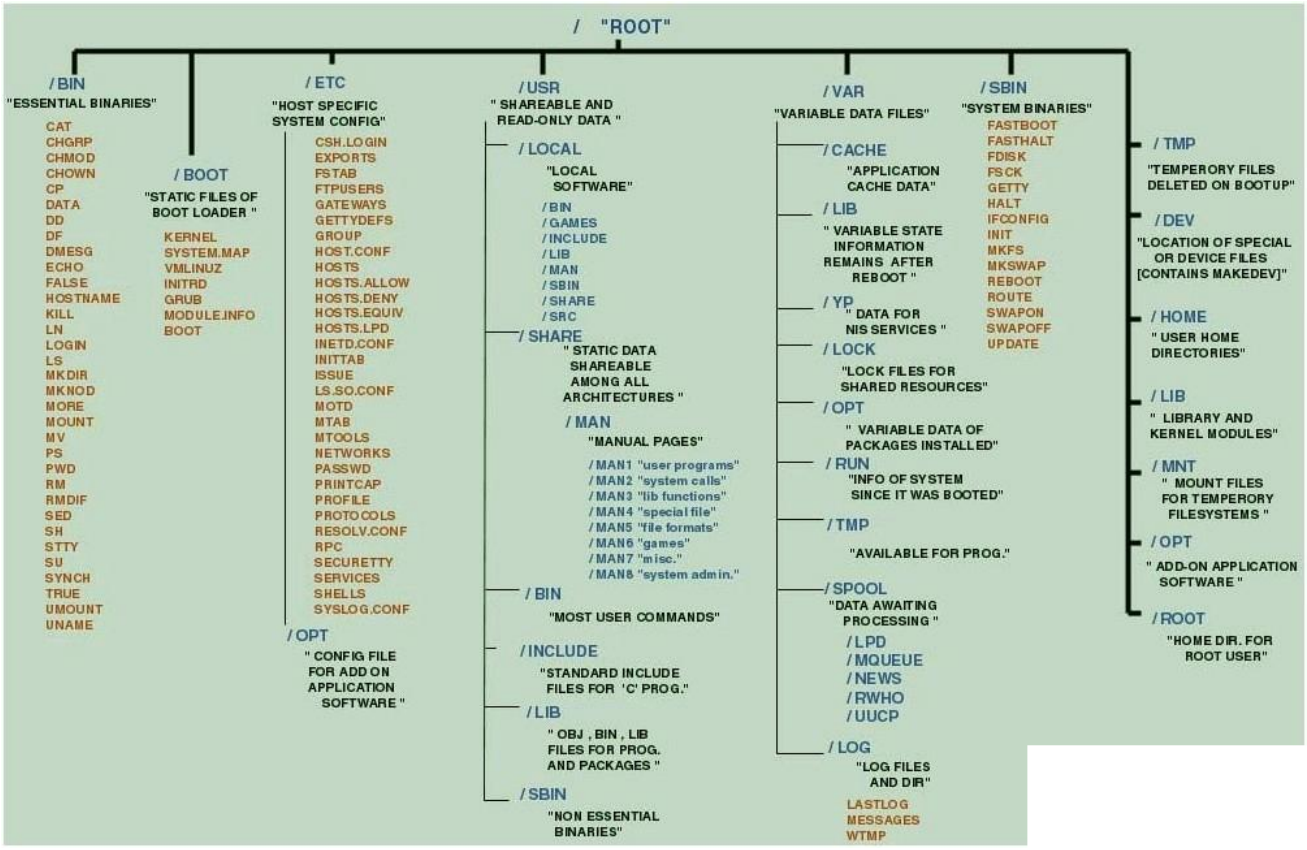


Figure B.1: linux Directory Structure

- / Root directory , the first hierarchy, the entire file system hierarchy
- **/bin/** Low-level programs (executable file): commands available for all users, for example:*cat ls cp*, And similar to */usr/bin*.
- **/boot/** System start up
- **/lib/** Low-level libraries
- **/dev/** Hardware access
- **/sbin/** Administration programs
- **/ proc/** System information
- **/var/** Files modified by system services
- **/root/** Root (administrator) home directory
- **/etc/** Configuration files

- **/media/** External drives
- **/tmp/** Temporary files
- **/usr/** Everything for normal operation (usr = UNIX system resources)
  - **/bin/** User programs
  - **/sbin/** Administration programs
  - **/include/** Header files for c/c++
  - **/lib/** Libraries
  - **/local/** Locally installed software
  - **/doc/** Documentation
- **/home/** Contains the user's home directories
  - **/user1/** Home directory for user1
  - **/user2/** Home directory for user2
  - **/user3/** Home directory for user3

B.1.2 Useful Linux Command Reference Table

FILE COMMANDS		
1	ls	Directory listing
2	ls -al	Formatted listing with hidden files
3	ls -lt	Sorting the Formatted listing by time modification
4	cd dir	Change directory to dir
5	cd	Change to home directory
6	pwd	Show current working directory
7	mkdir dir	Creating a directory dir
8	cat > file	Places the standard input into the file
9	more file	Output the contents of the file
10	head file	Output the first 10 lines of the file
11	tail file	Output the last 10 lines of the file
12	tail -f file	Output the contents of file as it grows,starting with the last 10 lines
13	touch file	Create or update file
14	rm file	Deleting the file
15	rm -r dir	Deleting the directory
16	rm -f file	Force to remove the file
17	rm -rf dir	Force to remove the directory dir
18	cp file1 file2	Copy the contents of file1 to file2
19	cp -r dir1 dir2	Copy dir1 to dir2;create dir2 if not present
20	mv file1 file2	Rename or move file1 to file2,if file2 is an existing directory
21	ln -s file link	Create symbolic link link to file

Table B.1: Linux File command table

PROCESS MANAGEMENT COMMANDS		
1	ps	To display the currently working processes
2	top	Display all running process
3	kill pid	Kill the process with given pid
4	killall proc	Kill all the process named proc
5	pkill pattern	Will kill all processes matching the pattern
6	bg	List stopped or background jobs,resume a stopped job in the background
7	fg	Brings the most recent job to foreground
8	fg n	Brings job n to the foreground

Table B.2: Linux process management command table

SEARCHING COMMAND		
1	grep pattern file	Search for pattern in file
2	grep -r pattern dir	Search recursively for pattern in dir
3	command | grep pattern	Search pattern in the output of a command
4	locate file	Find all instances of file
5	find . -name filename	Searches in the current directory (represented by a period) and below
6	pgrep pattern	Searches for all the named processes , that matches with the pattern

Table B.3: Linux searching command table

SYSTEM INFO COMMAND		
1	date	Show the current date and time
2	cal	Show this month's calender
3	uptime	Show current uptime
4	w	Display who is on line
5	whoami	Who you are logged in as
6	finger user	Display information about user
7	uname -a	Show kernel information
8	cat /proc/cpuinfo	Cpu information
9	cat proc/meminfo	Memory information
10	man command	Show the manual for command
11	df	Show the disk usage
12	du	Show directory space usage
13	free	Show memory and swap usage
14	whereis app	Show possible locations of app
15	which app	Show which applications will be run by default

Table B.4: Linux system info command table

FILE COMPRESSION COMMAND		
1	tar cf file.tar file	Create tar named file.tar containing file
2	tar xf file.tar	Extract the files from file.tar
3	tar czf file.tar.gz files	Create a tar with Gzip compression
4	tar xzf file.tar.gz	Extract a tar using Gzip
5	tar cjf file.tar.bz2	Create tar with Bzip2 compression
6	tar xjf file.tar.bz2	Extract a tar using Bzip2
7	gzip file	Compresses file and renames it to file.gz
8	gzip -d file.gz	Decompresses file.gz back to file

Table B.5: Linux file compression command table

Linux keyboard Shortcuts		
1	ctrl+c	Halts the current command
2	ctrl+z	Stops the current command
3	ctrl+d	Logout the current session, similar to exit
4	ctrl+w	Erases one word in the current line
5	ctrl+u	Erases the whole line
6	ctrl+r	Type to bring up a recent command
7	!!	Repeats the last command
8	exit	Logout the current session

Table B.6: Linux keyboard Shortcut Table

APPENDIX C

HANDS ON TUTORIAL ON YADE

C.1 Starting yade

If yade is installed on the machine, it can be run as any other program; without any arguments, it runs in the "dialog mode", where a command-line is presented:refer figure **??**

The command-line is in fact python, enriched with some yade-specifc features. (Pure python interpreter can be run with *python* or i*python* commands). Instead of typing commands on-by-one on the command line, they can be be written in a file (with the .py extension) and given as argument to *Yade*:

To run a script *example.py* from the terminal:

```
yade example.py
```

If users want to exit YADE immediately after running the script, *-x* is added:

```
yade -x example.py
```

To exit a simulation:

```
Exit() or ctrl+D
```

C.2 Architecture Overview

yade is developed through object oriented programming approach. Different class and objects(instance of class)are designed to assure flexibility of software design,code

re usability easy maintenance and modification of existing code.Hence yade makes clear distinction of 2 families of classes: *data components* and *functional components* . The former only store data without providing functionality, while the latter define functions operating on the data. In programming, this is known as visitor pattern (as functional components "visit" the data, without being bound to them explicitly).Entire simulation, i.e. both data and functions, are stored in a single *Scene* object. It is accessible through the **Omega** class in python (a singleton), which is by default stored in the **O** global variable.

C.2.1 Data component

Simulation in Yade is represented by *Bodies*, their *Interactions* and resultant *generalized forces* .

Bodies

Each Body comprises the following:

- **Shape :** represents particles geometry (neutral with regards to its spatial orientation), such as *Sphere*,*Facet* or infinite *Wall*; it usually does not change during simulation.
- **Material :** stores characteristics pertaining to mechanical behavior, such as Young's modulus or density,which are independent on particles shape and dimensions; usually constant, might be shared amongst multiple bodies
- **State :** contains state variables, in particular spatial *position* and *orientation*, *linear* and *angular velocity*;it is updated by the integrator at every step. The derived classes would contain other information related to current state of this body, e.g. its temperature, averaged damage or broken links between components.
- **Bound:** is used for approximate contact detection (refer chapter 4); updated as necessary following body's motion (refer chapter 4). Currently, Aabb(Axis-aligned bounding box) is used most often as Bound

(In addition to these 4 components, bodies have several more minor data associated, such as *Body::id* or *Body::mask*.)

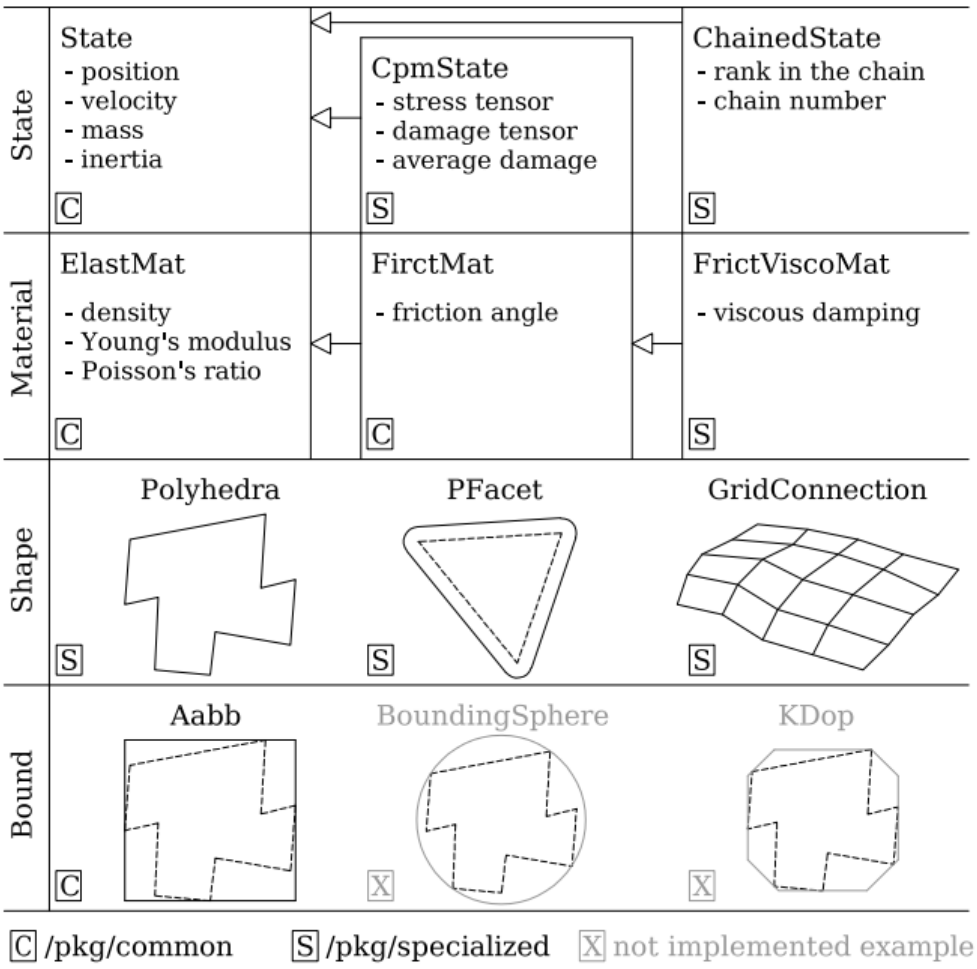


Figure C.1: classes used to describe a *Body: State, CpmState, ChainedState, Material, ElastMat, FrictMat, FrictViscoMat, Shape, Polyhedra, PFacet, GridConnection, Bound, Aabb.*

Interaction

Interactions are always between pair of bodies; usually, they are created by the collider based on spatial proximity; they can, however, be created explicitly and exist independently of distance. Each interaction has 2 components:

- **IGeom:** holding geometrical configuration of the two particles in collision; it is updated automatically as the particles in question move and can be queried for various geometrical characteristics, such as penetration distance or shear strain.

  Based on combination of types of Shapes of the particles, there might be different storage requirements; for that reason, a number of derived classes ex-

ists, e.g. for representing geometry of contact between Sphere+Sphere, Cylinder+Sphere etc. Note, however, that it is possible to represent many type of contacts with the basic sphere-sphere geometry (for instance in $Ig2_{W}all_{S}phere_{S}cGeom$)

- **IPhys** representing non-geometrical features of the interaction; some are computed from Materials of the particles in contact using some averaging algorithm (such as contact stiffness from Young's moduli of particles), others might be internal variables like damage.

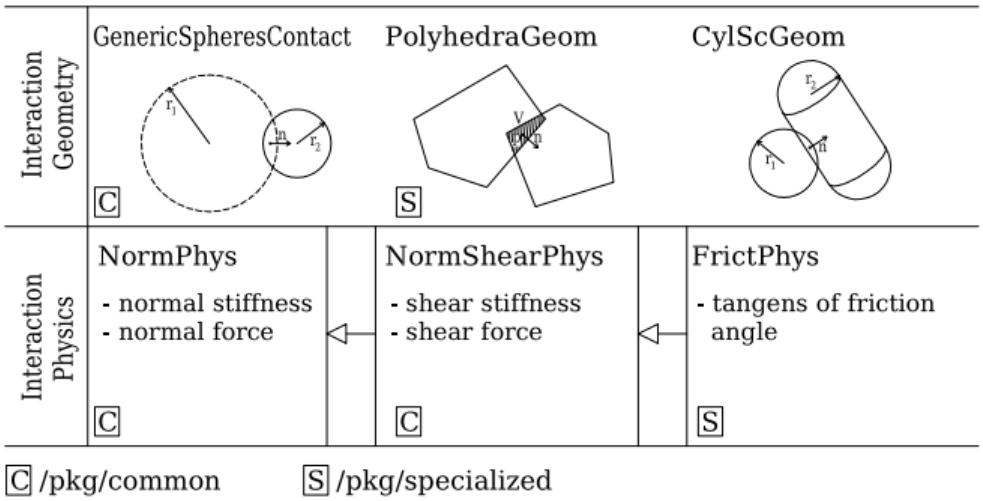


Figure C.2: classes used to describe an Interaction: IGeom, GenericSpheresContact, PolyhedraGeom, CylScGeom, IPhys, NormPhys, NormShearPhys, FrictPhys.

Generalized forces

Generalized forces include force, torque and forced displacement and rotation; they are stored only temporarily, during one computation step, and reset to zero afterwards. For reasons of parallel computation,they work as accumulators, i.e. only can be added to, read and reset.

C.2.2 Function component

In a typical DEM simulation, the following sequence is run repeatedly:

- reset forces on bodies from previous step

- approximate collision detection (pass 1)
- detect exact collisions of bodies, update interactions as necessary
- solve interactions, applying forces on bodies
- apply other external conditions (gravity, for instance).
- change position of bodies based on forces, by integrating motion equations.

Each of these actions is represented by an ***Engine***, functional element of simulation. The sequence of engines is called ***simulation loop***.

There are 3 fundamental types of Engines:

- ***GlobalEngines*** operating on the whole simulation (e.g. ForceResetter which zeroes forces acting on bodies or *GravityEngine* looping over all bodies and applying force based on their mass
- ***PartialEngine*** operating only on some pre-selected bodies (e.g. *ForceEngine* applying constant force to some selected bodies)
- ***Dispatchers*** do not perform any computation themselves; they merely call other functions, represented by function objects, Functors. Each functor is specialized, able to handle certain object types, and will be dispatched if such obejct is treated by the dispatcher.

C.3 Yade Basic Commands

Running and saving simulation

As explained in the architecture overview a simulation in yade contain both data and function component . It is stored in single Scene object called singleton .Yade allow us to access this simulation object trough a Class called **Omega** .The Omega class is stored in **O** global variable.

The following commands are used to access,save and control any simulations in yade .

- The number of bodies in a current simulation is obtained using:

```
len(O.bodies) # len stands for length
```

- To run a simulation:

```
O.run()
```

- The current time-step can be achieved by:

```
O.dt
```

- To obtain the current iteration:

```
O.iter
```

- To pause a running simulation:

```
O.pause()
```

- To save simulation :

```
# saves to a specified directory  with file extension ".bz2"
O.save('/tmp/a.yade.bz2')
```

- To reload simulation

```
O.reload()
```

- To Load saved simulation from specified directories

```
# loads the simulation file "yade.bz2".
O.load('/tmp/yade.bz2')
```

- To open the Graphical user interface Controller :

```
yade.qt.Controller()
```

- The current position of a body can be achieved:

```
O.bodies[i].state.pos
```

- Positions of all bodies are printed to screen using the following python for loop commands :

```
for i in O.bodies:
  print (i.state.pos)
```

- current velocity of a body can be determined:

```
O.bodies[i].state.vel
```

- velocity of all bodies are printed to screen using the following python for loop commands :

```
for i in O.bodies:
  print (i.state.vel)
```

- The shape of a body

```
O.bodies[i].shape
```

- Material of a body is achieved by:

```
O.bodies[i].mat
```

- To Display the density of all material on screen

```
for i in O.bodies:
  print (i.mat.density)
```

- forces acting on i^{th} element

```
O.forces.f(i))
```

- Interaction between particle i and j

```
O.interactions[i,j]
```

C.4 Soil sample generation and Scene construction

in the hybrid finite-discrete element simulations the following elements involves :

- Discrete elements which consists of spheres ,facets ,clumps,walls and boxes.
- Finite elements brick element,quadrilateral element and shell element
- Interface elements consist of triangular and quadrilateral elements

A soil sample under investigation involves the above three elements , hence this section covers how to generate each elements in Yade

C.4.1 Discrete element generation

Sphere packing

Single sphere

yade is written in object oriented programming approach . Any discrete element objects (sphere,facets,walls etc) are instances of their ***abstract classes*** .The process of constructing these objects is called ***instantiation*** . For example to create sphere object, first we need to instantiate it from ***Sphere()*** class with the following command

```
s1=Sphere() # s1 is object instantiated from Sphere()class
s1.radius = 0.001 # set radius of a sphere to be 0.001
```

If we want to generate a soil sample for direct shear test ,we might need to define thousands or may be millions of sphere objects to fill shear box. It is impractical to define these spheres in the above method . Hence yade provides as with predefine functions to avoid such tasks . some of the functions are ***utils.sphere, utils.facet, utils.wall and etc ...*** please refer yade documentation(insert citation here) module references for further pre-defined functions .

```
# generate a sphere with center (0, 0, 0) and radius 0.001
s1 = utils.sphere(center = (0,0,0), radius = 0.001)
# add the generated particle to O.bodies
O.bodies.append(s1)
```

Packing

sphere packing is Representing a solid of an arbitrary shape by arrangement of spheres , i.e. spatial arrangement of spheres such that a given solid is approximately filled with.

Yade uses 2 methods for representing exact volume of the solid in question these are **boundary representation** and **constructive solid geometry**. Despite their fundamental differences, they are abstracted in Yade in the ***Predicate*** class. *Predicate* provides the following functionality:

1. defines axis-aligned bounding box for the associated solid

2. can decide whether given point is inside or outside the solid; most predicates can also (exactly or approximately) tell whether the point is inside and satisfies some given padding distance from the represented solid boundary (so that sphere of that volume doesn't stick out of the solid).

Constructive Solid Geometry (CSG)

CSG approach describes volume by geometric primitives or primitive solids (sphere, cylinder, box, cone,etc) and boolean operations on them. Primitives defned in Yade include *inCylinder, inSphere, inEllipsoid,inHyperboloid*.

For instance, *inCylinder* specimen for triaxial compression test can be constructed in this way (ref fig **??**):

```
from yade import pack
# construct the predicate first
pred=pack.inCylinder(centerBottom=(0,0,-.1),
    centerTop=(0,0,.1),radius=.05)

# pack the predicate with spheres
spheres=pack.randomDensePack(pred,spheresInCell=2000,radius=3.5e-3)

# add spheres to simulation
O.bodies.append(spheres)
```

There are functions to generate a specific arrangement of particles in the *pack* module; for instance, cloud(random loose packing) of spheres can be generated with the ***pack.SpherePack*** class:

- loose packing samples can be generated using **makeCloud()**,

```
from yade import pack
sp1=pack.SpherePack()
# Generate a cloud of spheres within a box given by
lower corner (0, 0, 0) and
# upper corner (1, 1, 1).
# Radii of particles follow uniform distribution between
# rMean*(1 - rRelFuzz) and rMean*(1 + rRelFuzz)


```

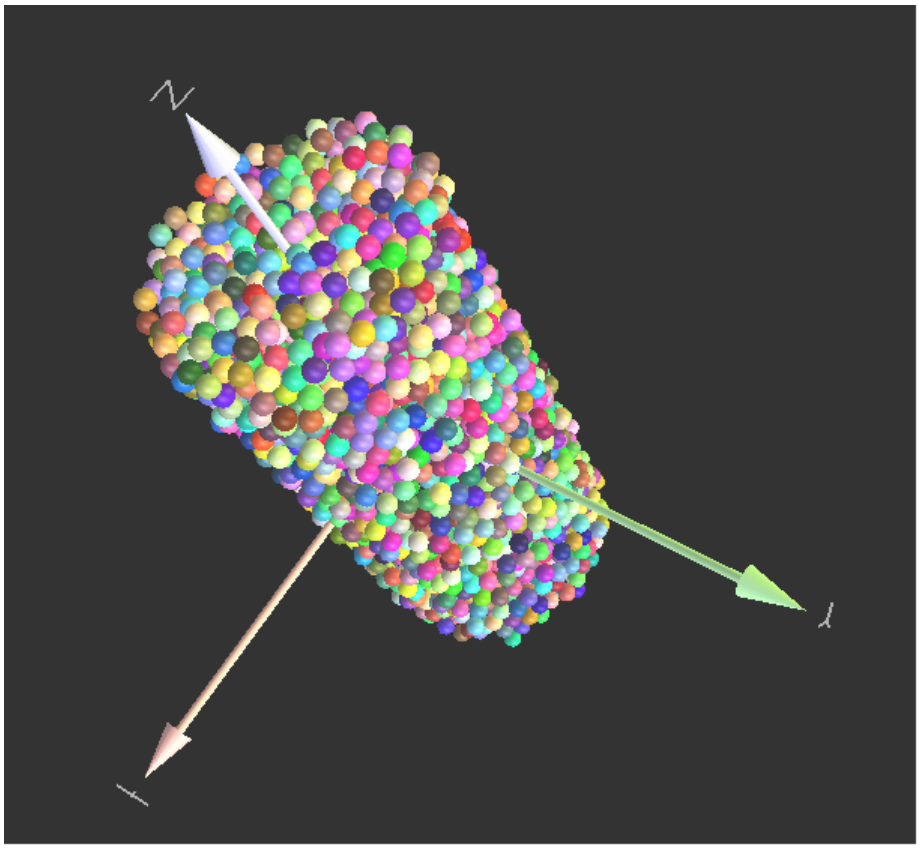


Figure C.3: packing using constructive solid Geometry pack.inCylinder()

```
sp1.makeCloud( minCorner = Vector3(0, 0, 0),
maxCorner = Vector3(1, 1, 1), rMean = 0.06, rRelFuzz = 0.2)

# add the generated particle to O.bodies
sp1.toSimulation()
or
O.bodies.append(sp1)
```

- A DE sample can be generated considering the particle size distribution, for example:

```
sieve = []
# add a sieve with particle size 0.0075 and % (in mass)
# of finer particles: 0%
sieve.append((0.0075,0))
sieve.append((0.01,85))
sieve.append((0.025,97))
sieve.append((0.03,100))
sp. makeCloud (minCorner = Vector3(0, 0, 0),
```

```
 maxCorner = Vector3(1, 1,1),
 sieveMass = sieve, targetPorosity = 0.39)

O.bodies.append([utils.sphere(s[0],s[1]) for s in sp])
```

Boolean operations on predicates

Boolean operations on pair of predicates (noted A and B) are defned:

- *intersection* **A** and **B** (conjunction): point must be in both predicates involved.
- *union* **A** | **B** (disjunction): point must be in the frst or in the second predicate.
- *difference* **A** − **B** (conjunction with second predicate negated): the point must be in the first predicate and not in the second one.
- *symmetric difference* **A** *symmetric* **B** (exclusive disjunction): point must be in exactly one of the two predicates.

For example, we can take box and remove cylinder from inside, using the **A** − **B** operation .

```
pred=pack.inAlignedBox((-2,-2,-2),(2,2,2))-
    pack.inCylinder((0,-2,0),(0,2,0),1)

spheres=pack.randomDensePack(pred,spheresInCell=2000,radius=.1,
rRelFuzz=.4,returnSpherePack=True)
```

Clumping particles together

Rigid particles can be created using *appendClumped()* as follows :

```
p1 = utils.sphere(center = (0, 0, 0), radius = 0.005)
p2 = utils.sphere(center = (0, 0.006, 0), radius = 0.006)
O.bodies. appendClumped ([p1, p2])
```

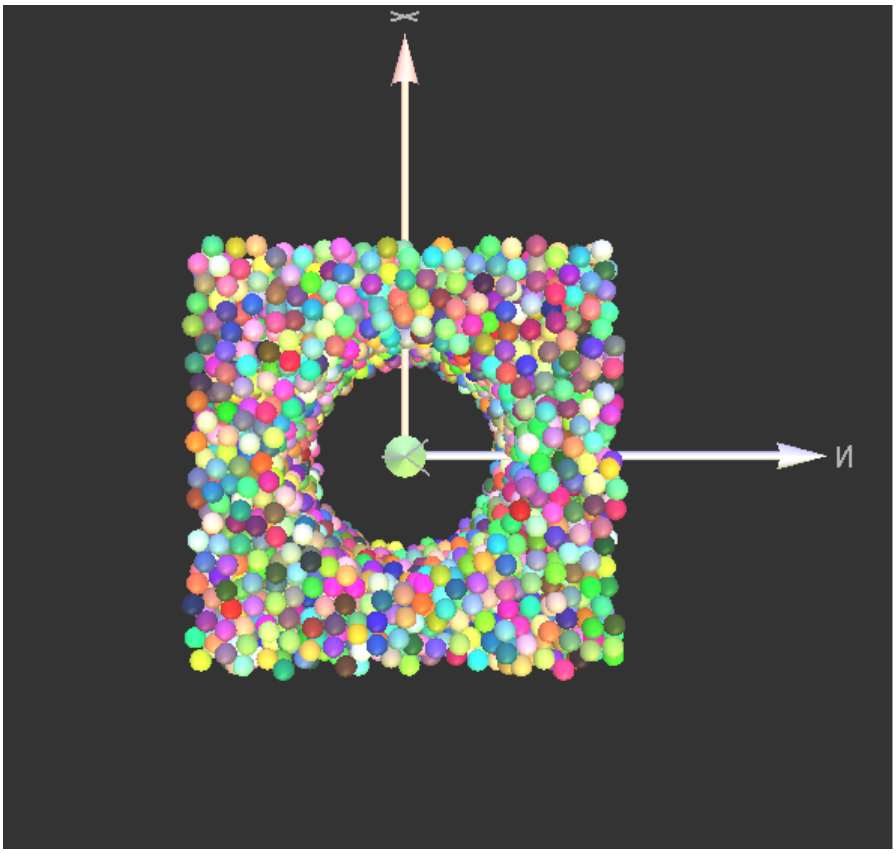


Figure C.4: Boolean operation on predicates in this case difference operation is executed

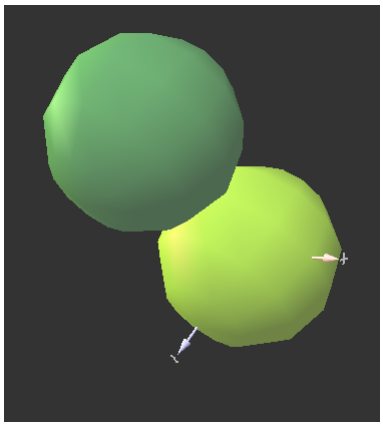


Figure C.5: clumped particles (spheres)

Creating Facets

facets can be created using *utils.facet()*function as follows :

```
# create a facet with three corners (0,0,0),(0,1,0),(0,0,1)
facet = utils.facet([(0,0,0),(0,1,0),(0,0,1)])
O.bodies. append (facet)
```

for further DE generation please refer yade documentation

Global engines

Global engines are the function component which operates on the whole simulation. DEM simulation in Yade does have at least the following Global engines (refer Function components for details):

1. Reset forces from previous step
2. Detect new collisions
3. Handle interactions
4. Apply forces and update positions of particles

```
O.engines=[ ForceResetter(), # reset forces
InsertionSortCollider([...]), # approximate collision detection
InteractionLoop([...],[...],[...]) # handle interactions
NewtonIntegrator() # apply forces and update positions
]
```

The order of engines is important. In majority of cases, you will put any additional engine after *InteractionLoop:*

- if it applies force, it should come before *NewtonIntegrator,* otherwise the force will never be effective.
- if it makes use of bodies positions, it should also come before *NewtonIntegrator*, otherwise, positions at the next step will be used (this might not be critical in many cases, such as output for visualization with VTKRecorder).

Partial engines

partial engines operate on some pre-selected simulation objects (bodies).It selects the bodies by their *id's* .

- *TranslationEngine* and *RotationEngine* for applying constant speed linear and rotational motion on subscribers.

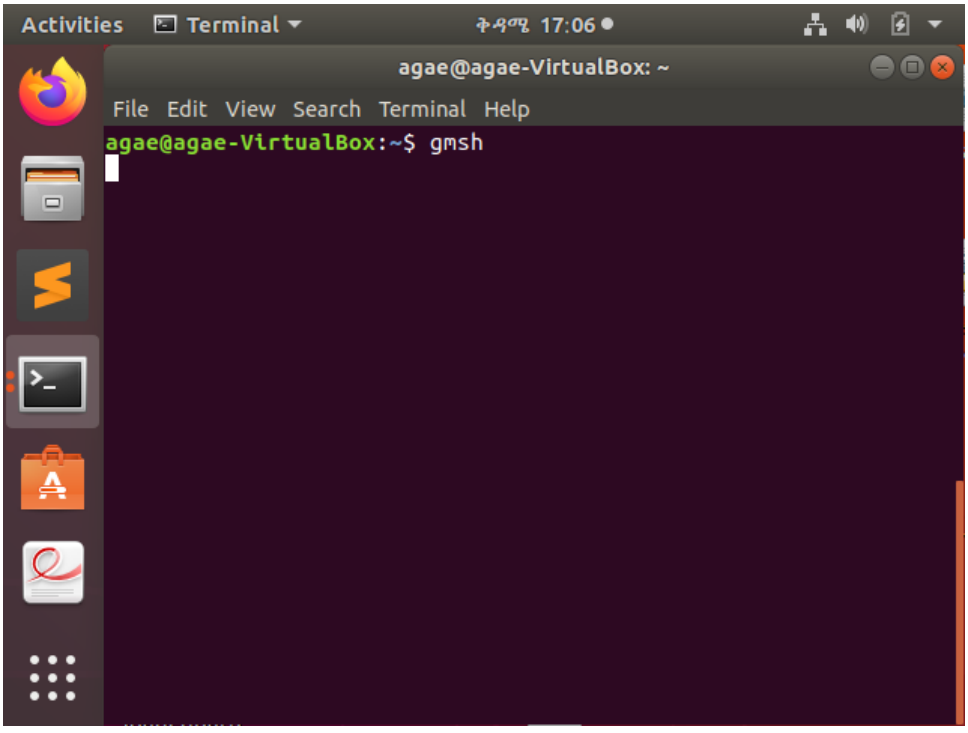
- *ForceEngine* and *TorqueEngine* applying given values of force/torque on subscribed bodies at every step .
- *StepDisplacer* for applying generalized displacement delta at every time step; designed for precise control of motion when testing constitutive laws on 2 particles.

C.4.2 Finite element generation

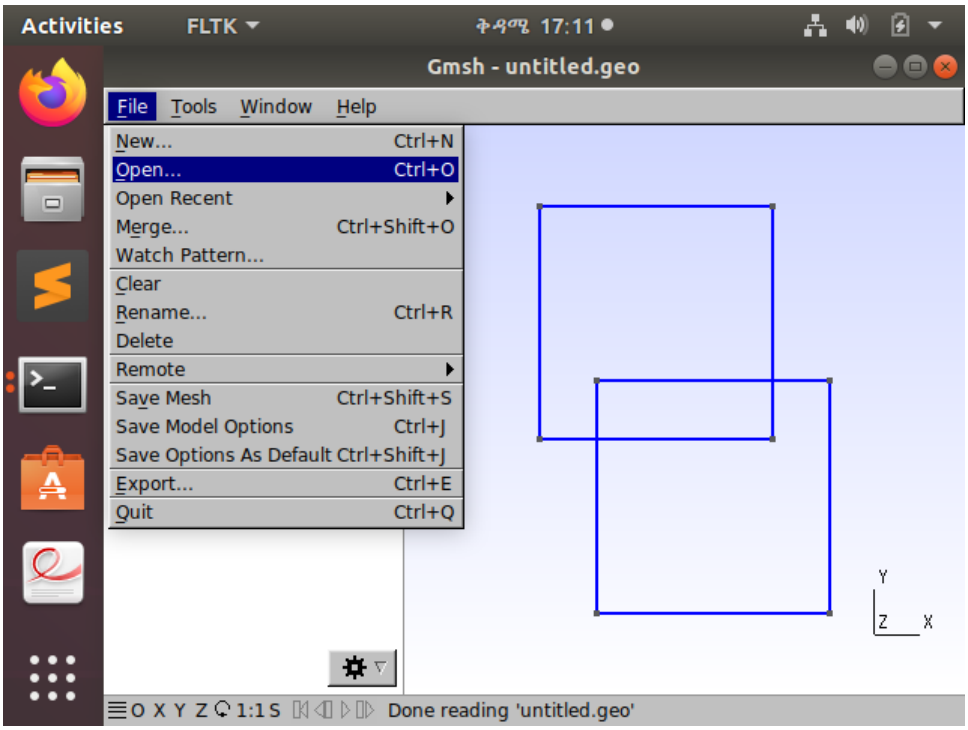
The finite element domain of the simulation is imported from external packages (software). yade built-in functions *yade.ymport()* enable us to import FE nodes ,elements and meshes from files exported from conventional FE software such Abaqus,Plaxis etc... . *yade.ymport()*built-in function is compatible with file extensions .txt,.ele,.mesh and other file formats .

Here is steps to import FE domain from Abaqus :

1. Model the FE domain in abaqus (Plese refer abaqus user manual).
2. Take the .stl file among the abaqus export files
3. open *gmsh* from the terminal (gmsh is finite element package used in Linux)

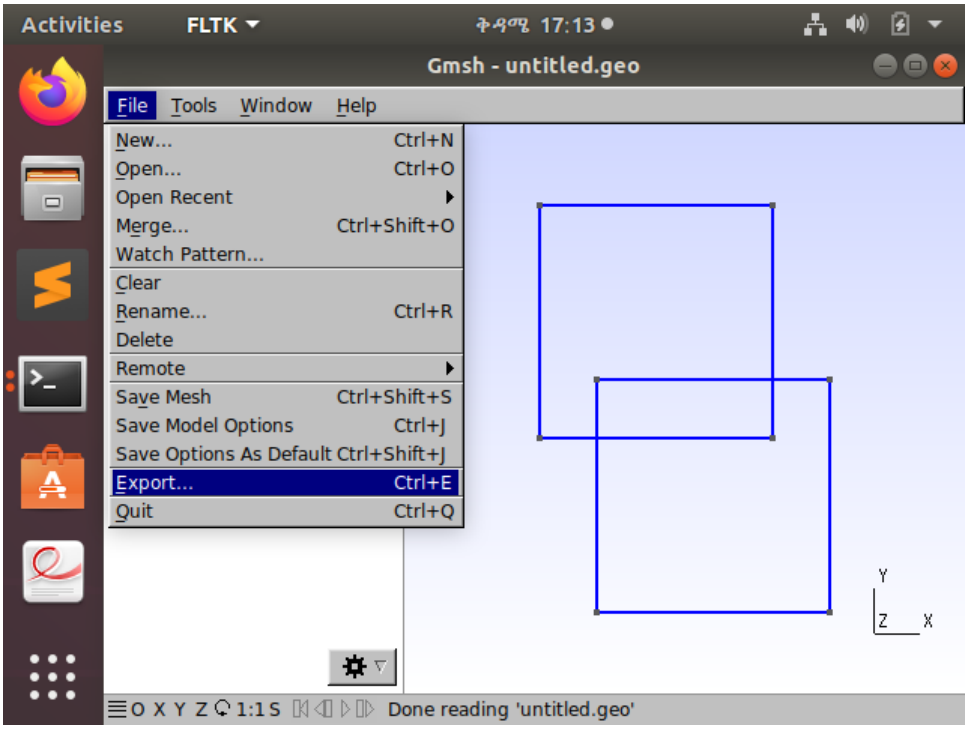


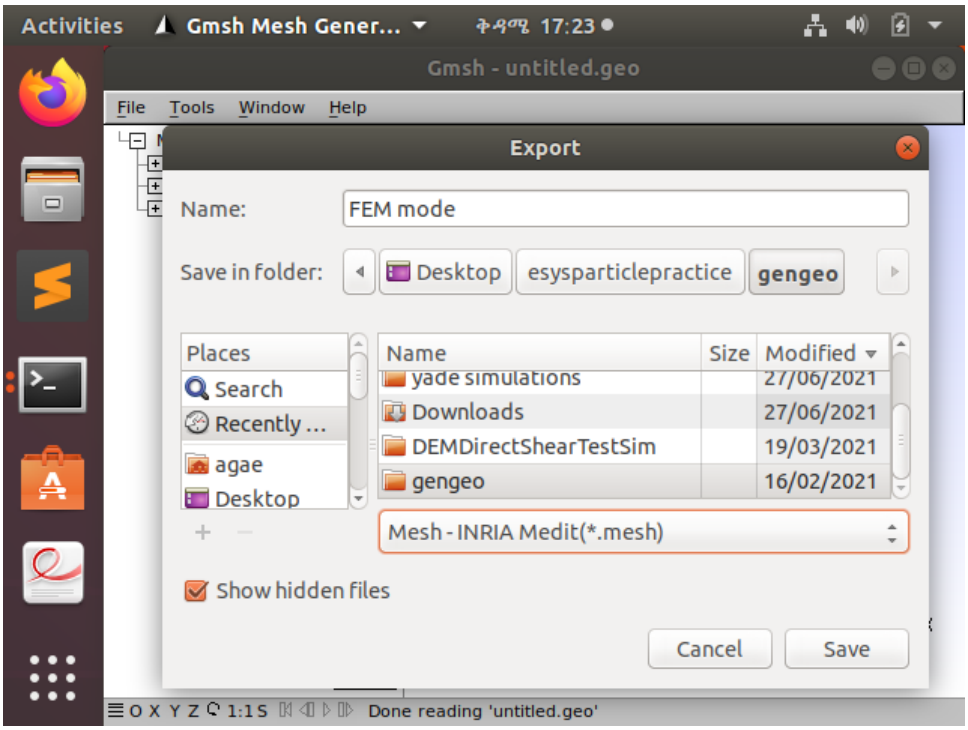
4. open the .stl file in gmsh .



5. export the .stl file to .mesh file format as shown in figure

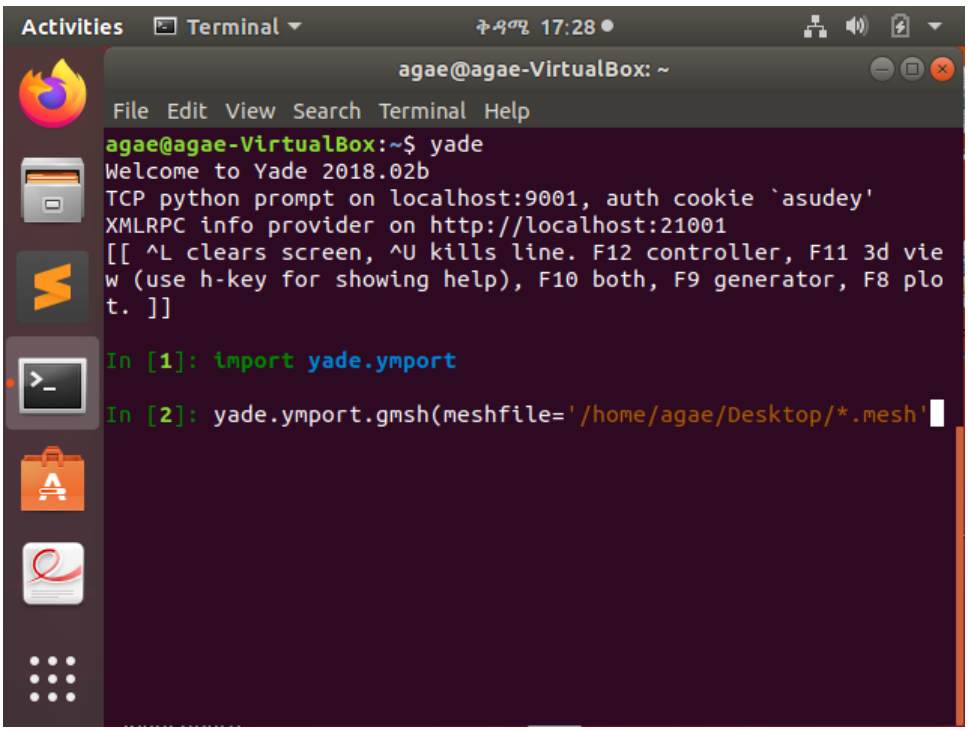
   File ⇒ Export ⇒ gmsh





6. finaly import the *.mesh file in yade using *yade.ymport()* built in function .

```
yade.ymport.gmsh("*.mesh")# replace * with the filename
    exported.
```



C.4.3 Interface element generation

Interface elements are used to assure the interaction between FE and DE domains.Yade uses facets as an interface element .Through integration with the GNU Triangulated surface library (GTS),yade provide variety of function for surface manipulation .GTS surfaces are geometrical objects, which can be inserted into simulation as set of particles whose *Body.shape* is of type *Facet* which is a single triangulation elements.p*ack.gtsSurface2Facets* can be used to convert GTS surface triangulation into list of *bodies* ready to be inserted into simulation via *O.bodies.append*.

Facet particles are created by default as non-*Body.dynamic* (they have zero inertial mass). That means that they are fixed in space and will not move if subject to forces. Yade currently offers 3 formats for importing triangulated surfaces from external files, in the *yade.ymport()* module:

- ***ymport.gts*** text fle in native GTS format.
- ***ymport.stl*** STereoLitography format, in either text or binary form; exported from Blender, but from many CAD systems as well.
- ***ymport.gmsh.*** text file in native format for GMSH, popular open-source meshing program.

C.4.4 boundary conditions

Motion Constraints

Motions of simulation objects can be restrained in several ways

- ***Body.dynamic*** determines whether a body will be accelerated by *NewtonIntegrator*; it is mandatory to make it *false* for bodies with zero mass, where applying non-zero force would result in infinite displacement.

  Facet makes them non-dynamic by default, as they have zero volume and zero mass (this can be changed, by passing *dynamic=True* to facet or setting *Body.dynamic*;setting *State.mass* to a non-zero value must be done as well). The same is true for *wall*.

  for Spheres use the following command:

```
utils.sphere([x, y, z), radius, dynamic = False )
```

- ***State.blockedDOFs*** permits selective blocking of any of 6 degrees of freedom in global space. For instance, a sphere can be made to move only in the xy plane by saying:

```
O.bodies.append(sphere((0,0,0),1))
O.bodies[0].state.blockedDOFs='zXY'
```

- Restrained conditions can also be achieved using engines such as *UniaxialStrainer()*, *TriaxialStressController()*, *PeriTriaxController()*, *ForceEngine()*, *RotationEngine()* and *TranslationEngine()*.

C.4.5 Controlling Simulation

pyrunner

A special engine ***PyRunner*** can be used to periodically call python code, specifed via the command parameter. Periodicity can be controlled by specifying computation time (*realPeriod*), virtual time (*virtPeriod*) or iteration number (*iterPeriod*).

For example, to print kinetic energy (using *kineticEnergy*) every 10 seconds, the following engine will be put to *O.engines*:

```
PyRunner(command = "print('kinetic energy',kineticEnergy())"
   ,realPeriod=10)
```

For running more complex commands, it is convenient to define an external function and only call it from within the engine. Since the *command* is run in the script's namespace, functions defined within scripts can be called. for instance if we want to check the unbalanced force periodicaly and pause the simulation if the unbalanced force is less than 0.01 we can write the following python code and run it using pyrunner engine by calling the function in command parameter as follows :

```
def checkUnbalanced():
   if unbalancedForce()<.05:
   O.pause()

O.engines=[...,
PyRunner(command="checkUnbalanced()",realPeriod=10)]
```

If a function was declared inside a live yade session (ipython) then an error NameError: name 'checkUnbalanced' is not defined will occur unless python *globals()* are updated with command:

```
globals().update(locals())
```

Plotting Variables

To plot variables ,we must import plot module using the following command .

```
from yade import plot
```

The plot module provides simple interface and storage for tracking various data. Although originally conceived for plotting only, it is widely used for tracking variables in general.

The data are in *plot.data* dictionary, which maps variable names to list of their values; the *plot.addData* function is used to add them. initially plot.data is an empty Dictionary object,To add data to it we use function plot.addData(). For example

```
plot.addData(sigma=12,xx=1e-4)
```

The above code will create a dictionary object *'sigma': [12,],'xx': [0.0001]* and add the data to *plot.data* . if we insert additional variables and data as shown in the following command

```
plot.addData(xx=1e-3)
plot.addData(force=1e3)
```

The new record is added to all columns at every time *plot.addData* is called; this assures that lines in different columns always match. The special value *nan or NaN* (Not a Number) is inserted to mark the record invalid.Hence following the newly added data and variables the plot.data object will be *'sigma': [12, nan, nan],'xx': [0.0001, 0.001, nan],'force': [nan, nan, 1000.0],*. if we want to retrieve only one column for instance

```
print(plot.data['xx'])
#This command will print out [0.0001, 0.001, nan]
```

Note: It is not possible to have two columns with the same name, since data are stored as a dictionary

To record data periodically, use *PyRunner*. For example to record the velocity of a body with id 1 in z direction in position z coordinate every 10 iteration,The following command will be used:

```
O.engines=O.engines+[PyRunner(command='myAddData()',iterPeriod=10)]
from yade import plot
def myAddData():
  b=O.bodies[1]
  plot.addData(z1=b.state.pos[2],v1=b.state.vel[2],t=O.time)
```

now that we have tracked variables and store in ***plot.data*** ,we can actually plot the variables in **x** and **y** plane .The ***plot.plots*** dictionary is a simple specification of plots. Keys are **x-axis** variable, and values are **tuple of y-axis** variables, given as strings that were used for *plot.addData*; each entry in the dictionary represents a separate figure :

```
plot.plots={'i':('t',), # plot t(i)

      't':('z1','v1') # z1(t) and v1(t)
}
```

Actual plot using data in *plot.data* and plot specification of plot.plots can be triggered by invoking the *plot.plot* function.

To live update the plots during calculation ,the following setting is used :

- *plot.live* - By setting ***yade.plot.live=True*** you can watch the plot being updated while the calculations run. Set to **False** otherwise.
- ***plot.liveInterval*** - This is the interval in seconds between the plot updates.
- *plot.autozoom* - When set to **True** the plot will be automatically re zoomed.

C.4.6 Post-processing

There are different tools for post-processing of data and produce visual presentation .Among the available open-source packages ,*paraview* is one of a specialized post-processing tools for scientific simulations .

Paraview is based on the Visualization Toolkit, which defines formats for saving various types of data. One of them (*with the .vtu extension*) can be written by a special engine VTKRecorder. It is added to the simulation loop

```
O.engines=[
VTKRecorder(iterPeriod=100,recorders=
  ['spheres','facets','colors'],fileName='/tmp/p1-')
]
```

- *iterPeriod*determines how often to save simulation data (besides iterPeriod, you can also use *virtPeriod or realPeriod*). If the period is too high (and data are saved only few times), the video will have few frames.
- *fileName* is the prefix for files being saved. In this case, output files will be named */tmp/p1-spheres.0.vtu* and */tmp/p1-facets.0.vtu*, where the number is the number of iteration; many files are created, putting them in a separate directory is advisable.
- *recorders* determines what data to save

for more detail of using paraview please refer paraview documentation (https://www.paraview.org

APPENDIX D

PYTHON SCRIPTS

D.1 Packing python Scripts

D.1.1 Packing using YADE

```
#
   ##############################################################################

# simple Gravity packig Demo  Script
# # Author: Agegnehu Gizaw
# Advisor: Dr Ing. Tensaye GebreMedhin
# Date: August 2021
# Addis Ababa Institute of Technology ,Addis Ababa University
#
#
#
   ##############################################################################


from yade import pack, plot

O.bodies.append(geom.facetBox((.5,.5,.5),(.5,.5,.5),wallMask=31))

sp=pack.SpherePack()

sp.makeCloud((0,0,0),(1,1,1),rMean=.05,rRelFuzz=.5)
```

```
18
19 sp.toSimulation()
20
21 O.engines=[ForceResetter(),InsertionSortCollider([Bo1_Sphere_Aabb()
    ,Bo1_Facet_Aabb()]),InteractionLoop([Ig2_Sphere_Sphere_ScGeom(),
    Ig2_Facet_Sphere_ScGeom()],
22 [Ip2_FrictMat_FrictMat_FrictPhys()],[
    Law2_ScGeom_FrictPhys_CundallStrack()]),NewtonIntegrator(gravity
    =(0,0,-9.81),damping=0.4,label ='grav'),
23 PyRunner(command='checkUnbalanced()',realPeriod=2),PyRunner(command
    ='addPlotData()',iterPeriod=100),PyRunner(command='printUf()',
    realPeriod=2),PyRunner(command='chkUf()',iterPeriod=1000)]
24
25 O.dt=.5*PWaveTimeStep()
26
27
28 O.trackEnergy=True
29
30 def checkUnbalanced():
31   if unbalancedForce()<.05:
32     O.pause()
33       plot.saveDataTxt('bbb.txt.bz2')
34
35
36
37 def addPlotData():
38   plot.addData(i=O.iter,uf=unbalancedForce(),**O.energy)
39
40 def printUf():
41   print("The unbalanced Force is :" , unbalancedForce())
42
43 def chkUf():
44   if unbalancedForce()<.2:
45     grav.damping =0.8
46   elif unbalancedForce()<.1:
47     O.pause()
48     print("The unbalanced Force is less than 0.1 :")
49
50
```

```

plot.plots={'i':('uf',None,O.energy.keys),'uf':('kinetic')}
plot.plot()
O.saveTmp()
```

```
#
    ################################################################################

# The scritp demonstrate how to generate sphere packing based on
    arbitrary PSD (particle size distribution)
# show the difference between size-based and mass-based (volume-
    based in our case) PSD.
#
# Author: Agegnehu Gizaw
# Advisor: Dr Ing. Tensaye GebreMedhin
# Date: August 2021
# Addis Ababa Institute of Technology ,Addis Ababa University
#
#
#
    ################################################################################


import matplotlib; matplotlib.rc('axes',grid=True)
from yade import pack
import pylab
# PSD given as points of piecewise-linear function

psdSizes,psdCumm
    =[.02,0.04,0.045,.05,.06,.08,.12],[0.,0.1,0.3,0.3,.3,.7,1.]
pylab.plot(psdSizes,psdCumm,label='precribed mass PSD')
sp0=pack.SpherePack();
sp0.makeCloud((0,0,0),(1,1,1),psdSizes=psdSizes,psdCumm=psdCumm,
    distributeMass=True)
sp1=pack.SpherePack();
sp1.makeCloud((0,0,0),(1,1,1),psdSizes=psdSizes,psdCumm=psdCumm,
    distributeMass=True,num=5000)
sp2=pack.SpherePack();
sp2.makeCloud((0,0,0),(1,1,1),psdSizes=psdSizes,psdCumm=psdCumm,
```

```
    distributeMass=True,num=20000)
pylab.semilogx(*sp0.psd(bins=30,mass=True),label='Mass PSD of (free
    ) %d random spheres'%len(sp0))
pylab.semilogx(*sp1.psd(bins=30,mass=True),label='Mass PSD of (
    imposed) %d random spheres'%len(sp1))
pylab.semilogx(*sp2.psd(bins=30,mass=True),label='Mass PSD of (
    imposed) %d random spheres (scaled down)'%len(sp2))

pylab.legend()

# uniform distribution of size (sp3) and of mass (sp4)

sp3=pack.SpherePack(); sp3.makeCloud((0,0,0),(1,1,1),rMean=0.03,
    rRelFuzz=2/3.,distributeMass=False);
sp4=pack.SpherePack(); sp4.makeCloud((0,0,0),(1,1,1),rMean=0.03,
    rRelFuzz=2/3.,distributeMass=True);
pylab.figure()
pylab.plot(*(sp3.psd(mass=True)+('g',)+sp4.psd(mass=True)+('r',)))
pylab.legend(['Mass PSD of size-uniform distribution','Mass PSD of
    mass-uniform distribution'])

pylab.figure()
pylab.plot(*(sp3.psd(mass=False)+('g',)+sp4.psd(mass=False)+('r',))
    )
pylab.legend(['Size PSD of size-uniform distribution','Size PSD of
    mass-uniform distribution'])
pylab.show()

pylab.show()
```

```
#
    ##############################################################################

# This script will pack particles base on Multi layer under
    compacton packig algortihgm
# particle is created with pre defined particle size distribution
# The packed particle can  be used for any simulation .
#
# Author: Agegnehu Gizaw
```

```
# Advisor: Dr Ing. Tensaye GebreMedhin
# Date: August 2021
# Addis Ababa Institute of Technology ,Addis Ababa University
#
#
#
    ##############################################################################
```

```
import matplotlib; matplotlib.rc('axes',grid=True)
from yade import pack
import pylab


psdsizes=[ 0.00014,0.00015,0.0003,0.0006, 0.00118,
    0.00236,0.00475,0.005] #psdsize and psdcumm should be in
    assending order.
psdCumm=[0.,0.0207,0.0717,0.2317,0.647,0.8803,0.9899,1] #psd cumm
    starts with o and ends with 1
idsandMat=O.materials.append(FrictMat(young = 1E7, poisson = 0.25,
    frictionAngle = 0.6,density = 2650,label='sand'))
facetmat=O.materials.append(FrictMat(young = 1E7, poisson = 0.25,
    frictionAngle = 0.6,density = 7480,label='famat'))
fa=geom.facetBox((.03,.0375,.03),(.03,.0375,.03),wallMask=63)
O.bodies.append(fa)

pylab.plot(psdsizes,psdCumm,label='precribed mass PSD')
sp0=pack.SpherePack();
sp0.makeCloud(minCorner=(0.0,0,0.0),maxCorner=(0.06,0.075,0.06),
    psdSizes=psdsizes,psdCumm=psdCumm,porosity=0.25)
sp0.toSimulation()
pylab.semilogx(*sp0.psd(bins=30,mass=True),label='Mass PSD of (free
    ) %d random spheres'%len(sp0))

#Sphere factory
R=0
Rmax=0
numSpheres=0
```

```
for o in O.bodies:
  if isinstance(o.shape,Sphere):
    o.mat.young=1e7
    o.mat.poisson = 0.2
    o.mat.frictionAngle = 0.6
    o.mat.density = 2650
    o.shape.color=(0.7,0.5,0.3)
    numSpheres+=1
    R+=o.shape.radius
    if o.shape.radius>Rmax:
      Rmax=o.shape.radius
Rmean=R/numSpheres
print("Rmean is ",Rmean,"numSpheres = " ,numSpheres)

#engines

O.engines=[ForceResetter(),
# sphere, facet, wall
  InsertionSortCollider([Bo1_Sphere_Aabb(),Bo1_Facet_Aabb()]),
  InteractionLoop(
  # the loading plate is a wall, we need to handle sphere+sphere,
    sphere+facet, sphere+wall
    [Ig2_Sphere_Sphere_ScGeom(),Ig2_Facet_Sphere_ScGeom()],
    [Ip2_FrictMat_FrictMat_FrictPhys()],
    [Law2_ScGeom_FrictPhys_CundallStrack()]),
    NewtonIntegrator(gravity=(0,-9.81,0),damping=0.6,label='gravity
    '),
# the label creates an automatic variable referring to this engine
# we use it below to change its attributes from the functions
    called
PyRunner(command='check()',realPeriod=2,label='checker'),]

O.dt=1e1*PWaveTimeStep()

# the following checkUnbalanced, unloadPlate and stopUnloading
    functions are all called by the 'checker'
# (the last engine) one after another; this sequence defines
    progression of different stages of the
# simulation, as each of the functions, when the condition is
```

```
    satisfied, updates 'checker' to call
# the next function when it is run from within the simulation next
    time
# check whether the gravity deposition has already finished

def check():

  if unbalancedForce() < 0.001 :
    i=1
    topsphere_pos = max([b.state.pos[2]+b.shape.radius for b in O.
   bodies if isinstance(b.shape,Sphere)])
    print ("the top spher position is ",topsphere_pos)
    if topsphere_pos < 0.025:
      print("round",i ,"packing is completed")
      print("round ",i+1, "packing process is started")

      sp=pack.SpherePack();
      sp.makeCloud(minCorner=(0.0,0.05,0.0),maxCorner
   =(0.06,0.075,0.06),psdSizes=psdsizes,psdCumm=psdCumm,
   distributeMass=True)
      sp.toSimulation()
      O.bodies.append(wall(max([b.state.pos[2]+b.shape.radius for b
    in O.bodies if isinstance(b.shape,Sphere)]),axis=2,sense=-1))
      global plate
      plate=O.bodies[-1]
      plate.state.vel=(0,0,-.1)
      O.engines=O.engines+[PyRunner(command='addPlotData()',
   iterPeriod=200)]
      checker.command='unloadPlate()'

    else:

      print("gravity packing is completed ")



def unloadPlate():
# if the force on plate exceeds maximum load, start unloading
  if abs(O.forces.f(plate.id)[2])>maxLoad:
```

```
103     plate.state.vel*=-1
104 # next time, do not call this function anymore, but the next one (
    stopUnloading) instead
105     checker.command='stopUnloading()'
106 
107 def stopUnloading():
108 
109   if abs(O.forces.f(plate.id)[2])<minLoad:
110 # O.tags can be used to retrieve unique identifiers of the
    simulation
111 # if running in batch, subsequent simulation would overwrite each
    otheroutput files otherwise
112 # d (or description) is simulation description (composed of
    parameter values)
113 # while the id is composed of time and process number
114     plot.saveDataTxt(O.tags['d.id']+'.txt')
115     O.pause()
116 
117 def addPlotData():
118 
119   if not isinstance(O.bodies[-1].shape,Wall):
120     plot.addData(); return
121   Fz=O.forces.f(plate.id)[2]
122 
123   plot.addData(Fz=Fz,w=plate.state.pos[2]-plate.state.refPos[2],
    unbalanced=unbalancedForce(),i=O.iter)
124 
125 # besides unbalanced force evolution, also plot the displacement-
    force diagram
126 
127 plot.plots={'i':('unbalanced',),'w':('Fz',)}
128 plot.plot()
129 
130 O.saveTmp()
131 O.run()
132 
133 # when running with yade-batch, the script must not finish until
    the simulation is done fully
134 # this command will wait for that (has no influence in the non-
```

```
    batch mode)
135
136 waitIfBatch()
```

D.1.2 Packing using GenGeo

```
1 #
    ##############################################################################
2 #
3 # This script is a packig algortithm using GenGeo
4 # Author: Agegnehu Gizaw
5 # Advisor: Dr Ing. Tensaye GebreMedhin
6 # Date: 20 Feburary  2021
7 # Addis Ababa Institute of Technology ,Addis Ababa University
8 # The packed paraticles can be imported in YADE or ESyS-Particle
9 # YADE is package for Discrete Element Modelling.
10 # ESyS-Particle is python module for Discrete element modelling
    ...
11 #Developed by :- DESSCC, The University of Queensland, Brisbane,
    AUSTRALIA
12 #
13 #
    ##############################################################################
14
15 from gengeo import *
16 ## ..........Step 1 Setting up neighbour table ..............#####
17 # - input parameters --
18 # block dimensions in mm
19 xdim=60
20 ydim=60
21 zdim=25
22
23 # particle size range
24 minRadius = 0.1 #For The given particle size you can write a
    function which create a box take it as an assignment .
25 maxRadius = 0.4
26 # ---------------------
```

```
27
28 # corner points
29 minPoint       = Vector3(0.0,0.0,0.0)      # To create the upper
      and lower part of the shear cell
30 maxPoint       = Vector3(xdim,ydim,zdim)   # minPoint and
      maxPointathalf  is to create the lower part of the shearcell
31 minpointathalf = Vector3(0.0,0.0,zdim/2)   # minPointathalf is
      minpoint for the upper shearbox
32 maxPointathalf = Vector3(xdim,ydim,zdim/2) # maxPointathalf is
      maxpoint for the lower shearbox
33
34 # neighbour table
35 mntable = MNTable3D(
36 minPoint=minPoint,
37 maxPoint=maxPoint,
38 gridSize=2.5*maxRadius,
39 numGroups=1
40 )
41
42 # block volume #for  the whole box volume
43 box = BoxWithPlanes3D(
44 minPoint=minPoint,
45 maxPoint=maxPoint
46 )
47
48 # block volume #for  the volume the lower shear box
49 box1 = BoxWithPlanes3D(
50 minPoint=minPoint,
51 maxPoint=maxPointathalf
52 )
53 # block volume #for  the volume the lower shear box
54 box2 = BoxWithPlanes3D (
55 minPoint=minpointathalf,
56 maxPoint=maxPoint
57 )
58
59
60
61
```

```
62 # boundary planes
63 xzufPlane=Plane(minPoint,Vector3(0.0,1.0,0.0))  #the particles near
       this plane is tag 1
64 xzlfPlane=Plane(minPoint,Vector3(0.0,1.0,0.0))  #the particles near
       this plane is tag 2
65 xzubPlane=Plane(maxPoint,Vector3(0.0,-1.0,0.0)) #the particles near
       this plane is tag 3
66 xzlbPlane=Plane(maxPoint,Vector3(0.0,-1.0,0.0)) #the particles near
       this plane is tag 4
67
68 yzuRPlane=Plane(maxPoint,Vector3(-1.0,0.0,0.0)) #the particles near
       this plane is tag 5
69 yzlRPlane=Plane(maxPoint,Vector3(-1.0,0.0,0.0)) #the particles near
       this plane is tag 6
70 yzulPlane=Plane(minPoint,Vector3(1.0,0.0,0.0))  #the particles near
       this plane is tag 7
71 yzllPlane=Plane(minPoint,Vector3(1.0,0.0,0.0))  #the particles near
       this plane is tag 8
72
73 bottomPlane=Plane(minPoint,Vector3(0.0,0.0,1.0))#the particles near
       this plane is tag 9
74 topPlane=Plane(maxPoint,Vector3(0.0,0.0,-1.0))  #the particles near
       this plane is tag 10
75
76 '''
77 # boundary planes optional
78 xzufPlane=Plane(minpointathalf,Vector3(0.0,1.0,0.0))  #the
      particles near this plane is tag 1
79 xzlfPlane=Plane(minPoint,Vector3(0.0,1.0,0.0))      #the particles
      near this plane is tag 2
80 xzubPlane=Plane(maxPoint,Vector3(0.0,-1.0,0.0))     #the particles
      near this plane is tag 3
81 xzlbPlane=Plane(maxPointathalf,Vector3(0.0,-1.0,0.0)) #the
      particles near this plane is tag 4
82
83 yzuRPlane=Plane(maxPoint,Vector3(-1.0,0.0,0.0))     #the particles
      near this plane is tag 5
84 yzlRPlane=Plane(maxPointathalf,Vector3(-1.0,0.0,0.0)) #the
      particles near this plane is tag 6
```

```
85 yzulPlane=Plane(minpointathalf,Vector3(1.0,0.0,0.0))  #the
      particles near this plane is tag 7
86 yzllPlane=Plane(minPoint,Vector3(1.0,0.0,0.0))      #the particles
      near this plane is tag 8
87
88 bottomPlane=Plane(minPoint,Vector3(0.0,0.0,1.0))    #the particles
      near this plane is tag 9
89 topPlane=Plane(maxPoint,Vector3(0.0,0.0,-1.0))      #the particles
      near this plane is tag 10
90
91 '''
92 # add them to the box
93 box2.addPlane(xzufPlane)
94 box1.addPlane(xzlfPlane)
95 box2.addPlane(xzubPlane)
96 box1.addPlane(xzlbPlane)
97
98 box2.addPlane(yzuRPlane)
99 box1.addPlane(yzlRPlane)
100 box2.addPlane(yzulPlane)
101 box1.addPlane(yzllPlane)
102
103 box1.addPlane(bottomPlane)
104 box2.addPlane(topPlane)
105
106 # -- setup packer --
107 # iteration parameters
108 insertFails = 10000 #run the script by increaseing insertFail and
      see how the porosity of the sample chages .
109 maxIter = 10000
110 tol = 1.0e-9
111
112 # packer
113 packer = InsertGenerator3D(
114 minRadius=minRadius,
115 maxRadius=maxRadius,
116 insertFails=insertFails,
117 maxIterations=maxIter,
118 tolerance=tol,
```

```
seed=False
)
# pack particles into volume

packer.generatePacking(
volume=box,
ntable=mntable
)

# create bonds between neighbouring particles:
mntable.generateBonds(
groupID=0,
tolerance=1.0e-8,# this tolerence is usualy be larger than th
    tolerence stated in the packer.
bondID=0
)

# calculate and print the porosity:
volume = xdim*ydim*zdim

porosity = (volume - mntable.getSumVolume(groupID=0))/volume

print ("Porosity: ", porosity)
# write a geometry file

mntable.write("DEMDSTgen.geo", 1)

mntable.write("DEMDSTgen.vtu", 2)
```

D.2 Direct shear Test Simulation Engine

```
#
    ##########################################################################

# Direct shear Test simulation  Script
# # Author: Agegnehu Gizaw
# Advisor: Dr Ing. Tensaye GebreMedhin
# Date: August 2021
# Addis Ababa Institute of Technology ,Addis Ababa University
```

```
#
#
#
    ##############################################################################


##################### parameters definition

PACKING='X1Y1Z1_5k'
DAMPING=0.5
dtCOEFF=0.5
normalSTRESS=50e5
normalVEL=normalSTRESS/1e8 # 0.001 for 100kPa // optimized for
    normalVEL=normalSTRESS/1e8?
shearVEL=1*normalVEL # try different values to ensure quasi-static
    conditions
intR=1.263 #1.263 for X1Y1Z1_5k//  refer Bo1_Sphere_Aabb
DENS=3000
YOUNG=50e9
FRICT=18
ALPHA=0.3
TENS=45e5
COH=45e6
iterMAX=400000
Output='shearTest_'+PACKING+'
    _K10_D3000_Y50e9A03F18T45e5C45e6_500kPa_shearVel1'

#Defining materials
#material properties for the sand, material id is 0
idsandMat=O.materials.append(FrictMat(young = 1E7, poisson = 0.25,
    frictionAngle = 0.6,density = 2650,label='sand'))

#materials property for the shear box side walls  ,material id 1
idsideWallMat=O.materials.append(FrictMat(young = 1E7, poisson =
    0.25, frictionAngle = 0.0,density = 7480,label='sidewallmat'))

#material property for the shear box top and bottom walls, material
```

```
    id 2
idTopBotomWallMat=O.materials.append(FrictMat(young = 1E7, poisson
    = 0.25, frictionAngle = 0.8,density = 7480,label='topbotwallmat'
    ))

#shear box size x,y,z
#sample size
'''
x,y,z=0.06, 0.0125, 0.06

dstpredicate=pack.inAlignedBox(minAABB=(0,0,0),maxAABB=(x,2*y,z))


sp=pack.randomDensePack(dstpredicate,radius=0.001,material='sand')#
    if sphereincell param is given the cell will be periodic

O.bodies.append (sp)
'''

#or import from gengeo library

from yade.ymport import *
sp=yade.ymport.text('/home/agae/Desktop/yade simulations/
    sp160000poro0.1',material='sand')
O.bodies.append(sp)

## preprocessing to get dimensions
dim=utils.aabbExtrema()
xinf=dim[0][0]
xsup=dim[1][0]
X=xsup-xinf
yinf=dim[0][1]
ysup=dim[1][1]
Y=ysup-yinf
zinf=dim[0][2]
zsup=dim[1][2]
Z=zsup-zinf
## initial surface Area
S0=X*Z
```

```
73
74 print("The Area is ",S0,"number of sphere is ",len(O.bodies))
75
76
77 ## spheres factory
78 R=0
79 Rmax=0
80 numSpheres=0.
81 for o in O.bodies:
82  if isinstance(o.shape,Sphere):
83    o.shape.color=(0.7,0.5,0.3)
84    numSpheres+=1
85    R+=o.shape.radius
86    if o.shape.radius>Rmax:
87      Rmax=o.shape.radius
88 Rmean=R/numSpheres
89 print("Rmean is ",Rmean,"numSpheres = " ,numSpheres)
90
91 #Assign different color to the particles along x to observe the
    Deformation pattern
92
93 #this peace of code is for coloring along x in four parts to see
    deformation .
94  for o in O.bodies:
95          if o.state.pos[0] < ((xinf+(X/10))) :
96               o.shape.color=(0,1,0)
97          elif((o.state.pos[0] > ((xinf+(X/10)))) and ( o.state.pos
    [0] < ((xinf+(X/5))) )) :
98               o.shape.color= (1,0,0)
99          elif((o.state.pos[0] > ((xinf+(X/5)))) and ( o.state.pos
    [0] < ((xinf+(3*X/10))) )) :
100              o.shape.color=(0,0,1)
101         elif((o.state.pos[0] > ((xinf+(3*X/10)))) and ( o.state.
    pos[0] < ((xinf+(2*X/5))) )) :
102              o.shape.color=(2,0,1)
103         elif((o.state.pos[0] > ((xinf+(2*X/10)))) and ( o.state.
    pos[0] < ((xinf+(X/2))) )) :
104              o.shape.color=(0,1,1)
105         elif((o.state.pos[0] > ((xinf+(X/2)))) and ( o.state.pos
```

```
    [0] < ((xinf+(3*X/5))) )) :
106                 o.shape.color=(1,0,1)
107           elif((o.state.pos[0] > ((xinf+(3*X/5)))) and ( o.state.pos
    [0] < ((xinf+(7*X/10))) )) :
108                 o.shape.color=(4,2,0)
109           elif((o.state.pos[0] > ((xinf+(7*X/10)))) and ( o.state.
    pos[0] < ((xinf+(4*X/5))) )) :
110                 o.shape.color=(1,2,1)
111           elif((o.state.pos[0] > ((xinf+(4*X/5)))) and ( o.state.pos
    [0] < ((xinf+(9*X/10))) )) :
112                 o.shape.color=(2,1,5)
113           else :
114                  o.shape.color=(0.7,0.5,0.3)
115
116 ###################### creation of shear box
117 thickness=Y/100
118 oversizeFactor=1.3
119
120 ### loading platens
121 O.bodies.append(utils.box(center=(xinf+X/2.,yinf-thickness+Rmean
    /10.,zinf+Z/2.),extents=(oversizeFactor*X/2,thickness,
    oversizeFactor*Z/2),material='topbotwallmat',fixed=True))
122 bottomPlate=O.bodies[-1]
123 O.bodies.append(utils.box(center=(xinf+X/2.,ysup+thickness-Rmean
    /10.,zinf+Z/2.),extents=(oversizeFactor*X/2,thickness,
    oversizeFactor*Z/2),material='topbotwallmat',fixed=True))
124 topPlate=O.bodies[-1]
125
126
127 ###################### Engines
128 O.engines=[
129
130  ForceResetter(),
131  InsertionSortCollider([Bo1_Sphere_Aabb(aabbEnlargeFactor=intR,
    label='aabb'),Bo1_Box_Aabb()]),
132  InteractionLoop(
133   [Ig2_Sphere_Sphere_ScGeom(interactionDetectionFactor=intR,label='
    ss2d3dg'),Ig2_Box_Sphere_ScGeom()],
134   [Ip2_FrictMat_FrictMat_FrictPhys( frictAngle = 0.6,label='
```

```
    interactionPhys')],
135  [Law2_ScGeom_FrictPhys_CundallStrack(label='interactionLaw')]
136 ),
137
138 GlobalStiffnessTimeStepper(defaultDt=0.1*PWaveTimeStep(),
    timestepSafetyCoefficient=dtCOEFF),
139 TranslationEngine(ids=[topPlate.id],translationAxis=(0.,-1.,0.),
    velocity=0.,label='yTranslation'),
140 PyRunner(iterPeriod=1,initRun=True,command='servoController()',
    label='servo'),
141 NewtonIntegrator(damping=DAMPING,gravity=(0,0,0),label='damper'),
142 PyRunner(iterPeriod=iterMAX/1000,initRun=True,command='
    dataCollector()'),
143 ]
144
145 ###################### Engines definition ( servoController() and
    dataCollector() )
146 shearing=False
147 sigmaN=0
148 tau=0
149 Fs1=0
150 Fs2=0
151 Xdispl=0
152 px0=0
153 Ydispl=0
154 py0=topPlate.state.pos[1]
155 prevTranslation=0
156 n=0
157
158 def servoController():
159     global px0, py0, sigmaN, n, Fn1, Fn2, shearing, butee, piston1,
    piston2
160     Fn1=abs(O.forces.f(topPlate.id)[1])
161     print ("Force is fn1",Fn1)
162     Fn2=abs(O.forces.f(bottomPlate.id)[1])
163     A=(S0-2*Xdispl*Z)
164     print("calculated Area =",A)
165     sigmaN=Fn1/(S0-2*Xdispl*Z) # necessary?
166     print("sigmanN",sigmaN)
```

```
167     #print (yTranslation.velocity, sigmaN)
168     if shearing==False:
169       if yTranslation.velocity<normalVEL:
170           yTranslation.velocity+=normalVEL/1000
171       if sigmaN>(0.9*normalSTRESS):
172           yTranslation.velocity=normalVEL*((normalSTRESS-sigmaN)/
    normalSTRESS)
173     if shearing==False and abs((normalSTRESS-sigmaN)/normalSTRESS)
    <0.01 and unbalancedForce()<0.1:
174         yTranslation.velocity=0
175
176     n+=1
177     if n>1000 and abs((sigmaN-normalSTRESS)/normalSTRESS)<0.01:
178       print ('stress on joint plane = ', utils.forcesOnPlane((X/2,Y
    /2,Z/2),(0,1,0))/S0)
179
180       ### add top box
181       O.bodies.append(utils.box(center =(xinf-thickness+Rmean
    /10,3*(ysup/4), zsup/2),extents=(thickness,Y/4,oversizeFactor*Z
    /2), material= 'sidewallmat',fixed=True))
182       butee = O.bodies[-1]
183       O.bodies.append(utils.box(center=(xsup+thickness-Rmean/10,3*(
    ysup/4),zsup/2),extents=(thickness,Y/4,oversizeFactor*Z/2),
    material='sidewallmat',fixed=True))
184       O.bodies.append(utils.box(center=(xsup/2,3*(ysup/4),zinf-
    thickness+Rmean/10),extents=(oversizeFactor*X/2,Y/4,thickness),
    material='sidewallmat',fixed=True,wire=True))
185       O.bodies.append(utils.box(center=(xsup/2,3*(ysup/4),zsup+
    thickness-Rmean/10),extents=(oversizeFactor*X/2,Y/4,thickness),
    material='sidewallmat',fixed=True,wire=True))
186       ### add bottom box
187       O.bodies.append(utils.box(center=(xsup+thickness-Rmean/10,(
    ysup/4),zsup/2),extents=(thickness,Y/4,oversizeFactor*Z/2),
    material='sidewallmat',fixed=True))
188       piston1=O.bodies[-1]
189       O.bodies.append(utils.box(center=(xinf-thickness+Rmean/10,(
    ysup/4),zsup/2),extents=(thickness,Y/4,oversizeFactor*Z/2),
    material='sidewallmat',fixed=True))
190       piston2=O.bodies[-1]
```

```
191     O.bodies.append(utils.box(center=(xsup/2,(ysup/4),zinf-
    thickness+Rmean/10),extents=(oversizeFactor*X/2,Y/4,thickness),
    material='sidewallmat',fixed=True,wire=True))
192     O.bodies.append(utils.box(center=(xsup/2,(ysup/4),zsup+
    thickness-Rmean/10),extents=(oversizeFactor*X/2,Y/4,thickness),
    material='sidewallmat',fixed=True,wire=True))
193     ### add engine and initialisation
194
195     O.engines=O.engines[:6]+ [TranslationEngine(ids=[piston1.id,
    piston2.id],translationAxis=(-1.,0.,0.),velocity=0.)]+O.engines
    [6:]
196     px0=piston1.state.pos[0]
197     py0=topPlate.state.pos[1]
198     shearing=True
199     print ('shearing started ! || iteration=', O.iter)
200
201   if shearing==True:
202     #yTranslation.velocity=0. # if you want a constant normal
    diaplacement control
203     yTranslation.velocity=50*normalVEL*((normalSTRESS-sigmaN)/
    normalSTRESS) # not good enough because not general (coeff needs
     to be adjusted)
204     if O.engines[6].velocity<shearVEL:
205       O.engines[6].velocity+=(shearVEL/1000)
206
207
208 def dataCollector():
209   global Xdispl, Ydispl, tau, Fs1, Fs2
210   if shearing:
211     Fs1=abs(O.forces.f(butee.id)[0])
212     Fs2=abs(O.forces.f(piston1.id)[0])
213     Xdispl=abs(piston1.state.pos[0]-px0)
214   tau=Fs1/(S0-2*Xdispl*Z)
215   Ydispl=abs(topPlate.state.pos[1]-py0)
216   yade.plot.addData({'t':O.time,'i':O.iter,'Xdispl':Xdispl,'
    sigmaN':sigmaN,'Fn1':Fn1,'Fn2':Fn2,'tau':tau,'Fs1':Fs1,'Fs2':Fs2
    ,'Ydispl':Ydispl,'unbF':utils.unbalancedForce(),'Ek':utils.
    kineticEnergy(),'bonds':numberOfCohesiveBonds})
217   plot.saveGnuplot(Output)
```

```
218
219 # defines window plot
220 from yade import plot
221 plot.plots={'i':('sigmaN','tau')}
222
223 ################## graphical intervace
224 from yade import qt
225 qt.Controller()
226 qt.View()
227 v=qt.Renderer()
228 v.dispScale=(1,1,1) # displacements are scaled for visualization
       purpose
229
230
231 ################## to manage interaction detection factor during the
       first timestep
232 O.step();
233 ################## initializes the interaction detection factor to
       its default value (new contacts will be created between strictly
       contacting particles only)
234 ss2d3dg.interactionDetectionFactor=-1.
235 aabb.aabbEnlargeFactor=-1.
236
237 ################## simulation starts here
238 O.saveTmp()
239 O.run(iterMAX)
```

```
1
2
3 from __future__ import print_function
4 O.periodic=True
5 O.cell.refSize=(2,2,2)#O.cell.setBox=(2,2,2)
6
7 from yade import pack,plot
8
9 # the "if 0:" block will be never executed, therefore the "else:"
       block will be
10 # to use cloud instead of regular packing, change to "if 1:" or
       something similar
```

```
11
12 if 0:
13   # create cloud of spheres and insert them into the simulation
14   # we give corners, mean radius, radius variation
15   sp=pack.SpherePack()
16   sp.makeCloud((0,0,0),(2,2,2),rMean=.1,rRelFuzz=.1,periodic=True)
17   # insert the packing into the simulation
18   sp.toSimulation(color=(0,0,1)) # pure blue
19 else:
20   # in this case, add dense packing
21
22   O.bodies.append(pack.regularHexa(pack.inAlignedBox((0,0,0)
       ,(2,2,2)),radius=.1,gap=0,color=(0,0,1)))
23 # create "dense" packing by setting friction to zero initially
24
25 O.materials[0].frictionAngle=0
26
27 # simulation loop (will be run at every step)
28
29 O.engines=[ ForceResetter(),InsertionSortCollider([Bo1_Sphere_Aabb
       ()]),InteractionLoop([Ig2_Sphere_Sphere_ScGeom()],
30 [Ip2_FrictMat_FrictMat_FrictPhys()],[
       Law2_ScGeom_FrictPhys_CundallStrack()]),NewtonIntegrator(damping
       =.4),
31 # run checkStress function (defined below) every second the label
       is arbitrary, and is used later to refer to this engine
32
33 PyRunner(command='checkStress()',realPeriod=1,label='checker'),
34 # record data for plotting every 100 steps; addData function is
       defined below
35
36 PyRunner(command='addData()',iterPeriod=100)]
37
38 # set the integration timestep to be 1/2 of the "critical" timestep
39
40 O.dt=.5*PWaveTimeStep()
41
42 # prescribe isotropic normal deformation (constant strain rate)
43 # of the periodic cell
```

```
44
45 O.cell.velGrad=Matrix3(-.1,0,0, 0,-.1,0, 0,0,-.1)
46 # when to stop the isotropic compression (used inside checkStress)
47
48 limitMeanStress=-5e5
49 # called every second by the PyRunner engine
50
51 def checkStress():
52
53   # stress tensor as the sum of normal and shear contributions
54   # Matrix3.Zero is the intial value for sum(...)
55   stress=sum(normalShearStressTensors(),Matrix3.Zero)
56   print('mean stress',stress.trace()/3.)
57   # if mean stress is below (bigger in absolute value)
     limitMeanStress, start shearing
58   if stress.trace()/3.> abs(limitMeanStress):
59     # apply constant-rate distorsion on the periodic cell
60     O.cell.velGrad=Matrix3(0,0,.1, 0,0,0, 0,0,0)
61     # change the function called by the checker engine
62     # (checkStress will not be called anymore)
63     checker.command='checkDistorsion()'
64     # block rotations of particles to increase tanPhi, if desired
65     # disabled by default
66     if 0:
67       for b in O.bodies:
68         # block X,Y,Z rotations, translations are free
69         b.state.blockedDOFs='XYZ'
70         # stop rotations if any, as blockedDOFs block accelerations
      realy
71         b.state.angVel=(0,0,0)
72     # set friction angle back to non-zero value
73     # tangensOfFrictionAngle is computed by the Ip2_* functor from
     material
74     # for future contacts change material (there is only one
     material for all particles
75
76     O.materials[0].frictionAngle=.5  # radians for existing
     contacts, set contact friction directly
77     for i in O.interactions: i.phys.tangensOfFrictionAngle=tan(.5)
```

```
78
79 # called from the 'checker' engine periodically, during the shear
       phase
80
81 def checkDistorsion():
82
83   # if the distorsion value is >.3, exit; otherwise do nothing
84   if abs(O.cell.trsf[0,2])>.5:
85     # save data from addData(...) before exiting into file
86     # use O.tags['id'] to distinguish individual runs of the same
       simulation
87     plot.saveDataTxt(O.tags['id']+'.txt')
88     # exit the program
89     #import sys
90     #sys.exit(0) # no error (0)
91     O.pause()
92
93 # called periodically to store data history
94
95 def addData():
96
97   # get the stress tensor (as 3x3 matrix)
98   stress=sum(normalShearStressTensors(),Matrix3.Zero)
99   # give names to values we are interested in and save them
100  plot.addData(exz=O.cell.trsf[0,2],szz=stress[2,2],sxz=stress
       [0,2],tanPhi=stress[0,2]/stress[2,2],i=O.iter)
101  # color particles based on rotation amount
102  for b in O.bodies:
103    # rot() gives rotation vector between reference and current
       position
104    b.shape.color=scalarOnColorScale(b.state.rot().norm(),0,pi/2.)
105
106 # define what to plot (3 plots in total)
107 ## exz(i), [left y axis, separate by None:] szz(i), sxz(i)
108 ## szz(exz), sxz(exz)
109 ## tanPhi(i)
110 # note the space in 'i ' so that it does not overwrite the 'i'
       entry
111
```

```
plot.plots={'i':('exz',None,'szz','sxz'),'exz':('szz','sxz'),'i ':('tanPhi',)}

# better show rotation of particles
Gl1_Sphere.stripes=True
# open the plot on the screen

plot.plot()

O.saveTmp()
```

D.3 Class Files

```
#
    ################################################################################

# This script is a class file created to apply force to a  wall
    that is incrementally moved a small distance...
# in order to maintain a constant prescribed net force acting on
    the wall.
# This class file will be imported in main.py module ....
# Author: Agegnehu Gizaw
# Advisor: Dr Ing. Tensaye GebreMedhin
# Date: 20 Feburary  2021
# Addis Ababa Institute of Technology ,Addis Ababa University
# ESyS-Particle is python module Developed by :- DESSCC, The
    University of Queensland, Brisbane, AUSTRALIA
#
#
    ################################################################################
```

```
12
13
14 #import the division module for compatibility between Python 2 and
      Python 3
15 from __future__ import division
16 #import the appropriate ESyS-Particle modules:
17 from esys.lsm import *
18 from esys.lsm.util import *
19
20
21 class ServoWallLoaderRunnable (Runnable):
22
23   def __init__ (self,LsmMpi=None,interactionName=None,force=Vec3
      (0,0,0),startTime=0,rampTime = 200):
24     """Subroutine to initialise the Runnable and store parameter
      values."""
25     Runnable.__init__(self)
26     self.sim = LsmMpi
27     self.interactionName = interactionName
28     self.force = force
29     self.dt = self.sim.getTimeStepSize()
30     self.rampTime = rampTime
31     self.startTime = startTime
32     self.Nt = 0
33
34   def run (self):
35     """
36     Subroutine to apply the force to a wall interaction. After self
      .startTime
37     timesteps, the force on the wall increases linearly over
38     self.rampTime timesteps until the desired wall force is
      achieved.
39     Thereafter the wall force is kept fixed.
40     """
41
42     if (self.Nt > self.startTime):
43       #compute the slowdown factor if still accelerating the wall:
44       if (self.Nt < (self.startTime + self.rampTime)):
```

```
        f = float(self.Nt - self.startTime) / float(self.rampTime)

      else:
        f = 1.0
      #compute the amount by which to move the wall this timestep:
      Dforce = Vec3(f*self.force[0],f*self.force[1],f*self.force
    [2])

      #instruct the simulation to apply the prescribed force to the
     wall:
      self.sim.applyForceToWall (self.interactionName, Dforce)

    self.Nt += 1
```

```
#
    ##############################################################################

# This script is a class file created to produce moving wall in
    the simulation.
# This class file will be imported in main.py module ....
# Author: Agegnehu Gizaw
# Advisor: Dr Ing. Tensaye GebreMedhin
# Date: 20 Feburary  2021
# Addis Ababa Institute of Technology ,Addis Ababa University
# ESyS-Particle is python module Developed by :- DESSCC, The
    University of Queensland, Brisbane, AUSTRALIA
#
#import the division module for compatibility between Python 2 and
    Python 3

from __future__ import divisio
#import the appropriate ESyS-Particle modules:
from esys.lsm import *
from esys.lsm.util import *

#This script implements a Runnable designed to move a wall at a
    specified
#speed. The Runnable also implements initial acceleration of the
    wall
```

```
#from zero to the desired speed as well as an optional initial idle
#period during which the wall does not move.

class WallLoaderRunnable(Runnable):

  def __init__(self,LsmMpi=None,wallName=None,vPlate=Vec3(0,0,0),
    startTime=0,rampTime = 200):
     """Subroutine to initialise the Runnable and store parameter
    values."""
     Runnable.__init__(self)
     self.sim = LsmMpi
     self.wallName = wallName
     self.Vplate = vPlate
     self.dt = self.sim.getTimeStepSize()
     self.rampTime = rampTime
     self.startTime = startTime
     self.Nt = 0

  def run (self):
     """
     Subroutine to move the specified wall. After self.startTime
     timesteps, the speed of the wall increases linearly over
     self.rampTime timesteps until the desired wall speed is
    achieved.
     Thereafter the wall is moved at that speed.
     """

     if (self.Nt >= self.startTime):
       #compute the slowdown factor if still accelerating the wall:
       if (self.Nt < (self.startTime + self.rampTime)):
         f = float(self.Nt - self.startTime) / float(self.rampTime)

       else:
         f = 1.0
       #compute the amount by which to move the wall this timestep:
       Dplate = Vec3(f*self.Vplate[0]*self.dt, f*self.Vplate[1]*self
    .dt, f*self.Vplate[2]*self.dt)

       #instruct the simulation to move the wall:
```

```
54          self.sim.moveWallBy (self.wallName, Dplate)
55      #count the number of timesteps completed thus far:
56      self.Nt += 1
```

BIBLIOGRAPHY

[1] Mroueh and Shahrour, "A full 3-d finite element analysis of tunneling–adjacent structuresinteraction," *Elsevier ,Computers and Geotechnics 30 (2003) 245–253*, 2002. [Online]. Available: www.elsevier.com/locate/compge.

[2] L.Zdravkovic, D.M.Potts, and H. S. John, "Modelling of a 3d excavation in finite element analysis," *Department of Civil and Environmental Engineering, Imperial College, London, UK ,Ge´otechnique 55, No. 7, 497–513*, 2005.

[3] S.Karthigeyan, Ramakrishna, and K. Rajagopal, "Influence of vertical load on the lateral response of piles in sand," *Elsevier,Computers and Geotechnics 33 (2006) 121–131*, 2006. DOI: doi:10.1016/j.compgeo.2005.12.002. [Online]. Available: http://www.elsevier.com/locate/compgeo.

[4] P. Cundall and O. Strack, "A discrete numerical model for granular assembelies," *Geotechnique*, 1979.

[5] Herten and Pulsfort, "Dem granular material and large deformation," 1999.

[6] Cui and O'Sullivan, "Macroscopic and microscopic responses of granular soil samples under direct shear condition," 2006.

[7] Guerrero, "Dem granular material and large deformation," 2006.

[8] P.Villard, B. Chevalier, B. L. Hello, and G. Combe, "Coupling between finite and discrete element methods for the modelling of earth structures reinforced by geosynthetic," *Elsevier,Computers and Geotechnics*, 2009. DOI: 10.1016/j.compgeo.2008.11.005. [Online]. Available: http://www.elsevier.com/locate/compgeo.

[9] A. Elmekati and U. E. Shamy, "A practical co-simulation approach for multi-scale analysis of geotechnical systems," *Elsevier,Computers and Geotechnics*, 2009. DOI: 10.1016/j.compgeo.2010.02.002. [Online]. Available: http://www.elsevier.com/locate/compgeo.

[10] H. K. Dang and M. A. Meguid, "An efficient finite–discrete element method for quasi-static nonlinear soil–structure interaction problems," *INTERNATIONAL JOURNAL FOR NUMERICAL AND ANALYTICAL METHODS IN GEOMECHANICS*, 2011. DOI: 10.1002/nag.1089. [Online]. Available: http://www.wileyonlinelibrary.com.

[11] A. Fakhimi, "A hybrid discrete–finite element model for numerical simulation of geomaterials," *Elsevier,Computers and Geotechnics*, 2009. DOI: 10.1016/j.compgeo.2008.05.004. [Online]. Available: http://www.elsevier.com/locate/compgeo.

[12] H. K. Dang and M. A. Meguid, "Coupled fem-dem approach for modelling soil structure interaction problems involving large deformation.," 2013.

[13] V. Šmilauer, E. Catalano, S. D. Bruno Chareyre, J. Duriez, and N. Dyck, "Yade documentation 2nd edition," in DOI: 10.5281/zenodo.34043. [Online]. Available: http://yade-dem.org/doc/.

[14] D. Weatherley, W. Hancock, and V. Boros, "Esys-particle tutorial and user's guide version 2.3.1," in.

[15] *Finite Element procedure*. Prentice-Hall, 1996.

[16] P. Cundall and O. Strack, "The development of constitutive laws for soil using the distinct element method," *In Third International Conference on Numerical Methods in Geomechanics*, 1979.

[17] A.Drescher and G. D. J. Jong, "Photoelastic verification of a mechanical model for the flow of a granular material," *Elsevier,Journal of the Mechanics and Physics of Solids, University of Technology, Delft, The Netherlands, Printed in GreatBritain.*, 1972. [Online]. Available: http://www.elsevier.com/locate/compgeo.

[18] D. Barreto, M. A. Wadee, K. J. Hanley, and O'Sullivan, "Use of dem and elastic stability analysis to explain the influence of the intermediate principal stress on shear strength," *Géotechnique ,Imperial College London,Department of Civil and Environmental Engineering UK*, 2012.

[19] X. LI and H.-S. YU, "Numerical investigation of granular material behaviour under rotational shear," *Géotechnique ,Nottingham Centre for Geomechanics, University of Nottingham, UK.*, 2010.

[20] H. Ouadfel and L. Rothenburg, "Stress force fabric' relationship for assemblies of ellipsoids," *Elsevier, Mechanics of Materials 33 2001*, 2001. [Online]. Available: `http://www.elsevier.com/locate/mechmat`.

[21] Thornton, "Numerical simulations of the direct shear test," *Chem. Eng. Technol. 26 Engineering and Applied Science Aston University, Birmingham*, 2003.

[22] Park, "Rock joints under direct shear using bonded-particles," 2009.

[23] Ng, Cui, and Belheine, "Triaxial tests of granular soil samples," 2004.

[24] Villard and Chareyre, "Theoretical versus experimental modeling of the anchorage capacity of geotextile in trenches," *Geosynthetics International, Vol. 9, No. 2, pp. 97-123*, 2004.

[25] McDowell, "Discrete element modelling of geogrid-reinforced aggregates," *York Universit Proceedings of the Institution of Civil Engineers Geotechnical Engineering 159 Issue GE1*, 2006.

[26] C. Chen, "Discrete element modelling of geogrid -reinforced railway balast and track transition zones," Ph.D. dissertation, The University of Nottingham, 2013. [Online]. Available: `http://eprints.nottingham.ac.uk/13399/`.

[27] J. Han, A. Bhandari, and F. Wang, "Dem analysis of stresses and deformations of geogrid-reinforced embankments over piles," *American Society of Civil Engineers*, 2012. DOI: `DOI:10.1061/(ASCE)GM.1943-5622.0000050.`. [Online]. Available: `http://www.ascelibrary.org`.

[28] H. Faheem, F. Cai, and K. Ugai, "Three-dimensional base stability of rectangular excavations in soft soils using fem," *Elsevier,Computers and Geotechnics 31 (2004) 67–74*, 2004. [Online]. Available: http://www.elsevier.com/locate/compgeo.

[29] R. J.Finno, J. Blackburn, and J. F.Roboski, "Three-dimensional effects for supported excavations in clay," *JOURNAL OF GEOTECHNICAL AND GEOENVIRONMENTAL ENGINEERING,ASCE,Vol. 133, No. 1*, 2007. DOI: 10.1061/ASCE1090-02412007133:130. [Online]. Available: http://www.ascelibrary.org.

[30] Y. A. Khodair and S. Hassiotis, "Analysis of soil–pile interaction in integral abutment," *Elsevier,Computers and Geotechnics 32 (2005) 201–209*, 2005. DOI: doi:10.1016/j.compgeo.2005.01.005. [Online]. Available: http://www.elsevier.com/locate/compgeo.

[31] B. K. Maheshwari, K. Z. Truman, M. Hesham, E. Naggar, and P. L. Gould, "Three-dimensional finite element nonlinear dynamic analysis of pile groups for lateral transient and seismic excitations," *Geotechnical Research Centre, Faculty of Engineering Science, The University of Western Ontario, London, ON N6A 5B9, Canada*, 2004. DOI: 10.1139/T03-073.

[32] G.Beer, "An isoparametric joint/interface element for finite element analysis," *INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING, VOL. 21, 585–600 (1985)*, 1985.

[33] H. Langen and P.A.Vermeer, "Interface elements for singular plasticity points," *INTERNATIONALJOURNALFOR NUMERICALAND ANALYTICAL METHODS IN GEOMECHANICS.VOL.15,301-315(1991)*, 1991.

[34] D. Karabatakis and T. Hatzigogos*, "Analysis of creep behaviour using interface elements," *Elsevier,Computers andGeotechnics 29 (2002) 257–277*, 2002. DOI: S0266-352X(01)00033-7. [Online]. Available: http://www.elsevier.com/locate/compgeo.

[35] B. V.G., "Earth pressure on the cylindrical retaining walls," in *Conference on Earth Pressure Problems*.

[36] T. Tobar and M. A. Meguid, "Experimental study of the earth pressure distribution on cylindrical shafts," Journal of Geotechnical and Geoenvironmental Engineering ,ASCE, Tech. Rep., 2011. DOI: DOI:10.1061/(ASCE)GT.1943-5606.0000535. [Online]. Available: http://www.scelibrary.org.

[37] S. W. . Agaiby and C. J. Jones, "Design of reinforced fill system over void," *Can. Geotech. J. 32: 939-945 (1995,*, 1995. [Online]. Available: http://www.nrcresearchpress.com.

[38] E. M. Palmeira, "Soil–geosynthetic interaction: Modelling and analysis," *Elsevier,Geotextiles and Geomembranes*, 2009. DOI: doi:10.1016/j.geotexmem.2009.03.003. [Online]. Available: http://www.elsevier.com/locate/geotexmem.

[39] R. L. Michalowski, "Limit loads on reinforced foundation soils," *ASCE ,Journal of Geotechnical and Geoenvironmental Engineering, Vol. 130, No. 4,*, 2004. DOI: 10.1061/ASCE!1090-02412004!130:4381. [Online]. Available: http://www.elsevier.com/locate/geotexmem.

[40] H. K. Dang and M. A. Meguid, "An efficient finite-discrete element method for qusi static nonlinear soil structure interaction problems.," *International Journal for Numerical and Analytical Methods in Geomechanics*, 2011. [Online]. Available: http://www.ascelibrary.org.

[41] K. Han and D. Peri, "A combined finite/discrete element simulation of shot peening processes," *Engineering Computations,Vol. 17 No. 5, 2000, pp. 593-619,MCB University Press*, 2000.

[42] S. Xiao and T. B. b, "A bridging domain method for coupling continua with molecular dynamics," *Elsevier,Comput. Methods Appl. Mech. Engrg. 193 (2004) 1645–1669*, 2004. DOI: 10.1016/j.cma.2003.12.053. [Online]. Available: http://www.elsevier.com/locate/cma.

[43] H. K. Dang and M. A. Meguid, "Algorithm to generate a discrete element specimen with predefined properties," COLUMBIA UNIVERSITY, Tech. Rep., 2013.

[44] P. Cundall and R. Hart, "Numerical modelling of discontinua," *Engineering Computations*, 1992.

[45] R. Codina, "Stabilization of incompressibility and convection through orthogonal sub-scales in finite element methods," *Comput. Meth. Appl. Mech. Eng., 190:1579 1599*, 2000.

[46] R. Codina and J. Blasco, "Stabilized finite element method for transient navierstokes equations based on pressure gradient projection," *Comput. Meth. Appl. Mech. Eng., 182:287-300*, 2000.

[47] E. O nate and J. Rojek, "Combination of discrete element and finite element methods for dynamic analysis of geomechanics problems," *Computer Methods in Applied Mechanics and Engineering, 193(27-29):30873128*, 2004.

[48] N. GUO, "Multiscale characterization of the shear behavior of granular media," Ph.D. dissertation, The Hong Kong University of Science and Technology, 2014.

[49] V. D. H. Tran, "A coupled finite-discrete element framework for soil-structure interaction analysis," Ph.D. dissertation, McGill University Montréal, 2013.

[50] C. A. Labra, E. Onate, and J. Rojek, "Advances in the development of the discrete element method for excavation processes," *International Center for Numerical Methods in Engineering*, 2012.

[51] M. Jiang, J. Konrad, and S. Leroueil, "An efficient technique for generating homogeneous specimens for dem studies," *Computers and Geotechnics*, 2003. [Online]. Available: `www.elsevier.com/locate/compgeo`.

[52] M. Jiang, J. Liu, and B. X, "Exploring the critical state properties and major principal stress rotation of sand in direct shear test using the distinct element method," *Granular Matter , Springer-Verlag GmbH Germany*, 2018. [Online]. Available: `https://doi.org/10.1007/s10035-018-0796-z`.