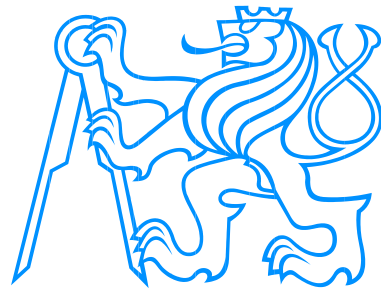


**CZECH TECHNICAL UNIVERSITY  
IN PRAGUE**

**Faculty of Civil Engineering  
Department of Mechanics**



**Object-Oriented Design and Implementation of  
Contact Mechanics into Finite Element Code  
"OOFEM"**

**Objektově orientovaný návrh a implementace  
algoritmů kontaktní mechaniky do konečněprvkového  
programu "OOFEM"**

**DIPLOMA THESIS**

Bc. Ondřej Faltus

Study programme: Stavební inženýrství

Discipline: Konstrukce a dopravní stavby

Supervisor: Ing. Martin Horák, Ph.D.

Prague, 2020



## ZADÁNÍ DIPLOMOVÉ PRÁCE

### I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: Faltus Jméno: Ondřej Osobní číslo: 439041  
Zadávací katedra: K132  
Studijní program: Stavební inženýrství  
Studijní obor: Konstrukce a dopravní stavby

### II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce: Objektově orientovaný návrh a implementace algoritmů kontaktní mechaniky do konečně prvkového programu OOFEM

Název diplomové práce anglicky: Object-oriented design and implementation of contact mechanics into finite element code OOFEM

Pokyny pro vypracování:

Cílem práce je implementace základních algoritmů pro řešení kontaktních úloh metodou konečných prvků. Algoritmy budou implementovány do programového prostředí OOFEM. Součástí implementace bude návrh vhodného rozšíření stávající struktury programu OOFEM využívající jeho objektově orientovanou strukturu. Implementované algoritmy budou ověřeny pomocí vhodně zvolených příkladů.

Seznam doporučené literatury:

- [1] Wriggers, Peter, *Computational contact mechanics*, 2006, Springer
- [2] Konyukhov, Alexander and Izi, Ridvan, *Introduction to computational contact mechanics: a geometrical approach*, 2015, John Wiley & Sons
- [3] Yastrebov, Vladislav A, *Numerical methods in contact mechanics*, 2013, John Wiley & Sons

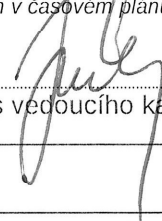
Jméno vedoucího diplomové práce: Ing. Martin Horák, Ph.D.

Datum zadání diplomové práce: 26.9.2019 Termín odevzdání diplomové práce: 5.1.2020

Údaj uveďte v souladu s datem v časovém plánu příslušného ak. roku



Podpis vedoucího práce



Podpis vedoucího katedry

### III. PŘEVZETÍ ZADÁNÍ

*Beru na vědomí, že jsem povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je nutné uvést v diplomové práci a při citování postupovat v souladu s metodickou příručkou ČVUT „Jak psát vysokoškolské závěrečné práce“ a metodickým pokynem ČVUT „O dodržování etických principů při přípravě vysokoškolských závěrečných prací“.*

27. 9. 2019

Datum převzetí zadání



Podpis studenta(ky)



---

## Declaration

I hereby declare that this diploma thesis on the topic of *Object-Oriented Design and Implementation of Contact Mechanics into Finite Element Code "OOFEM"*, conducted under the supervision of Ing. Martin Horák, PhD., is the authentic result of my own work and ideas, except when other sources have been cited and referenced.

In Prague January 5<sup>th</sup> 2020

Ondřej Faltus



---

## Acknowledgement

Firstly, Ing. Martin Horák, PhD. is due many thanks for the supervision of this thesis and his utmost dedication to the task, be it help with the theoretical basis of the problem or with code testing and debugging.

I would also like to extend many thanks toward all the people in my surroundings, who have provided much support and demonstrated remarkable understanding on many levels during the time the thesis was being written.





---

## Abstract

The topic of the thesis are algorithms of computational contact mechanics. Underlying theoretical laws are reviewed and selected existing algorithms are adapted for integration into the "OOFEM" finite element software. The implementation is discussed in detail and extensively tested on several examples of benchmark problems with varying degrees of complexity.

## Abstrakt

Tématem práce jsou numerické algoritmy kontaktní mechaniky. Jsou představeny základní teoretické zákonitosti a vybrané algoritmy jsou adaptovány pro zapracování do konečněprvkového softwaru "OOFEM". Provedená implementace je podrobně rozebrána a rozsáhle otestována na několika příkladech testovacích problémů o různém stupni složitosti.

## Keywords

contact mechanics, OOFEM, contact discretization, node-to-node contact, node-to-segment contact, penalty method, Lagrangian multiplier method, rigid flat punch problem, Hertz problem

## Klíčová slova

kontaktní mechanika, OOFEM, diskretizace kontaktu, kontakt uzel-uzel, kontakt uzel-segment, metoda penalty, metoda Lagrangeova multiplikátoru, problém pevného plochého úderu, Hertzův problém



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Computational Contact Mechanics . . . . .	1
<b>2</b>	<b>Theory</b>	<b>3</b>
2.1	The Finite Element Method . . . . .	3
2.2	The Contact Constraints . . . . .	6
2.2.1	The Non-Penetration Condition . . . . .	6
2.2.2	Lagrangian Multiplier Method . . . . .	8
2.2.3	Penalty Method . . . . .	9
2.2.4	Other methods . . . . .	10
2.3	Contact Constraints in FEM . . . . .	10
2.3.1	Contact Discretization . . . . .	11
2.3.2	Node-to-Node . . . . .	12
2.3.3	Node-to-Segment . . . . .	13
2.3.4	Penalty Method in FEM . . . . .	14
2.3.5	Lagrangian Multiplier Method in FEM . . . . .	15
<b>3</b>	<b>Implementation</b>	<b>18</b>
3.1	The Specifics of OOFEM . . . . .	18
3.1.1	Inner Structure . . . . .	18
3.1.2	Note on OOFEM Coding Conventions . . . . .	19
3.2	Implementation of Node-to-Node Contact . . . . .	19
3.2.1	Use of the ActiveBoundaryCondition class . . . . .	19
3.2.2	Node-to-Node Penalty Contact . . . . .	20
3.2.3	Node-to-Node Lagrange Multiplier Contact . . . . .	23
3.3	Implementation of Node-to-Segment Contact . . . . .	27
3.3.1	Class Structure . . . . .	27
3.3.2	Boundary Conditions . . . . .	29
3.3.3	Element Edge Contact Segment . . . . .	32
3.3.4	Analytical Function Contact Segment . . . . .	36
3.4	Avenues of Further Development . . . . .	41
<b>4</b>	<b>Numerical Experiments</b>	<b>43</b>
4.1	Development Testing . . . . .	43
4.2	Contact of Two Bars . . . . .	48
4.3	Two Bars with Penalty Condition . . . . .	52
4.4	Penalty Size Study . . . . .	55

## Contents

---

4.5 Rigid Flat Punch Problem . . . . .	58
4.6 Hertz Contact Problem . . . . .	71
<b>5 Conclusion</b>	<b>78</b>
<b>List of Illustrations</b>	<b>79</b>
<b>List of Tables</b>	<b>81</b>
<b>List of Listings</b>	<b>82</b>
<b>References</b>	<b>83</b>

# 1 Introduction

The aim of this thesis is to introduce algorithms of contact mechanics into the finite element software OOFEM. OOFEM is a free, open-source, strictly object oriented and cross-platform finite element code developed chiefly at Czech Technical University in Prague, Faculty of Civil Engineering, Department of Mechanics [Patzák, 2000]. As OOFEM is a C++ code, the main focus of the thesis is formulation of the principles of computational contact mechanics as C++ algorithms and the subsequent introduction of these into the existing OOFEM architecture.

## 1.1 Motivation

As of now, OOFEM is equipped to handle various problems from the realms of mechanical, transport and fluid mechanics [Patzák, 2000].

The structural mechanics module (known in the internal organization of code libraries as the `sm` module) is the most advanced of those, with extensive material and element libraries as well as several analysis modes. However, a working framework of contact mechanics is, as of now, missing. Finite element contact mechanics is a growing field with a large and varied range of possible practical applications in civil engineering, mechanical engineering and more. Implementation of contact algorithms into OOFEM is therefore a sound and reasonable move which will both extend the vast and complex code and open a way into its utilization for a new spectrum of computational analyses.

## 1.2 Computational Contact Mechanics

Computational contact mechanics is a specialized field dealing mainly with numerical simulations of physical contact of solid bodies. The origins of contact mechanics lie at the end of the 19<sup>th</sup> century, when Heinrich Hertz first published his work [Hertz, 1881] [Yastrebov, 2013]. Hertz's example involved two elliptical elastic bodies coming into contact under the condition of no friction. The so-called Hertz contact problem has an analytical solution and is an important benchmark problem until this day. As such it is also discussed in section 4.6 of this thesis [Konyukhov and Izi, 2015].

Even in the original first Hertz formulation, contact mechanics already found much practical use in mechanical engineering, namely in contact problems involving cylindrical bodies, such as rail-wheel interactions or design of bearings [Konyukhov and Izi, 2015] [Yastrebov, 2013].

Further development of the field came throughout the 20<sup>th</sup> century, however progress was hampered by the fact that only a few special cases of solid body contact allow for direct analytical solutions, which makes it largely unsuitable for large applications on the industrial scale [Wriggers, 2006] [Yastrebov, 2013].

With the dawn of the finite element method in the 1960s, contact mechanics had come once again to the forefront of scientific interest. Throughout the 1960s and 1970s, large number of methods incorporating contact mechanics into FEM was developed, driven by high industrial demand [Wriggers, 2006] [Yastrebov, 2013].

In recent years, with the rapid evolution of computational power in hardware, many previous restrictions on FEM task complexity are quickly disappearing, creating demand for more and more complicated methods to solve increasingly complex contact problems [Yastrebov, 2013]. Modern applications of computational contact methods include a wide range of problems from drilling, through metal forming processes to as complex tasks as car crash test simulations [Wriggers, 2006].

## 2 Theory

In this section, the problem of contact in finite element computations shall be discussed on the theoretical level.

It is important to note that contact mechanics as a field include a large number of distinct problems needing specific algorithms to be numerically solved. This theoretical review, same as the implementation in OOFEM subject of this thesis, focuses exclusively on the case of frictionless contact, i.e. on enforcing the non-penetration condition in the context of the finite element method. Furthermore, wherever necessary, two-dimensional space shall be assumed as the basic case, although most of the theoretical basis is domain-independent.

### 2.1 The Finite Element Method

Firstly, the basics of the finite element method (FEM) shall be reviewed to present a theoretical foundation for later discussion of the specific contact applications. Specifically of interest is the finite element formulation of structural mechanics.

For the purpose of this theoretical explanation, the engineering notation (also known as the Voigt notation) of the *stress tensor* and the *strain tensor* is used. Both tensors are stored in one-dimensional arrays and henceforth referred to as the *stress vector* and *strain vector*. This is a useful for both the FEM formulation as well as the practical implementation in computer code where such notation saves valuable space in memory [Patzák, 2019].

To use a two-dimensional domain as an example, the stress and strain vectors are reduced to three items each:

$$\boldsymbol{\sigma} = \begin{pmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{12} \end{pmatrix} \quad \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ 2\varepsilon_{12} \end{pmatrix} \quad (1)$$

where

$\boldsymbol{\sigma}$  is the stress vector

$\boldsymbol{\varepsilon}$  is the strain vector

$\sigma_{ij}$  and  $\varepsilon_{ij}$  are the relevant components of the stress tensor and the strain tensor, respectively

Furthermore, it is necessary to also introduce a way of applying the symmetric gradient to these vectors. For this, the matrix operator  $\boldsymbol{\mathcal{D}}$  shall be used. It assumes different forms for each domain dimension. The example for the two-dimensional case reads [Patzák, 2019]:

$$\boldsymbol{\partial} = \begin{bmatrix} \frac{\partial}{\partial x_1} & 0 \\ 0 & \frac{\partial}{\partial x_2} \\ \frac{\partial}{\partial x_2} & \frac{\partial}{\partial x_1} \end{bmatrix} \quad (2)$$

The problem of structural mechanics in a continuum space  $\Omega$  can be interpreted as the *equilibrium equations* in the following form [Konyukhov and Izi, 2015] [Zienkiewicz and Taylor, 2000]:

$$\boldsymbol{\partial}^T \boldsymbol{\sigma}(\mathbf{x}) = -\mathbf{f}(\mathbf{x}) \quad \forall \mathbf{x} \in \Omega \quad (3)$$

where

$\mathbf{x}$  is a geometrical point in the continuum

$\boldsymbol{\sigma}(\mathbf{x})$  is the stress vector in location  $\mathbf{x}$

$\mathbf{f}$  is the *vector of external forces*

This differential problem is constrained usually by two types of boundary conditions. Those are the Dirichlet boundary conditions, prescribing in this case a displacement value, acting on the *Dirichlet boundary*  $\Gamma_u$ , expressed as

$$\mathbf{u}(\mathbf{x}) = \bar{\mathbf{u}}(\mathbf{x}) \quad \forall \mathbf{x} \in \Gamma_u \quad (4)$$

where

$\mathbf{u}$  is the *displacement vector*

$\bar{\mathbf{u}}$  is the *vector of prescribed displacements*

and the Neumann boundary conditions, acting on the *Neumann boundary*  $\Gamma_\sigma$ , in the form

$$\mathbf{n}_\Gamma^T(\mathbf{x}) \boldsymbol{\sigma}(\mathbf{x}) = \bar{\mathbf{t}}(\mathbf{x}) \quad \forall \mathbf{x} \in \Gamma_\sigma \quad (5)$$

where

$\mathbf{n}_\Gamma = \begin{bmatrix} n_1 & 0 & n_2 \\ 0 & n_2 & n_1 \end{bmatrix}$  is a matrix assembling components of the unit normal  $n_i$  to the boundary.

$\bar{\mathbf{t}}$  is the *vector of prescribed boundary forces*

It is important to note that the Neumann and Dirichlet boundary conditions are mutually exclusive, in other terms  $\Gamma_u \cap \Gamma_\sigma = \emptyset$ , and at the same time can no part of the boundary have neither of them,  $\Gamma_u \cup \Gamma_\sigma = \Gamma$  [Zienkiewicz and Taylor, 2000].

No less important are the *constitutive equations* describing the influence of materials. They form the relationship between stress and strain, which is dictated by the material of the solid body and therefore differs for each material [Zienkiewicz and Taylor, 2000]. For



the linear elastic material as an example, the constitutive equations take the form known as the *linear Hook law* [Patzák, 2019] [Konyukhov and Izi, 2015]

$$\boldsymbol{\sigma}(\mathbf{x}) = \mathbf{D}_e \boldsymbol{\varepsilon}(\mathbf{x}) \quad \forall \mathbf{x} \in \Omega \quad (6)$$

where

$\mathbf{D}_e$  is the *elastic stiffness tensor* (in the Voigt notation as a  $6 \times 6$  matrix)

Finally, the *geometric equations* tie together strain and the displacement vector in the form [Patzák, 2019]

$$\boldsymbol{\varepsilon}(\mathbf{x}) = \boldsymbol{\partial} \mathbf{u}(\mathbf{x}) \quad \forall \mathbf{x} \in \Omega \quad (7)$$

All the mentioned equations put together present the *strong form* of a boundary value problem [Konyukhov and Izi, 2015]. The essence of the finite element method lies in the idea of substituting a so-called *weak form* for this strong form, which allows for linearization and computerized solution of the problem [Zienkiewicz and Taylor, 2000].

The weak form for the equilibrium equation (3) reads as follows [Konyukhov and Izi, 2015]:

$$\int_{\Omega} (\boldsymbol{\partial}^T \boldsymbol{\sigma})^T \delta \mathbf{u} d\Omega + \int_{\Omega} \mathbf{f}^T \delta \mathbf{u} d\Omega = \mathbf{0} \quad (8)$$

where

$\delta \mathbf{u}$  are the *test functions* (also known in this context as *virtual displacements* or *variational displacements*)

What this means is that the requirement for fullfilling the equilibrium equations in all points has regressed to a requirement for an average fullfillment across the domain. The averaging is provided by the test functions, as long as those are both smooth and compliant with the Dirichlet boundary conditions on boundary  $\Gamma_u$  [Konyukhov and Izi, 2015] [Patzák, 2019].

After applying certain rather extensive transformations to equation (8) (for details see [Konyukhov and Izi, 2015]), it can take the form

$$\int_{\Omega} \delta \boldsymbol{\varepsilon}^T \boldsymbol{\sigma} d\Omega = \int_{\Gamma_{\sigma}} \delta \mathbf{u}^T \bar{\mathbf{t}} d\Gamma_{\sigma} + \int_{\Omega} \delta \mathbf{u}^T \mathbf{f} d\Omega \quad (9)$$

where

$\delta \boldsymbol{\varepsilon}$  are the *virtual deformations* obtained from the virtual displacements as  $\delta \boldsymbol{\varepsilon} = \boldsymbol{\partial} \delta \mathbf{u}$

which is in fact physically equivalent to the equilibrium of virtual energy [Patzák, 2019]

$$\delta W_{int} = \delta W_{ext} \quad (10)$$

where

$\delta W_{int}$  is the *virtual internal energy*

$\delta W_{ext}$  is the *virtual external energy*

After discretizing the domain  $\Omega$  into  $n$  elements connected by nodes, all variables and virtual variables can be approximated as linear combinations of nodal displacement values  $\mathbf{d}$  or nodal virtual displacement values  $\delta \mathbf{d}$ , respectively. The combination coefficients for this are provided in the form of local element weight functions arranged in matrices  $\mathbf{N}_e(\mathbf{x})$  and their derivatives arranged in matrices  $\mathbf{B}_e(\mathbf{x})$  [Patzák, 2019]. Due to this, the integrals in equation (9) morph into sums over finite elements:

$$\delta \mathbf{d} \left( \left( \sum_1^n \int_{\Omega_e} \mathbf{B}_e^T \mathbf{D}_e \mathbf{B}_e d\Omega \right) \mathbf{r} - \left( \sum_1^n \int_{\Omega_e} \mathbf{N}_e^T \mathbf{f} d\Omega \right) - \left( \sum_1^n \int_{\Gamma_{\sigma,e}} \mathbf{N}_e^T \bar{\mathbf{t}} d\Gamma_\sigma \right) \right) = 0 \quad (11)$$

where

$\Omega_e$  is the part of the domain belonging to a single finite element

$\Gamma_{\sigma,e}$  is the part of the Neumann boundary belonging to a single finite element

Finally, as equation (11) is satisfied for any field of test functions if and only if the terms in brackets are zero, the virtual nodal displacements can be discarded from the equation and the terms in brackets defined as the matrices  $\mathbf{K}$ ,  $\mathbf{f}_\Omega$  and  $\mathbf{f}_\Gamma$  [Patzák, 2019], obtaining a linear system of equations

$$\mathbf{K} \mathbf{d} = \mathbf{f}_\Omega + \mathbf{f}_\Gamma \quad (12)$$

where

$\mathbf{K}$  is the *global stiffness matrix*

$\mathbf{d}$  is the *global vector of nodal displacements*

$\mathbf{f}_\Omega$  and  $\mathbf{f}_\Gamma$  are the *global force vectors*

All this allows a FEM solver to solve the equation system (12), either directly or iteratively, and obtain the nodal displacement values. From them, all other variables can be obtained by the means of interpolation functions  $\mathbf{N}_e$  and their derivatives  $\mathbf{B}_e$ .

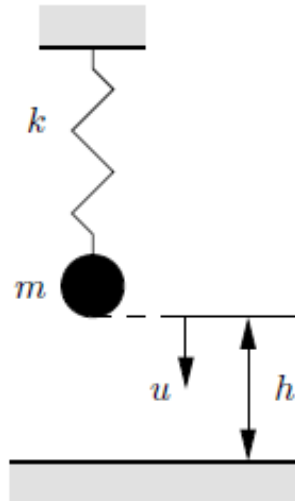
How do contact mechanics integrate into this scheme shall be the topic of the following pages.

## 2.2 The Contact Constraints

### 2.2.1 The Non-Penetration Condition

The core idea of contact of solid bodies is the *non-penetration condition*. In layman's terms, it is the principle that the bodies can never occupy both the same space. In figure

1, a spring mass system constrained by a rigid plane is pictured, discussed in [Konyukhov and Izi, 2015] and [Wriggers, 2006]. The spring of a stiffness  $k$  and undeformed length  $l$  is burdened by a weight of mass  $m$ . In the height  $h$  from the equilibrium and  $H$  from the spring origin point (therefore  $H = h + l$ ) lies the boundary which cannot be penetrated.



**Figure 1:** A spring-mass system in contact, taken from [Wriggers, 2006]

Were it not for the contact condition, the energy of the system would be given as [Wriggers, 2006]

$$W(u) = \frac{1}{2}ku^2 - mgu \quad (13)$$

where

$W$  is the energy

$u$  is the current displacement of the point of mass as seen in figure 1

$g$  is gravity

and therefore the equilibrium state of the system occurs when [Konyukhov and Izi, 2015]

$$W(u) = \frac{1}{2}ku^2 - mgu \rightarrow \min \quad (14)$$

However, there is the rigid plane present. Expressed by the means of a penetration function  $c$ ,<sup>1</sup> a non-penetration condition takes the form of the following inequality [Konyukhov and Izi, 2015]:

<sup>1</sup>Literature differs in regards to a symbol for this function, e.g. [Konyukhov and Izi, 2015] uses the letter  $p$ . Given that  $p$  is later used in this text to denote the penalty parameter,  $c$  (known e.g. from [Wriggers, 2006]) was chosen as a better alternative.

$$c(u) = l + u - H \leq 0 \tag{15}$$

This adds a restriction to the minimization task denoted in equation (14). There are two possibilities for the state of the system at any given time: contact or no contact. This is reflected in the *Karush–Kuhn–Tucker conditions* [Konyukhov and Izi, 2015]:

$$c = 0 \wedge N > 0 \tag{16}$$

$$c < 0 \wedge N = 0 \tag{17}$$

$$cN = 0 \tag{18}$$

where

$N$  is the *normal contact force*, also taking on the role of a Lagrangian multiplier as is going to be seen further

The first condition describes the situation of contact, the second the situation of no contact. The third equation stresses that the situations are mutually exclusive. Now, the methods to enforce these constraints during the minimization task (14) shall be discussed.

### 2.2.2 Lagrangian Multiplier Method

This method rests in the idea of reformulating problem (14) together with conditions (16) - (18) into a single Lagrangian functional in the form [Konyukhov and Izi, 2015]:

$$L(u, \lambda) = W + \lambda c \rightarrow \min \tag{19}$$

where

$L$  is the *Lagrangian functional*

$\lambda$  is the *Lagrangian multiplier*

The way to minimize the new constrained problem (19) leads through separately differentiating it by both  $u$  and  $\lambda$  [Konyukhov and Izi, 2015] [Wriggers, 2006]. After some transformations, the task takes on the form of a system of linear equations:

$$ku + \lambda = -mg \tag{20}$$

$$u = H - l \tag{21}$$

The Lagrangian multiplier therefore indeed assumes the form of the physical contact force (consider the physical dimension of equation (20)). The most significant advantage of this method is the fact that the non-penetration condition is fulfilled exactly. There

are serious disadvantages as well though. For each contact condition, the task equation system must contain one more variable [Konyukhov and Izi, 2015]. This is insignificant in the case of this spring-mass example, however plays a large role in the large tasks typical for FEM computations. Moreover, the very existence of this variable is not constant, it only exists when contact occurs, putting additional strain on the software procedures.

### 2.2.3 Penalty Method

It is possible to avoid the additional variable introduced by the Lagrangian multiplier method. The procedure is called the *penalty method* and its cost is precision of the solution [Yastrebov, 2013] [Konyukhov and Izi, 2015].

This time, the functional (14) is transformed in the following way:

$$W_p(u) = W + \frac{1}{2}pc^2(u) \rightarrow \min \quad (22)$$

where

$\Pi_p$  is the *penalty-enhanced functional*

$p$  is the *penalty parameter*

The idea behind this is in fact rather similar to the idea of the Lagrangian multiplier. The additional contact force is however not introduced as an unknown variable here, but rather approximated as a certain large multiple of the penetration function. The penalty parameter serves as the multiplier.

The new functional only depends on one variable  $u$  and to minimize it, one differentiation is sufficient [Konyukhov and Izi, 2015]. The resulting equation reads

$$ku - mg + p(l + u - H) = 0 \quad (23)$$

Upon closer examination of this condition, it can be seen that the contact force is now expressed as  $p(l - u - H)$ , or  $pc$ . Physically, this is as if an additional spring was present in the system, acting against penetration by the stiffness  $p$  [Konyukhov and Izi, 2015]. A perfect solution would be for this spring to be rigid, i.e.  $p \rightarrow \infty$ .

This thought directly leads to the main disadvantage of the penalty method: it allows for (small) penetration. An infinite penalty parameter is impossible in numerical modelling. Very large penalty parameters should yield very small penetrations, which could be acceptable for a result of numerical computation. However, very large penalty parameters are prone to disturb the equation systems and make them difficult to solve for most FEM equation solvers [Wriggers, 2006] [Konyukhov and Izi, 2015].

### 2.2.4 Other methods

The penalty method and the Lagrangian multiplier method are the only two methods implemented into OOFEM as a part of this thesis. Nevertheless, it is appropriate to at least mention other existing methods. Most prominent of those is probably the *augmented Lagrangian multiplier method*, which combines the advantages of the two described methods [Yastrebov, 2013] [Konyukhov and Izi, 2015].

Among others, the *Nitsche method* is used for contact of two deformable bodies, altering the functional of the penalty method (see equation (22)) by adding a condition of equality of contact forces between the bodies or the *mortar method* used in the case of segment-to-segment contact discretization [Konyukhov and Izi, 2015].

## 2.3 Contact Constraints in FEM

In a finite element task, usually only one solid body is present. For contact tasks however, the case is different, as often multiple bodies are present within the same domain. Without contact, the equation system for a domain consisting of two separate bodies A and B looks like

$$\begin{bmatrix} \mathbf{K}_A & \mathbf{0} \\ \mathbf{0} & \mathbf{K}_B \end{bmatrix} \begin{pmatrix} \Delta \mathbf{d}_A \\ \Delta \mathbf{d}_B \end{pmatrix} = \begin{pmatrix} \mathbf{f}_A \\ \mathbf{f}_B \end{pmatrix} \quad (24)$$

where

$\mathbf{K}_A$  is the stiffness matrix of body A

$\mathbf{K}_B$  is the stiffness matrix of body B

$\Delta \mathbf{d}_A$  is the vector of differential displacements of body A (used here because contact inherently being non-linear [Wriggers, 2006] and therefore the task solution being iterative)

$\Delta \mathbf{d}_B$  is the vector of differential displacements of body B

$\mathbf{f}_A$  is the force vector for body A

$\mathbf{f}_B$  is the force vector for body B

In the moment when the bodies come into contact, the situation changes though and the two equation systems become entangled:

$$\begin{bmatrix} \mathbf{K}_A + \mathbf{K}_{AA}^c & \mathbf{K}_{AB}^c \\ \mathbf{K}_{BA}^c & \mathbf{K}_B + \mathbf{K}_{BB}^c \end{bmatrix} \begin{pmatrix} \Delta \mathbf{d}_A \\ \Delta \mathbf{d}_B \end{pmatrix} = \begin{pmatrix} \mathbf{f}_A + \mathbf{f}_A^c \\ \mathbf{f}_B + \mathbf{f}_B^c \end{pmatrix} \quad (25)$$

where

$\mathbf{K}_{AA}^c$ ,  $\mathbf{K}_{AB}^c$ ,  $\mathbf{K}_{BA}^c$  and  $\mathbf{K}_{BB}^c$  are the contributions to the stiffness matrix created by the contact constraints

$\mathbf{f}_A^c$  and  $\mathbf{f}_B^c$  are the contributions to the force vector created by the contact constraints

FEM software operates typically by assembling and localizing contributions to global matrices and vectors from all domain components [Zienkiewicz and Taylor, 2000] [Patzák, 2019]. A question arises: how to achieve the assembly of the contact contributions from equation (25) into the global stiffness matrix and force vector?

A typical approach makes use of the concept of *contact elements* [Yastrebov, 2013]. Virtual elements are created in the domain, holding information about the contacting surfaces, providing detection of contact, applying the contact constraints and passing the resulting stiffness matrices and force vectors to the rest of the code [Yastrebov, 2013]. In OOFEM, the chosen approach is different. The contact constraints are created as a sub-type of active boundary conditions, which are able to assemble their own contributions to stiffness and forces [Patzák, 2000]. Detailed discussion of this feature is provided in section 3.2.1.

### 2.3.1 Contact Discretization

Contact resolution in FEM consists of three tasks [Yastrebov, 2013]:

- contact detection
- assembly of contact constraints
- resolution of the constructed problem

The first two of those steps are heavily dependent on the *contact discretization* [Yastrebov, 2013], i.e. the way contact is defined as an interaction between the components of a FEM domain. The most prominent contact discretizations include [Konyukhov and Izi, 2015]:

- **node-to-node** (NTN): penetration is considered between a pair of nodes only, one on each contacting body, one of them usually designated a *master node* and the other a *slave node*
- **node-to-segment** (NTS): a node contacts with a segment, e.g. an edge of an element, and is constrained from penetrating it at any point
- **segment-to-analytical-segment** (STAS): a segment is constrained from penetrating a given boundary, usually defined as an analytical function and independent from the deformable body (bodies) described by FEM
- **segment-to-segment** (STS): two active, element segments in contact with each other

Only the first two discretizations of these are implemented in OOFEM within the scope of this thesis. In addition a possibility for a node to interact with an analytical segment is implemented as a sub-feature of node-to-segment contact.

The following sections describe the differences between the node-to-node and node-to-segment approaches. Apart from these differences in the geometry of contact detection and localization of contact forces, the rest of the constraint formulation is independent of the discretization, as explained further.

### 2.3.2 Node-to-Node

The task of contact detection lies in finding the *projection vector*<sup>2</sup> and the *normal gap* [Konyukhov and Izi, 2015]. In the case of two nodes this is most simple, as the vector is just

$$\mathbf{n} = \mathbf{x}_m - \mathbf{x}_s \quad (26)$$

where

$\mathbf{n}$  is the projection vector

$\mathbf{x}_m$  is the coordinate vector of the master node

$\mathbf{x}_s$  is the coordinate vector of the slave node

and the normal gap is its size. The gap is defined as negative if the nodes are penetrating one another and positive in the case there is no contact between them.

It shall be noted however, that the node-to-node discretization is considered only suitable for small deformations [Yastrebov, 2013] and in OOFEM is limited to that case. This means that while the gap is constructed from the current node coordinates, the projection vector is based on the initial coordinates and therefore remains the same for the whole duration of the computation.

The contact forces and contact stiffness, which the contact constraints, regardless of their formulation, assemble to the global force vector and stiffness matrix, respectively, have to be allocated to both the master and slave node, in opposite directions. To ensure this serves the *extended N-matrix*<sup>3</sup> [Konyukhov and Izi, 2015], for nodes with two coordinates in the form:

$$\mathbf{N}^* = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \quad (27)$$

where

$\mathbf{N}^*$  is the extended  $N$ -matrix

---

<sup>2</sup>also known as the *normal vector*

<sup>3</sup>also *position matrix* in literature [Konyukhov and Izi, 2015]



Detailed discussion of the role of this matrix and how it relates to the same matrix in the node-to-segment discretization can be found in the section 3.3.3.

### 2.3.3 Node-to-Segment

Compared to node-to-node, the node-to-segment discretization is much less straightforward. Most importantly, the construction of both the projection vector and the extended  $N$ -matrix depends on the type of segment. Apart from the different sorts of element edges, as already mentioned, in OOFEM also the analytical functions are considered a subset of segments, complicating the matter further.

In general, the projection vector of node-to-segment is a vector connecting the node with a contact point lying on the segment. This contact point is found by minimizing a distance function [Yastrebov, 2013], i.e. it is the closest point to the node. The minimization may take the form of a simple formula for linear segments or some sort of closest-point-projection technique for more complicated segments [Konyukhov and Izi, 2015]. The details on how this was achieved for the different segments implemented in OOFEM are described together with the implementation of the given segments in section 3.3.

The gap remains to be simply the size of the projection vector. In the case of node-to-segment, both the vector and the gap are computed from deformed coordinates, however, and so both are updated each FEM iteration.

For the segments which are element edges, the extended  $N$ -matrix takes a form which betrays the origin of its name. For node-to-node, a unit matrix with an appropriate sign belonged to each node in the matrix. Now not only the sign of the allocated forces and stiffness has to be ensured, but also the proper distribution within the contacting element [Konyukhov and Izi, 2015]. Therefore, the extended  $N$ -matrix reads:

$$\mathbf{N}^* = \begin{pmatrix} \mathbf{N}_e(\mathbf{x}_c) & -\mathbf{I}_{(d \times d)} \end{pmatrix} \quad (28)$$

where

$\mathbf{N}_e(\mathbf{x}_c)$  is the  $N$ -matrix of interpolation functions of the element edge evaluated at the point of contact  $\mathbf{x}_c$

$\mathbf{I}_{(d \times d)}$  is a unit matrix of the dimension  $d$ , where  $d$  is the number of node coordinates

The segment assumes the role of the master node here. Because nothing can be assembled to an element or an element edge, the use of the  $N$ -matrix ensures that the forces and stiffness are appropriately distributed among the nodes of the element edge.

### 2.3.4 Penalty Method in FEM

Recalling the penalty-enhanced functional (22), it shall now be investigated how it could be discretized in the spirit of the finite element method.

For a single non-penetration condition expressed by the penalty method, if the energy of contact force were to be isolated from the energy of the rest of the system, it would appear in the form:

$$W^c = \frac{1}{2}pg_c^2 \quad (29)$$

where

$W^c$  would be the contact energy

$p$  would be the penalty parameter

$g_c$  would be the normal gap, acting here as the penetration function (previously in the case of the spring-mass system denoted  $c$ )

Note the lack of integral in this expression. For node-to-node and node-to-segment discretizations, contact forces are discrete, only appearing at points of contact.

This energy formulation can be subjected to the same variational transformation as the whole structural mechanics problem (see equations (3) and (8)). After the introduction of the variational nodal displacements  $\delta\mathbf{d}$ , the variational energy function reads

$$\delta W^c = pg_c\delta g_c \quad (30)$$

The gap function and the new *variational gap*  $\delta g_c$  can be expressed as

$$g_c = \mathbf{N}^T(\mathbf{X}_c + \mathbf{d}_c) \quad (31)$$

$$\delta g_c = \mathbf{N}^T\delta\mathbf{d}_c \quad (32)$$

where

$\mathbf{N}$  is the *normal matrix of contact*, to be explained shortly

$\mathbf{X}_c$  are the undeformed coordinates of the master node(s) followed by the undeformed coordinates of the slave node (in the case of node-to-segment discretization, all nodes of the master segment are used)

$\mathbf{d}_c$  are the nodal displacements of the master and slave nodes similarly arranged

$\delta\mathbf{d}_c$  is the variation of  $\mathbf{d}_c$

Recalling the equation (11), the variational nodal displacements can be completely discarded, as they reliably vanish upon FEM discretization anyway [Patzák, 2019], leaving only the normal matrix in place. The normal matrix is given as [Konyukhov and Izi, 2015]

$$\mathbf{N} = \mathbf{N}^{*T} \frac{\mathbf{n}_0}{\|\mathbf{n}_0\|} \quad (33)$$

where

$\mathbf{N}^*$  is the extended  $N$ -matrix obtained from the contact discretization as described in sections 2.3.2 and 2.3.3

$\mathbf{n}_0$  is the similarly acquired projection vector expressing the direction of contact<sup>4</sup>

With all this in mind, the external force vector of contact can be constructed as [Konyukhov and Izi, 2015]

$$\mathbf{f}_c = \mathbf{N} p g_c \quad (34)$$

To construct a stiffness matrix of contact, equation (30) is subjected to a second variation to yield

$$\delta^2 W^c = p \delta g_c \delta g_c \quad (35)$$

Substituting (32) and disregarding the variational displacements again, the stiffness matrix of contact emerges as [Konyukhov and Izi, 2015]

$$\mathbf{K}_c = p \mathbf{N}^T \mathbf{N} \quad (36)$$

Note that the stiffness matrix does not include the gap anymore.

Both the external force vector and the stiffness matrix are only to be assembled on the condition of the gap being negative. This is ensured in the code by conditional clauses elsewhere, however it could be expressed formally by multiplying both equations (34) and (36) by the Heaviside function in the form of  $H(-g_c)$ .

### 2.3.5 Lagrangian Multiplier Method in FEM

To construct the FEM formulation of the Lagrangian multiplier functional (19), the same process as with the penalty method can be applied<sup>5</sup>. The contact energy in this case is simply

$$W^c = \lambda g_c \quad (37)$$

where

---

<sup>4</sup>differing from  $\mathbf{n}$  only in the case of perfect contact in node-to-segment, when the actual projection is a zero vector, but for this purpose the direction of contact is still used

<sup>5</sup>Same as for the penalty method a single non-penetration condition is sufficient to explain the workings of the method. In a real case, there may be multiple Lagrangian multiplier terms in the functional, yet the process is the same for each of them.

$\lambda$  is the Lagrangian multiplier

$g_c$  has the same meaning as for the penalty method: the gap function

Again there is no integral in the energy expression. The Lagrangian multipliers as well are discrete and only apply at a single point in the continuum.

However, it has to be remembered that now not just the displacements (hidden inside the gap function), but also the Lagrangian multiplier  $\lambda$  are unknown values. For the construction of the FEM weak form to work, a *variational Lagrangian multiplier*, denoted  $\delta\lambda$ , has to be introduced. Considering this, the variation of energy transforms into

$$\delta W^c = \delta\lambda g_c + \lambda \delta g_c \quad (38)$$

where

$\delta\lambda$  is the variational Lagrangian multiplier

$\delta g_c$  is the variational gap function

Physically, the variational energy is nothing more and nothing less than an expression of the work carried out by the internal forces of the system on the virtual degrees of freedom [Patzák, 2019]. With that in mind, the following equality might be inferred:

$$\delta W^c = f_{int,\lambda}^c \delta\lambda + (\mathbf{f}_{int,u}^c)^T \delta \mathbf{d} = \delta\lambda g_c + \lambda \frac{\partial g_c}{\partial \mathbf{d}} \delta \mathbf{d} \quad (39)$$

where

$\delta \mathbf{d}$  are the *variational nodal displacements*; compare equations (12) and (25)

$\frac{\partial g_c}{\partial \mathbf{d}}$  is in fact, recalling expression (32), the transposed normal matrix of contact  $\mathbf{N}^T$

whence the definitions of internal forces are apparent:

$$\mathbf{f}_{int,u}^c = \lambda \mathbf{N}, \quad f_{int,\lambda}^c = g_c \quad (40)$$

In this discrete form, the equilibrium equations, divided into a displacement part and a Lagrangian-multiplier part, can be expressed in the form

$$\begin{pmatrix} \mathbf{f}_{int} \\ 0 \end{pmatrix} + \begin{pmatrix} \mathbf{f}_{int,u}^c \\ f_{int,\lambda}^c \end{pmatrix} = \begin{pmatrix} \mathbf{f}_{ext} \\ 0 \end{pmatrix} \quad (41)$$

where

$\mathbf{f}_{int}$  is the vector of contact-unrelated internal forces

$\mathbf{f}_{ext}$  is the vector of external forces

Note how the two force expressions differ in the place where they are assembled. After applying the interpolation functions, the value of  $f_{int,\lambda}^c$  is only the gap function, forming the non-penetration condition (compare with the spring-mass case, where this is exactly the equation (21)). On the other hand, the vector  $\mathbf{f}_{int,u}^c$  is the contribution of the Lagrangian multiplier (the contact force) to the standard FEM equilibrium equations (in the spring-mass case, represented by equation (20)).

Similarly to the division in expression (41), the stiffness matrix as well can be divided similarly into four independent blocks as follows:

$$\mathbf{K}_c = \begin{bmatrix} \mathbf{K}_{uu}^c & \mathbf{K}_{u\lambda}^c \\ \mathbf{K}_{\lambda u}^c & K_{\lambda\lambda}^c \end{bmatrix} \quad (42)$$

The respective blocks represent variations of internal forces with respect to the degrees of freedom (either displacements or Lagrangian multipliers) [Patzák, 2019]. Therefore, it can be easily shown that both the diagonal blocks are in fact empty. The first diagonal block involves a derivative of the term  $\lambda \mathbf{N}$  by displacements. Neither the Lagrangian multiplier nor the normal matrix of contact depend on them, and therefore

$$\mathbf{K}_{uu}^c = \frac{\partial \mathbf{f}_{int,u}^c}{\partial \mathbf{d}} = \mathbf{0} \quad (43)$$

The second diagonal block involves a derivative of the gap function by the Lagrangian multiplier. The gap function is entirely independent of the Lagrangian multiplier and so

$$K_{\lambda\lambda}^c = \frac{\partial f_{int,\lambda}^c}{\partial \lambda} = 0 \quad (44)$$

Only the diagonal blocks are non-zero. In the case of frictionless contact, the stiffness matrix shall be symmetrical [Konyukhov and Izi, 2015], and that is indeed the case as seen in the expressions

$$\mathbf{K}_{u\lambda}^c = \frac{\partial \mathbf{f}_{int,u}^c}{\partial \lambda} = \mathbf{N} \quad (45)$$

$$\mathbf{K}_{\lambda u}^c = \frac{\partial f_{int,\lambda}^c}{\partial \mathbf{d}} = \frac{\partial g_c}{\partial \mathbf{d}} = \mathbf{N}^T \quad (46)$$

$$\mathbf{K}_{\lambda u}^c = (\mathbf{K}_{u\lambda}^c)^T \quad (47)$$

The only component of stiffness apart from interpolation functions is therefore the normal matrix of contact  $\mathbf{N}$ . To clarify, see the reviewed shape of the stiffness matrix:

$$\mathbf{K}_c = \begin{bmatrix} \mathbf{0} & \mathbf{N} \\ \mathbf{N}^T & 0 \end{bmatrix} \quad (48)$$

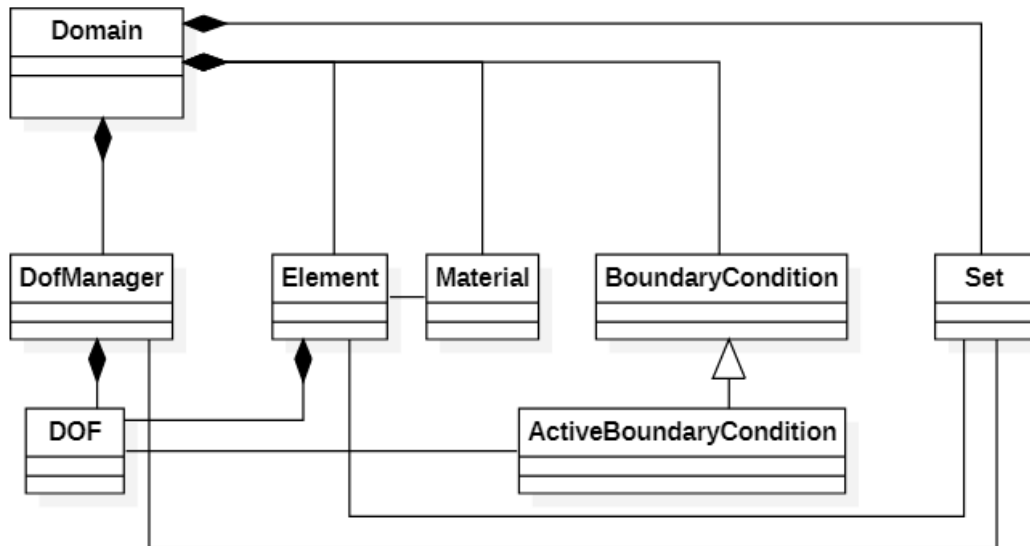
### 3 Implementation

In this section, an example of practical handling of the insofar reviewed theoretical findings shall be discussed. The core of work on this thesis has been the implementation of contact algorithms into the OOFEM software [Patzák, 2000]. A number of problems were encountered, stemming above all from a need to adapt those algorithms to smoothly integrate into the existing OOFEM framework, and from attempts to utilize this existing framework to the maximum of its possibilities and avoid duplicating existing software processes.

#### 3.1 The Specifics of OOFEM

##### 3.1.1 Inner Structure

OOFEM is developed, as its name itself references, as an object-oriented software. This means that all specific attributes of a FEM computation task (i.e. elements, nodes, material models and similar) are presented within the code as objects bound together by a complex system of relations and inheritance [Patzák, 2000].



**Figure 2:** A simplified diagram of object relations within OOFEM

The base object class intended to store all task data in program memory and provide access to them is the `Domain` class. Figure 2 reproduces the most important relationships within the task data in graphical form and demonstrates the way in which those are centered around `Domain`. It is a picture simplified to a very large degree, omitting many other object classes not as pertinent to contact problems. Also, it should be noted that

OOFEM itself is a universal software intended for most imaginable finite element calculations. The program core, therefore, is intended as universal without any bias towards any type of problem. All code specific to a certain problem type is contained within its own package. As contact problems are problems of structural mechanics, it is the `sm` package in which they were developed and which is the focus here. In that package, subclasses of the universal classes pictured in the diagram are used, for example `StructuralElement` as a subclass of `Element` or `Node` as a subclass of `DofManager` [Patzák, 2000].

In the following sections, functions of and interactions between various OOFEM classes is discussed in closer detail in regard to the changes introduced by the performed contact implementation.

### 3.1.2 Note on OOFEM Coding Conventions

This is a short explanation of some conventions in OOFEM that are, further in the text of this thesis, assumed as universal [Patzák, 2000].

- As visible from the class and function names mentioned, names are always given in `TitleCase` for classes and `camelCase` for functions.
- Function results, especially those which are non-primitive, are passed in as reference rather than returned (by copy or reference). Any mention of a "return value" usually does not mean an actual return value in C++ sense, but rather this reference serving as an "out-parameter". Which also explains how multiple return values for some functions are achieved.

## 3.2 Implementation of Node-to-Node Contact

Node-to-node contact is substantially simpler than other forms of contact algorithms. What should be above all pointed out is the fact that it does not require an implementation of any new type of object into the software framework. Nothing is also required in terms of contact search, at least in this implementation, where the decision was made to limit the usability to small-deformation problems, in which it is entirely reasonable to leave the definition of master-slave node pairs to the user [Konyukhov and Izi, 2015].

In this thesis, two versions of node-to-node contact conditions were implemented, one of them using the penalty approach as theoretically discussed in 2.2.3 and the other using Lagrange multipliers introduced in 2.2.2.

### 3.2.1 Use of the `ActiveBoundaryCondition` class

While node-to-node contact theoretically only consists of two nodes interacting with each other, there is the question of where within the object-oriented code the associated computations should be performed. An obvious solution would be to create subclasses of

`Node` for both master and slave nodes and add the necessary functions to them. The usual solution for contact problems is to create fictional elements in the space between the contacting bodies which track the node pairs [Yastrebov, 2013].

However, neither of this is necessary in OOFEM. The OOFEM code contains a special feature implemented as the class `ActiveBoundaryCondition`. As the name hints, this is a boundary condition capable of assembling its own contribution to the global stiffness matrix and to the vector of external forces and even of managing its own degrees of freedom unassociated with any regular DOF managers (nodes). This is achieved by `ActiveBoundaryCondition` (and all classes derived from it) implementing all the necessary functions otherwise mostly found in elements, chief amongst them the functions `assemble()`, `assembleVector()` and `giveLocationArrays()` [Patzák, 2000].

Utilizing `ActiveBoundaryCondition`, there is actually no need to alter the code of the `Node` class in any way. Subclasses of `ActiveBoundaryCondition` were created, which hold lists of master and slave nodes corresponding to each other, perform all the necessary computations and - in the case of the Lagrange multiplier approach - also manage the necessary additional global equations.

### 3.2.2 Node-to-Node Penalty Contact

The single newly implemented class is visualised in relation to other parts of the OOFEM environment in figure 3. Only functions immediately related to contact computational functionality are pictured.

As discussed in the previous section, `ActiveBoundaryCondition` was used as a base class for a new class named `Node2NodePenaltyContact`, which achieves all necessary functionality by communicating with its known master and slave nodes. For this communication, simple existing functions are used; actually only three are necessary: one to determine the DOFs managed by the given node and two to determine the initial and deformed coordinates of the node.

The initialization of a node-to-node penalty contact boundary condition is triggered by a line in the OOFEM input file consisting of the boundary condition's name and the necessary parameters. An example of such a line can be found in listing 1.

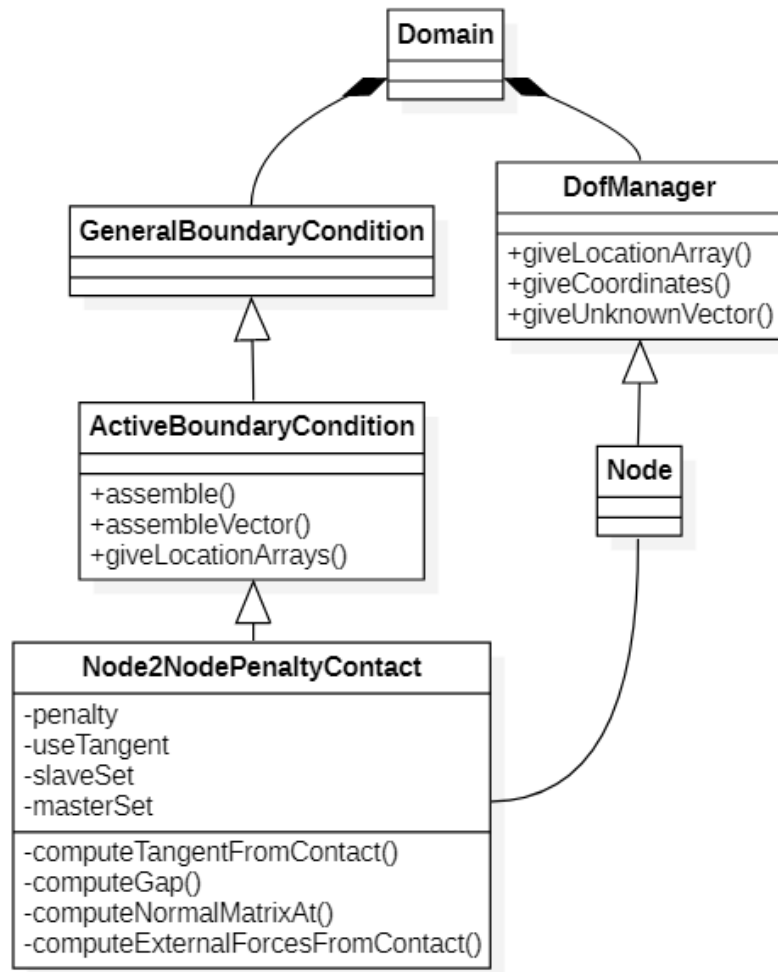
```
n2npenaltycontact 5 loadTimeFunction 1 penalty 1.e10 masterset 2 4 3
  slaveset 2 5 6 usetangent
```

**Listing 1:** An excerpt from an OOFEM input file initializing a node-to-node penalty contact boundary condition

Apart from the `loadTimeFunction` parameter (controlling when the boundary condition shall be applied) inherited from the `GeneralBoundaryCondition` class, there are four parameter input keywords, namely



- penalty,
- masterset,
- slaveset and
- usetangent.



**Figure 3:** Implementation of node-to-node penalty contact within OOFEM environment

The `penalty` parameter is a real number defining the penalty and is an obligatory parameter. The penalty has to be defined by the user, as its effect on the computation is highly task-dependent, as discussed in the 2.2.3 and 4.4 sections of this thesis.

Next are the `masterset` and `slaveset` parameters. Those take one ordered integer array each, defining the slave and master node ids. The first master node corresponds to the first slave node, the second to the second etc. The boundary condition cycles through the pairs so defined when assembling the stiffness matrix and external forces vector of

contact; only one boundary condition is, therefore, necessary for a single input file. A completely arbitrary number of node pairs can be defined, provided that the sizes of the master and slave arrays stay the same so that each node has a clearly defined counterpart. One node id may appear more than once in the list(s), if it shall correspond to more than one other node. The set functionality natural to OOFEM was not used here in favor of using those integer arrays; this is because a set always returns an ordered list of nodes internally, making it impossible at runtime to correctly pair the master and slave nodes as intended [Patzák, 2000].

Finally, the `usetangent` parameter is a boolean value that toggles whether the tangential stiffness matrix shall be assembled at all. If not included, the `assemble()` function of the `Node2NodePenaltyContact` class is deactivated.

Four new private functions are defined within the `Node2NodePenaltyContact` class to aid with the inner computations.

The `computeGap()` function takes two nodes as arguments, returning the distance between them in the direction of contact. The direction of contact is defined here as the vector between the initial positions of the nodes; this avoids a lot of problems with node interactions, however it effectively limits the functionality to small deformation problems.

The `computeNormalMatrixAt()` function also takes two nodes as arguments. It computes the normal vector between them, this again being the not-updated vector of initial coordinates. This vector is then included assembled into the normal matrix of contact in the form of

$$\mathbf{N} = \begin{pmatrix} \mathbf{n} \\ -\mathbf{n} \end{pmatrix} \quad (49)$$

where

$\mathbf{N}$  is the returned normal matrix

$\mathbf{n}$  is the distance vector of the initial node coordinates

This serves as an intermediate step for the next function, `computeExternalForcesFromContact()`, which takes this matrix and multiplies it by the penalty and gap value, if the gap is negative (therefore, a penetration occurred), or conversely by zero if the gap is positive (and therefore there is no penetration). The inner workings of `computeExternalForcesFromContact()` can be summarized by the following equation:

$$\mathbf{f}_{ext,c} = \mathbf{N}pgH(-g) \quad (50)$$

where

$\mathbf{f}_{ext,c}$  is the vector of external forces returned

$\mathbf{N}$  is the normal matrix as defined by (49)

$p$  is the penalty value

$g$  is the gap value

$H$  is the Heaviside function

This vector of external forces, thanks to the shape of  $\mathbf{N}$ , already encompasses the contribution of both nodes to the global vector.

In a very similar fashion, the `computeTangentFromContact()` function also utilizes the enlarged normal matrix  $\mathbf{N}$  to construct the tangential stiffness contribution of both involved nodes in the form (compare also equation (36))

$$\mathbf{K}_c = (\mathbf{N}^T \mathbf{N}) p H(-g) \quad (51)$$

where

$\mathbf{K}_c$  is the tangential stiffness matrix returned

$\mathbf{N}$  is the normal matrix as defined by (49)

$p$  is the penalty value

$H$  is the Heaviside function

$g$  is the gap value

Note that thanks to not updating the normal vector between nodes and leaving it to be the initial direction and thanks to the penalty being a constant parameter, this is actually a constant value which is only switched on or off by the Heaviside function depending on whether contact occurs or not.

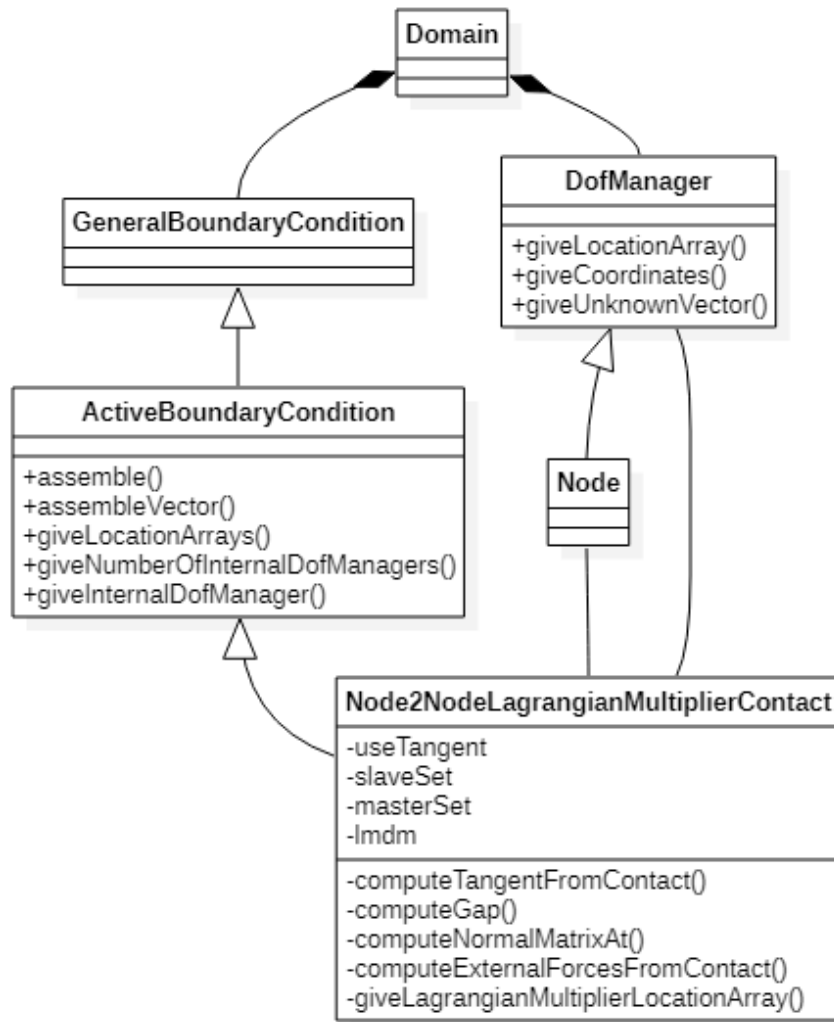
The combination of the inner workings of those four functions means that very little actual work is left to be done by the inherited `assemble()` and `assembleVector()` functions. Those just cycle through the master-slave node pairs, ask them for their localization code numbers and assemble the computed stiffness matrices or external forces vectors, respectively, to the global matrix or vector.

Finally, the `giveLocationArrays()` function deserves a brief mention. Inherited from the `ActiveBoundaryCondition` class, this function gives the caller information about the degrees of freedom which the boundary condition binds together. This is necessary for the use of advanced matrix storage techniques by the FEM solver. If the global stiffness matrix is stored in a skyline matrix, the information on interacting DOFs is used to allocate non-zero matrix members [Patzák, 2000] [Felippa, 1975].

### 3.2.3 Node-to-Node Lagrange Multiplier Contact

The implementation of a node-to-node contact boundary condition using the Lagrangian multiplier approach is largely similar to the simpler penalty condition. It still holds that

there is only one necessary class to be implemented into the existing OOFEM code, this time aptly named `Node2NodeLagrangianMultiplierContact`. The essential relationships of this class and the rest of OOFEM are again displayed by the means of an UML diagram - in figure 4.



**Figure 4:** Implementation of node-to-node Lagrangian multiplier contact within OOFEM environment

The class `Node2NodeLagrangianMultiplierContact` is again derived from `ActiveBoundaryCondition`. This time though, even more capabilities of the base class are exploited. As is visible in the UML diagram, now the functions `giveNumberOfInternalDofManagers()` and `giveInternalDofManager()` are inherited. Those are essential for maintenance of the newly introduced DOFs which correspond to the Lagrangian equations. The new private field `lmdm` inside the inherited class is the standard C++ `vector` array [Virus, 2018] holding pointers to the OOFEM class `DofManager`. This is the parent class to `Node` (or, more eloquently put, the `Node` class is the special case of `DofManager` applicable to problems of structural mechanics [Patzák,

2000]). Here, the `DofManager` objects are however used to conveniently represent the virtual Lagrangian DOFs.

In listing 2, an example of an input file line defining a node-to-node Lagrangian multiplier contact boundary condition is provided. Not much has changed from the previously referred listing 1, which was an example of node-to-node penalty contact. Only the initial keyword is different and the `penalty` parameter is missing.

```
n2nlagrangianmultipliercontact 5 loadTimeFunction 1 masterset 2 4 3
  slaveset 2 5 6 usetangent
```

**Listing 2:** An excerpt from an OOFEM input file initializing a node-to-node Lagrangian multiplier contact boundary condition

Yet again, the nodes involved are specified by two ordered integer arrays, making a sole boundary condition responsible for any number of contact points desired. Only corresponding master-slave pairs are checked against each other and a node may be mentioned more than one time in the arrays. Because each master-slave pair receives one Lagrangian multiplier, a `Node2NodeLagrangianMultiplierContact` boundary condition introduces precisely as many new DOFs into the task as is the size of the `masterset` and `slaveset` arrays.

The `usetangent` parameter again allows to prohibit application of the tangential stiffness matrix from contact (by its omission).

Of the private functions within `Node2NodeLagrangianMultiplierContact` class, the `computeGap()` and `computeNormalMatrixAt()` functions are entirely identical to their counterparts within `Node2NodePenaltyContact` and the reader is kindly asked to refer to section 3.2.2 for an explanation of their purpose and inner workings. The `giveLocationArrays()` function is more complicated here thanks to the more complex relationships among the DOFs worked with, however its purpose has also been already sufficiently elaborated on in said section.

On the other hand, the `computeExternalForcesFromContact()` function is significantly simpler than the function in penalty contact, which needed to prepare a vector of external forces<sup>6</sup> to be allocated to the DOFs of both affected nodes. Here it is the Lagrange multiplier which takes over the role of an affected DOF. Therefore, the vector of external forces assumes the single-value form given as:

$$\mathbf{f}_{ext,c} = (gH(-g)) \quad (52)$$

where

---

<sup>6</sup>In theory, this is a vector of internal forces (as seen in (40)). However considering the shape of the equation system (41), it makes no difference on which side of the equality it is actually assembled in the code.

$g$  is the gap as given by the `computeGap()` function

$H$  is the Heaviside function

A similar change occurs in the `computeTangentFromContact()` function. While previously the result was an  $2d \times 2d$  matrix, where  $d$  was the dimension of the task domain (i.e. the number of DOFs per node), now only an  $2d \times 1$  vector is returned<sup>7</sup>, which is nothing more than the result of the `computeNormalMatrixAt()` function<sup>8</sup>. Another minor change from penalty contact is the fact that as a second return value, the computed gap is given. This enables the `assemble()` function to decide whether or not to actually assemble the matrix.

The `assemble()` function is slightly more complicated than its penalty method counterpart. The reason is the fact that the additional equations introduced by the Lagrangian DOFs should only be active when contact occurs. A significant question arises about how to ensure that. It would be possible to properly disable the DOFs when they are not used. However, that would mean changing the size of the global tangent stiffness matrix and external forces vector possibly even each iteration, a process that could prove very costly, depending on the method of matrix storage used by the solver [Gould et al., 2007]. A decision was therefore made to not disable the DOFs, and just, in the case that contact is not occurring, instead of assembling the stiffness matrix, assemble a 1 to the diagonal position corresponding to the DOF in question. Given that in such case, zero is assembled to the external forces vector (see equation (52)), the resulting Lagrangian equation looks as follows:

$$1 \cdot \lambda_i = 0 \tag{53}$$

where

$\lambda_i$  is the  $i$ -th Lagrangian multiplier

$i$  is the code number of the DOF for which contact is not occurring

This essentially equals completely disabling the DOF, but does not require resizing the global matrices. Obviously, if this happens with a larger number of DOFs, it may also disturb some equation solvers [Gould et al., 2007]. This effect remains to be investigated.

For the case when contact is occurring, the `assemble()` function assembles the stiffness matrix (column vector) returned by the `computeTangentOnContact()` function to

---

<sup>7</sup>For purposes of coding practicality, this vector is in fact still returned as an instance of the `FloatMatrix` class, rather than the admittedly more appropriate `FloatArray`. This is because the `assemble()` function, to which the result is passed, would otherwise have to convert it anyway to comply with the existing code it uses (namely the function `SparseMtrx::assemble()`) [Patzák, 2000]. The function `computeTangentFromContact()` therefore actually acts just as a wrapper which takes the `FloatArray` returned by `computeNormalMatrixAt()` and passes it further as a `FloatMatrix`.

<sup>8</sup>The reason is apparent from (48).

the column of the Lagrangian multiplier DOF and the rows of the node DOFs. The transposition of the same vector is then assembled to the symmetrical location.

As well as `assemble()`, the `assembleVector()` function too is more complicated in the Lagrangian formulation. Here the disabling of DOFs is not an issue - the Heaviside function in equation (52) sees to that. However, the vector of internal forces is now to be assembled. In OOFEM structure, this is done by this same `assembleVector()` function, the two vectors being distinguished by the `CharType` function argument [Patzák, 2000].

For the external vector, just the single value given by the `computeExternalForcesFromContact()` function is assembled to the row of the Lagrangian multiplier DOF. For the internal vector, a new vector is constructed by the formula:

$$\mathbf{f}_{int,c} = \lambda_i \mathbf{N} \quad (54)$$

where

$\mathbf{f}_{int,c}$  is the internal force vector

$\lambda_i$  is the  $i$ -th Lagrangian multiplier

$i$  is the code number of the Lagrangian multiplier DOF

$\mathbf{N}$  is the normal matrix given by the `computeNormalMatrixAt()` function

This internal force vector is then assembled to the rows of node DOFs.

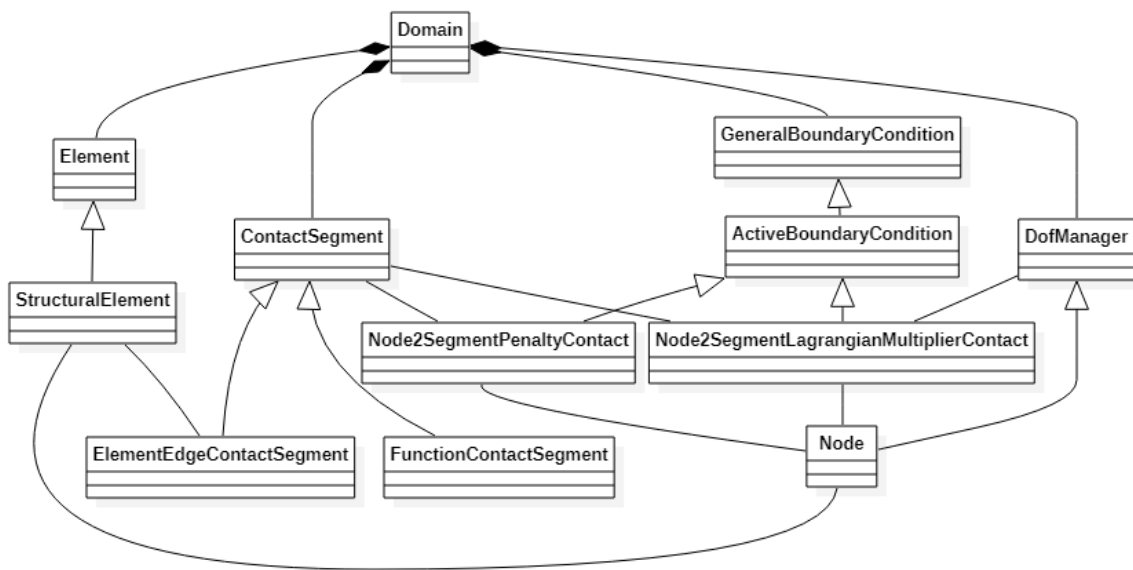
For good measure, the fact shall be noted that all described actions within the functions `assemble()` and `assembleVector()` are always performed for all master-slave node pairs, through which the functions cycle.

## 3.3 Implementation of Node-to-Segment Contact

### 3.3.1 Class Structure

In comparison with the so far described node-to-node implementations, node-to-segment contact poses a significantly greater challenge. No longer is it sufficient to use the `ActiveBoundaryCondition` class to derive subclasses that can do everything. Those boundary conditions are still implemented with as much similarity to the node-to-node implementation as possible, however it is necessary for them to interact not only with nodes, but with new classes representing the contact segments. The design of the class system within OOFEM is pictured in figure 5.

The decision has been made to introduce a new `ContactSegment` class to OOFEM as a completely new type of a FEM component. This is a purely virtual class derived from `FEMComponent`, which serves as a parent class to various classes representing different types of contact segments. In the hierarchy of OOFEM, it is on the same level as other general parent classes, like for example `Element` or `Material`.



**Figure 5:** An overview of all classes pertinent to node-to-segment contact and their hierarchy

This approach to contact segment implementation means that the algorithms of the central `Domain` class had to be expanded to acknowledge the existence of `ContactSegment`, especially with respect to loading segments from input files and notifying them about the progress of computation [Patzák, 2000]. Notably, the functions `postInitialize()`, called by `Domain` on all contact segments after an input file finishes reading, and `updateYourself()`, called by `Domain` whenever convergence is reached in an iteration step, were introduced.

For defining contact segments, a new section appears in the OOFEM input file, located between the element list above and the cross-section list below. If any contact segments are defined, the number of them has to be given by the `ncontactseg` keyword in the input file header. Listing 3 shows an example of an input file with such section.

```

### Elements
planestress2d 1 nodes 4 1 2 3 4 crossSect 1 mat 1
Truss2d 2 nodes 2 5 6 crossSect 1 mat 1 cs 1
### Contact Segments
ElementEdgeContactSegment 1 edgeset 5
### CrossSection
SimpleCS 1 thick 1 area 1
  
```

**Listing 3:** Contact segment definition block within the OOFEM input file

It shall be emphasised that this approach of creating a broad virtual class providing all different contact segments with a universal interface has a significant advantage for future code development. It is now possible for any node-to-segment boundary condition



to work with any type of contact segment, even with such as are not yet implemented and even with different types at the same time. It is expected that this should to a large extent encompass future expansion in other task dimensions as well, as is discussed further in section 3.4.

### 3.3.2 Boundary Conditions

As is the case with node-to-node contact, two boundary conditions are implemented for node-to-segment contact too. Those are, as expected, named `Node2SegmentPenaltyContact` and `Node2SegmentLagrangianMultiplierContact`.

There are obvious similarities between the node-to-node and node-to-segment classes. This section shall only discuss what is different in node-to-segment in comparison to node-to-node. For a detailed description of the identical parts, sections 3.2.2 and 3.2.3 can be referred.

Both new boundary condition classes, their functions and relevant functions of other intertwined classes are displayed in figure 6.

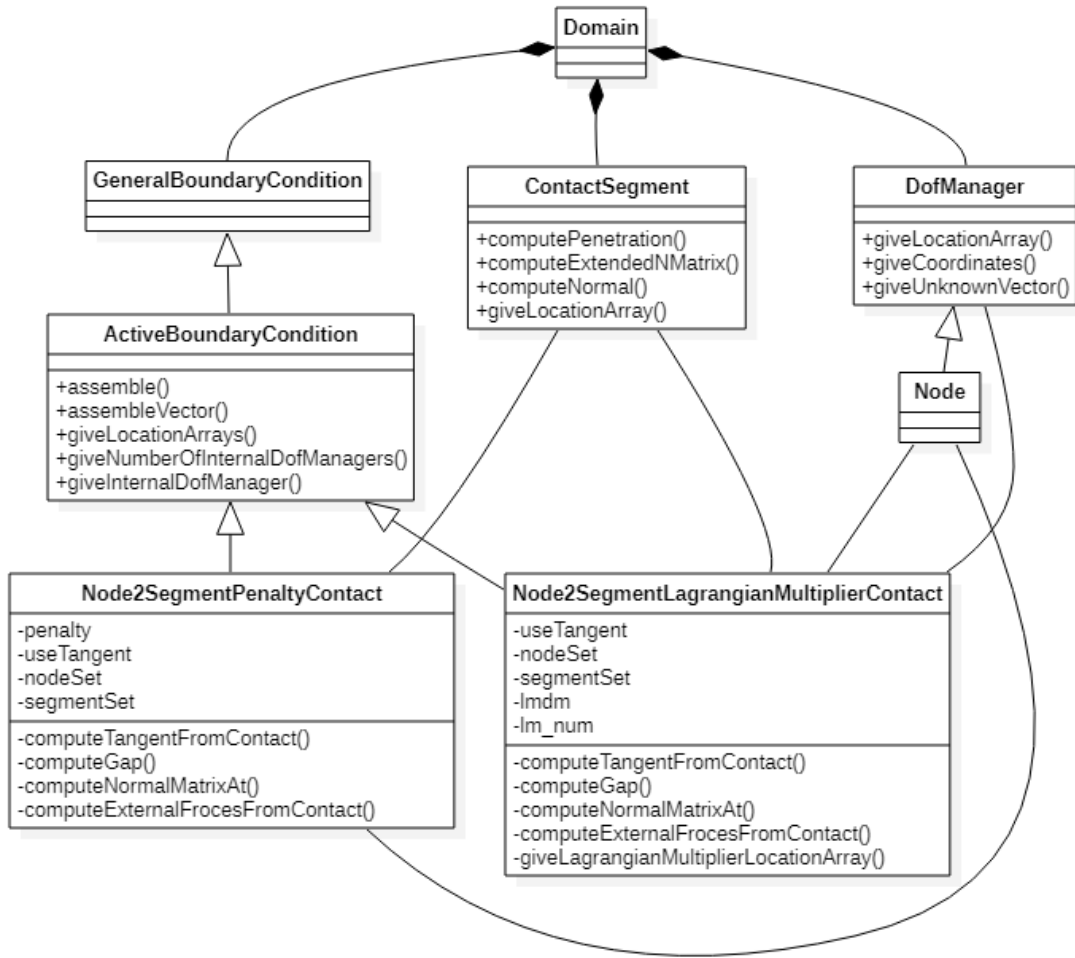
One very notable difference from the node-to-node implementation is how the node-segment pairs are handled. In node-to-node contact, the master and slave nodes are paired one to one, decided by the user. In node-to-segment, this is no longer the case. Listings 4 and 5 depict that the `masterset` and `slaveset` keywords have been replaced by `nodeset` and `segmentset`. No specific master-slave relationship exists here, since in the stiffness matrix and force vector assembly, each segment is checked for contact with all nodes. This is a design decision which has been made in anticipation of more complex tasks, possibly accounting for large deformations as well, where it may not be always precisely clear which segment a node comes into contact with, or whether even this will be always the same segment throughout the course of the computation. As a consequence of this, more caution is expected on the side of the user, because if larger numbers of nodes or segments are entered into one boundary condition, computation complexity could rise significantly. It is no longer true that only one contact boundary condition is sufficient per task - on the contrary, it may often prove prudent to use multiple of them.

```
n2spenaltycontact 5 loadTimeFunction 1 penalty 1.e10 nodeset 1 5
segmentset 1 1 usetangent
```

**Listing 4:** An excerpt from an OOFEM input file initializing a node-to-segment penalty contact boundary condition

```
n2slagrangianmultipliercontact 5 loadTimeFunction 1 nodeset 1 5
segmentset 1 1 usetangent
```

**Listing 5:** An excerpt from an OOFEM input file initializing a node-to-segment Lagrangian multiplier contact boundary condition



**Figure 6:** Implementation of node-to-segment boundary conditions within OOFEM environment .

Again, the node and segment sets are entered by means of an integer array. Thanks to the differences described above, the order of nodes or segments is no longer of consequence and the OOFEM set functionality could theoretically be used. For the sake of similarity with node-to-node however, the integer array input has been retained. The loading of segments by their input index utilizes the newly implemented function `giveContactSegment()` in the `Domain` class.

This is also the reason of the new `lm_num` internal parameter of the `Node2SegmentLagrangianMultiplierContact` class. It is nothing more than the number of Lagrange multipliers (which is a multiple of the dimensions of `nodeset` and `segmentset`).

Of the internal functions of the boundary condition classes, all of `assemble()`, `assembleVector()`, `computeTangentFromContact()` and `computeExternalForcesFromContact()` remained essentially the same as in node-to-node contact, respecting of course the various differences between penalty and

Lagrangian multiplier classes.

The `computeGap()` function is now very simple. In node-to-node contact, it calculated the gap between the two nodes. Computing the gap between node and segment, however, is delegated to the contact segments themselves, allowing different contact segments to apply whichever definition of the gap and whichever way to calculate it they deem most appropriate. The `computeGap()` function only calls the `ContactSegment::computePenetration()` function, passing the node in question, and relays the obtained result (in the form of a real number) further.

For similar reasons, the `computeNormalMatrixAt()` function behaves differently. Firstly, the same as the gap computation, the normal vector computation is a task for the contact segments, by means of the `ContactSegment::computeNormal()` function, which again takes the node as an argument. This effectively decouples the boundary condition class from the decisions whether to allow for large deformations, how to define direction of contact and similar. Secondly, in node-to-node contact, the normal vector is just duplicated (see equation (49)) to allow for allocation to both node DOFs. Contact segments, however, can have various numbers and orderings of DOFs. This is obtained by the means of an "extended  $\mathbf{N}$ -matrix"<sup>9</sup>. This matrix is obtained from the contact segment using its `computeExtendedNMatrix()` function and the normal matrix is then computed as

$$\mathbf{N} = (\mathbf{N}^*)^T \mathbf{n} \quad (55)$$

where

$\mathbf{N}$  is the normal matrix of contact

$\mathbf{N}^*$  is the extended  $\mathbf{N}$ -matrix of the contact segment

$\mathbf{n}$  is the unit normal vector between node and segment

Owing to this separation from the contact segment, the boundary conditions are entirely independent of how the different types of segments order their degrees of freedom and how many of them they have, as long as the last part of the extended  $\mathbf{N}$ -matrix is a negative unit matrix providing for the node DOFs. The boundary condition does not even need to care how many DOFs a node has, as long as the dimensions of  $\mathbf{N}^*$  and  $\mathbf{n}$  agree to allow for the multiplication. Both of those values are naturally provided by the contact segment.

Careful observation of equation (55) betrays another secret. If we considered an extended  $\mathbf{N}$ -matrix in the form

$$\mathbf{N}^* = \begin{pmatrix} \mathbf{I}_{(d \times d)} & -\mathbf{I}_{(d \times d)} \end{pmatrix} \quad (56)$$

where

---

<sup>9</sup>Named for the fact that in case of an element (edge), it is truly just the extended matrix of base functions. In literature, as for example [Konyukhov and Izi, 2015], this is also known as the  $\mathbf{A}$ -matrix.

$\mathbf{I}_{(x \times x)}$  is an unit matrix of the dimension  $x$

$d$  is the dimension of the FEM task (i.e. the number of DOFs per node)

and (provided that the normal vector  $\mathbf{n}$  also is of dimension  $d$ ) substituted it into equation (55), we would obtain a normal matrix of contact in the shape

$$\mathbf{N} = (\mathbf{N}^*)^T \mathbf{n} = \begin{pmatrix} \mathbf{I}_{(d \times d)} \\ -\mathbf{I}_{(d \times d)} \end{pmatrix} \mathbf{n} = \begin{pmatrix} \mathbf{n} \\ -\mathbf{n} \end{pmatrix} \quad (57)$$

which is the same as the node-to-node matrix shown in equation (49). It lets itself be concluded that node-to-node contact is, in fact, but a special case of node-to-segment contact.

As a final note, the `giveLocationArrays()` functions, needed for the allocation of global matrix members, are of course adapted to extract segment location arrays using the `ContactSegment::giveLocationArray()` function and changed in respect to the fact that all contact segments can interact with all nodes.

#### 3.3.3 Element Edge Contact Segment

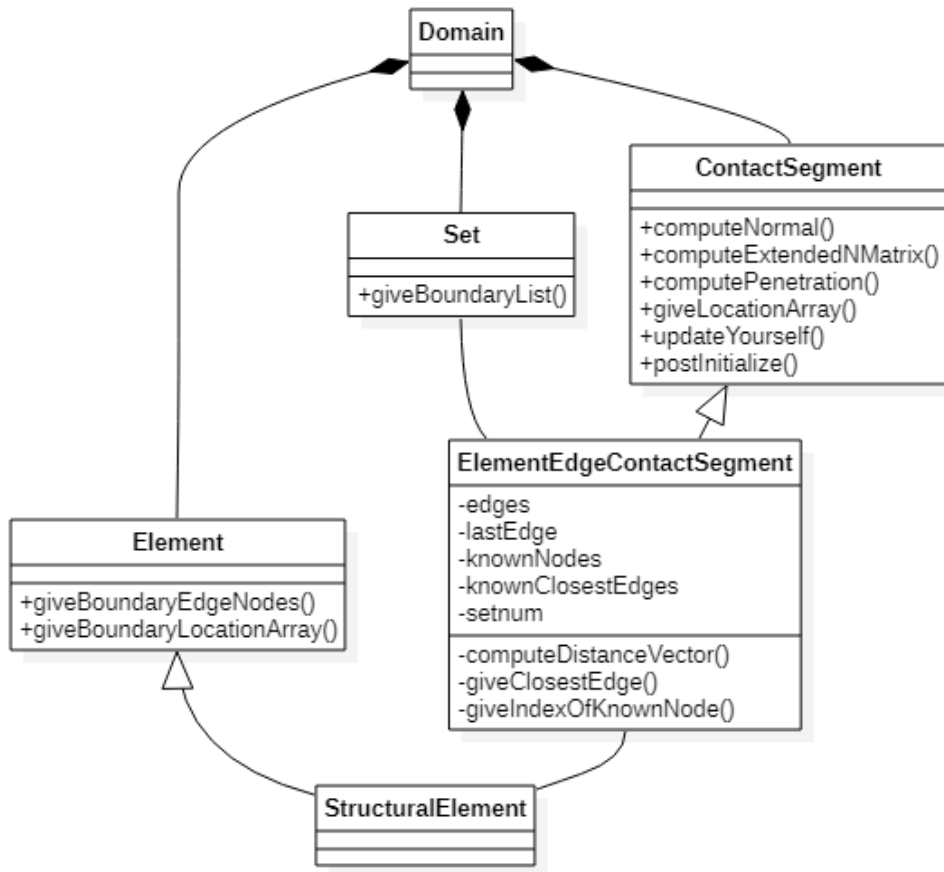
The most natural type of contact segment and the one anticipated to see the widest practical use is the element edge segment. In the OOFEM implementation, it is realized as a set of boundaries of 2D linear elements. The expected use is to form a segment encompassing numerous neighboring elements, thus defining an edge of a discretized body to be a single contact segment coming into contact with external nodes.

The class implementing this is called `ElementEdgeContactSegment`, being a subclass of the general `ContactSegment` class. Its implementation is described in figure 7.

As indicated in said figure, the contact segment at last makes use of the most handy OOFEM set functionality, represented on the code side by the `Set` class. In listing 6, it is apparent how this is achieved. The initialization of the contact segment itself is very straightforward, the only parameter needed is the index of the set used, given by the `edgeset` keyword. The index is stored in the private `setnum` variable and in the `postInitialize()` function the set is loaded<sup>10</sup>. Using the `Set::giveBoundaryList()` function, an integer array of element edges is obtained, which is what is actually then stored in the `ElementEdgeContactSegment` class throughout the computation. This integer array of element edges has a specific format - it is twice as large as the number of element edges and two successive numbers always indicate an index of an element and an index of an edge within that element [Patzák, 2000].

---

<sup>10</sup>This is necessary for the reason that sets are defined later than contact segments (as the very last thing in fact) and therefore referring to a set is only possible after the entire input file is read and processed [Patzák, 2000]



**Figure 7:** Implementation of an element edge contact segment within OOFEM environment .

```

### Contact Segments
ElementEdgeContactSegment 1 edgeset 5
[...]
#element edge set
Set 5 elementboundaries 2 1 3
  
```

**Listing 6:** An excerpt from an OOFEM input file initializing an element edge contact segment, including the initialization of the set of element edges referred to

In the following paragraphs, a description of the `ElementEdgeContactSegment` functionality is provided. For the sake of clarity, it shall proceed in the direction in which the functions are nested within each other.

The function `computeDistanceVector()` is the underlying base of the class. Its purpose is to perform the geometric calculations that give a projection of a point on a line. Two-dimensional space and a straight linear segment are assumed. The function takes coordinates of two points of a line segment and of one external point for parameters. Several return values are provided, a vector of the projection, coordinates of the point

### 3.3 Implementation of Node-to-Segment Contact

---

of projection and a logical value determining whether this point lies in between the line points.

Figure 8 illustrates geometrically what is computed. First, the vector  $(a_n, b_n)$  of the edge line is determined as the difference in edge point coordinates. Switching its coordinates and multiplying one of them by  $-1$  gives the perpendicular vector, marked  $(a_e, b_e)$ . Both the edge line and the perpendicular line can be now expressed by equation using for each line the components of the perpendicular vector as such:

$$a_e x + b_e y + c_e = 0 \quad (58)$$

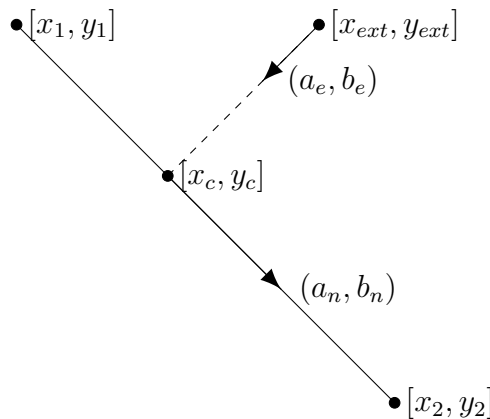
$$a_n x + b_n y + c_n = 0 \quad (59)$$

The values  $c_e$  and  $c_n$  are obtained by substituting the coordinates of one of the edge points and of the external point into equations (58) and (59), respectively. Now the contact point  $[x_c, y_c]$  is found, using the formula

$$x_c = \frac{-(c_e b_n - c_n b_e)}{a_e b_n - a_n b_e} \quad (60)$$

$$y_c = \frac{-(a_e c_n - a_n c_e)}{a_e b_n - a_n b_e} \quad (61)$$

This formula would only fail if the lines were parallel to each other (in which case the factor of the fraction would be zero). That cannot happen here since they were defined as perpendicular mere few lines of code higher, making the code safe from any computational errors.



**Figure 8:** Projection of a point to a straight line: geometrical illustration of the `computeDistanceVector()` function

The remaining code of the `computeDistanceVector()` function is trivial. The vector between the contact point and the external point is found, and by comparing distances

between the contact point and both points on the edge line, it is determined whether the contact point lies in the middle.

As shown, this function is what determines the normal vector of contact and penetration, and whether a node is aligned with a given element edge at all. As a consequence of those multiple uses, the function is invoked extensively throughout the whole class.

By design, an instance of `ElementEdgeContactSegment` may very well contain a large number of element edges. Before executing any operation, it is necessary to determine which of these edges is most likely to come into contact with a given node. If this were always done by computing a projection on all of them, it could very quickly become a very computationally intensive operation. A system has therefore been put in place to reduce this complexity, utilizing the internal parameters `knownNodes`, `knownClosestEdges` and the private functions `giveClosestEdge()` and `giveIndexOfKnownNode()`. Anytime a node is passed to the contact segment, the `giveClosestEdge()` function is called. If this node has been seen before, its pointer is located in the `knownNodes` array and the corresponding element edge with which the node interacted is located at a corresponding position of the `knownClosestEdges` array. Nothing has to be searched for and the contact segment only considers contact with the previously established element edge. If, however, the `giveIndexOfKnownNode()` fails to identify the node, then the `giveClosestEdge()` function tries to project the node onto all element edges and find the one where the projection is in between the edge nodes. If such an element edge exists, it is both returned as the closest edge to the node and stored as known together with the node. If it does not exist, there is no contact and such information is relayed to whatever part of the contact segment class requested the closest edge. The arrays of known nodes and known closest edges are emptied every time a step convergence is reached, indicated by the `Domain` class calling the `updateYourself()` function on all contact segments. This provides a balance between considering that a large deformation may cause a change in the relative position of a node toward the element edges and the fact that checking this every time is resource-impractical.

After obtaining a closest edge, the `computeExtendedNMatrix()` function queries the element for an  $\mathbf{N}$ -matrix of the given edge. The `StructuralElement::computeEdgeNMatrix()` function is used for this. It requires a point of integration of the base functions. The contact point returned from `computeDistanceVector()` is passed, albeit only after being converted to element local coordinate system using `Element::giveInterpolation()`. The obtained  $\mathbf{N}$ -matrix, therefore, is already integrated. It is now extended by appending a  $2 \times 2$  unit matrix to the end, which serves to accommodate the node DOFs as seen in equation (55). In the case that there is no closest edge to the given node, the function `computeExtendedNMatrix()` returns a zero matrix of the dimension  $2 \times 6$ <sup>11</sup>.

<sup>11</sup>The fixed dimensions in this and other functions of `ElementEdgeContactSegment` are possible because

The function `computeNormal()` behaves in a rather similar fashion. Its goal is to return a normal vector of projection. That is returned regardless of whether the projection on the closest edge is found to be in between the edge nodes<sup>12</sup>. In the case that there is no closest edge for this particular node, a vector of zeros is returned. The normal vector is normalized if it is not zero.

To determine the size of this normal vector, i.e. the distance of projection is the purpose of the `computePenetration()` function. The projection is computed twice: once identically as in the other functions, using the deformed coordinates of all nodes, and once using the initial coordinates. The two computed normal vectors are then compared to determine whether penetration has occurred. If they are found to have opposite directions, meaning that the node is currently on the opposite side of the element edge than it was in the initial configuration, the size of the current distance vector is returned as a negative number. Otherwise, it is returned as a positive number. The test of penetration is performed by the formula

$$\mathbf{n}\mathbf{n}_0 = \cos \phi \leq 0 \tag{62}$$

where

$\mathbf{n}$  is the normal vector in the current deformed configuration

$\mathbf{n}_0$  is the normal vector in the initial configuration

$\phi$  is the angle between the two vectors

The use of the cosine guarantees that the test is robust in regards to most modes of relative deformation of the node and element edge.

Finally, the function `giveLocationArray()` is supposed to return the location array of a segment for the purposes of global matrix assembly in the boundary condition which invokes it. No node is given, and therefore the contact segment returns the location array of the last element edge that it worked with (and an array of zeros if it has not worked with any edge yet). The information on the last element edge worked with is stored in the private class variable `lastEdge`. For obtaining the location array, the functions `Element::giveBoundaryEdgeNodes()` and `Element::giveBoundaryLocationArray()` are used.

#### 3.3.4 Analytical Function Contact Segment

Sometimes it is necessary to simulate contacts of deformable bodies with a rigid unyielding surface. To allow for this in OOFEM, several classes of contact segments were 

---

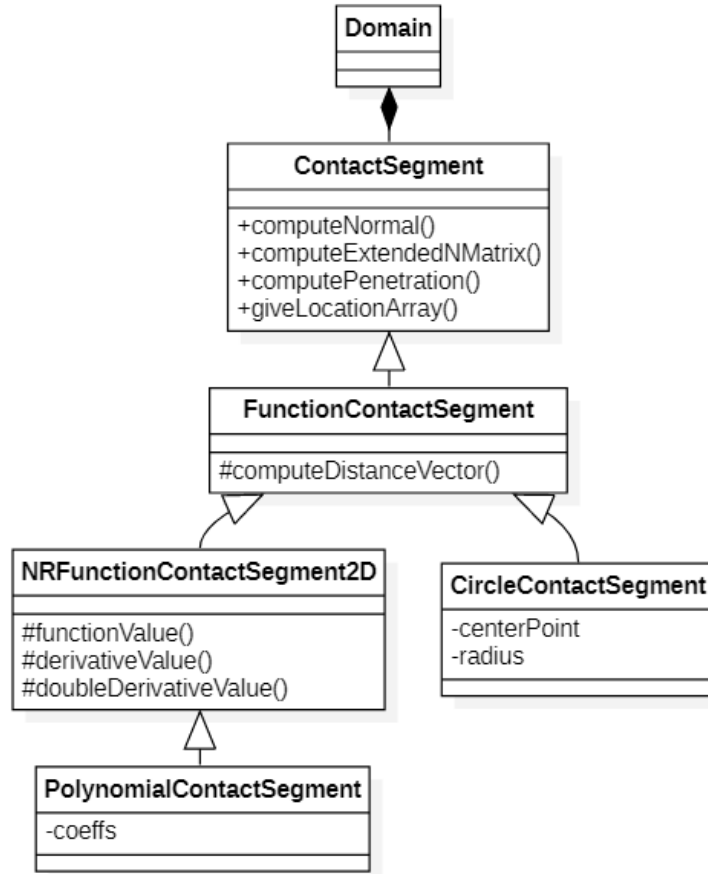
the class is already bound to only be working in two-dimensional space with edges of linear elements thanks to the algorithm of projection.

<sup>12</sup>It has been found that this greatly improves stability for the cases when the external node oscillates around an edge node. The check is still performed by `giveClosestEdge()` every step.



implemented, representing various sorts of rigid contact bodies defined by analytical functions. Most of those are intended for use in two-dimensional space exclusively; some are, however, able to function in three-dimensional space as well.

Those classes are derived from one another in a complex net of inheritance. The whole structure is presented in figure 9.



**Figure 9:** Implementation of analytical function contact segments within the OOFEM environment. `PolynomialContactSegment` and `CircleContactSegment` presented as examples of derived classes

Class `FunctionContactSegment` is an abstract class. It implements all of the functions which are necessary to inherit from the (likewise abstract) `ContactSegment` class, but whenever it needs to compute the projection of a node to the segment (i.e. the analytical function), it invokes the `computeDistanceVector()` function, which is purely virtual (i.e. its implementation is missing and left to the derived classes [Virius, 2018]). It follows that `FunctionContactSegment` cannot be initialized on its own and therefore lacks any input file keywords.

The functions which `FunctionContactSegment` does implement, namely `computeNormal()`, `computeExtendedNMatrix()`, `computePenetration()` and `giveLocationArray()`, are very similar in implementation to their counterparts in

the `ElementEdgeContactSegment` class (as discussed in section 3.3.3). However, the fact that an analytical function is not part of the construction - and therefore has no degrees of freedom - simplifies matters to a large degree.

The `giveLocationArray()` function only returns an empty array, likewise `computeExtendedNMatrix()` only returns the extension (a unit matrix of a dimension corresponding to the number of coordinates of the given node).

The functions `computeNormal()` and `computePenetration()` both provide the virtual function `computeDistanceVector()` with the node coordinates and receive a normal vector of projection in turn. The `computeNormal()` function only normalizes this vector and returns it. The `computePenetration()` function compares it with the distance vector obtained from the initial node position using the procedure described in section 3.3.3 in the form of equation (62). If penetration is occurring, the size of the projection vector is returned as a negative value, otherwise it is returned as a positive value.

The class `FunctionContactSegment` is entirely independent on the dimension of the task domain. Derived classes may be independent as well or lock the domain to 2D or 3D, depending on their approach to implementing `computeDistanceVector()`.

One of these derived classes is the `CircleContactSegment` class. Despite its name, it is actually also dimension independent and may represent a circle in a two-dimensional space as well as a sphere in a three-dimensional space. Except for the initializer needed to retrieve parameters from an input file, it only has the `computeDistanceVector()` function and no others. Listing 7 provides the input file representation of this class. There are only two parameters with self-explanatory names, `centerPoint`, an array of real numbers, and `radius`, a real number. The values are expected to be provided in global coordinates.

```
### Contact Segments
CircleContactSegment 1 centerpoint 2 1. 1. radius 0.5
```

**Listing 7:** An excerpt from an OOFEM input file initializing a circular analytical function contact segment

The computation of the projection is very simple and relies on the fact that the shortest projection of a point on a circle (or a sphere) lies on a direct line to the center of said circle (or sphere). In mathematical terms, the function `computeDistanceVector()` performs the following computation:

$$\mathbf{n} = \frac{\|\mathbf{r}_c - \mathbf{r}_n\| - r}{\|\mathbf{r}_c - \mathbf{r}_n\|}(\mathbf{r}_c - \mathbf{r}_n) \quad (63)$$

where

$\mathbf{n}$  is the projection vector computed

$\mathbf{r}_c$  is the vector of global coordinates of the circle (sphere) center point

$\mathbf{r}_n$  is the vector of global node coordinates

$r$  is the radius of the circle (sphere)

Thanks to this implementation and to the cosine algorithm for checking penetration in `FunctionContactSegment`, the circle (sphere) defined by this contact segment can be positioned both as an external object (i.e. the analyzed structure is outside of the circle/sphere and is not permitted to infiltrate it) or an encompassing boundary (i.e. the analyzed construction is located within the circle/sphere and is not permitted to leave it).

Another class derived from the `FunctionContactSegment` class would be the `NRFunctionContactSegment2D` class. This is an abstract class as well, which cannot be initialized on its own. It implements the `computeDistanceVector()` function, however it in turn creates three new purely virtual functions to be implemented by derived classes - the `functionValue()`, `derivativeValue()` and `doubleDerivativeValue()` functions.

The purpose of `NRFunctionContactSegment2D` is to provide a common framework for many analytical functions in 2D space. Its implementation of the `computeDistanceVector()` function utilizes a Newton-Raphson iterative algorithm to find the closest projection of a point on a  $C^2$  continuous function, which is defined by the child class implementation of the `functionValue()`, `derivativeValue()` and `doubleDerivativeValue()` functions mentioned earlier.

The Newton-Raphson algorithm is based on minimizing the distance function

$$d(x) = \sqrt{(x - x_n)^2 + (f(x) - y_n)^2} \quad (64)$$

where

$d(x)$  is the distance function

$x$  is the global  $x$  coordinate

$[x_n, y_n]$  are the global node coordinates

$f(x)$  is the analytical function

with respect to  $x$ . After omitting the (for purposes of finding the minimum) unnecessary square root and constant members, the simplified distance function  $d_0(x)$  and its derivatives read:

$$d_0(x) = x^2 - 2xx_n + f(x)^2 - 2f(x)y_n \quad (65)$$

$$\frac{dd_0(x)}{dx} = 2x - 2x_n + 2f(x)\frac{df(x)}{dx} - 2y_n\frac{df(x)}{dx} \quad (66)$$

$$\frac{d^2d_0(x)}{dx^2} = 2 + 2f(x)\frac{d^2f(x)}{dx^2} + 2\left(\frac{df(x)}{dx}\right)^2 - 2y_n\frac{d^2f(x)}{dx^2} \quad (67)$$

Those formulas are then used in a very simple Newton-Raphson iterative sequence with the initial condition of  $x = x_n$ . The coordinates  $[x, f(x)]$  obtained after convergence

### 3.3 Implementation of Node-to-Segment Contact

---

are the coordinates of the contact point, which is then used to calculate the distance vector.

The Newton-Raphson method is of course susceptible to fall into local extrema around the starting position and thus fail to reach the proper global minimum. No specific counter against this is implemented here with the assumption that for most reasonable functions in practical application, starting in the  $x_n$  position is enough. It stands to reason that a global minimum of the distance function would rather be closer along the  $x$ -axis from the node than further.

An example of a class which utilizes this Newton-Raphson iteration is the `PolynomialContactSegment` class. It is, as is necessary, derived from `NRFunctionContactSegment2D` and only implements the `functionValue()`, `derivativeValue()` and `doubleDerivativeValue()` functions. An example of an input is included in listing 8.

```
### Contact Segments
PolynomialContactSegment 1 coeffs 3 -1. 1.2 0.65
```

**Listing 8:** An excerpt from an OOFEM input file initializing a polynomial function contact segment

Only an array of coefficients is given, considered to be in descending order of power; those are used to represent a polynomial function. The coefficients in listing 8 would result in a quadratic function in the form

$$f(x) = -x^2 + 1.2x + 0.65 \quad (68)$$

The function `functionValue()`, `derivativeValue()` and `doubleDerivativeValue()` then calculate the function and derivative values as

$$f(x) = k_1x^{n-1} + k_2x^{n-2} + \dots + k_nx^{n-n} \quad (69)$$

$$\frac{df(x)}{dx} = (n-1)k_1x^{n-2} + (n-2)k_2x^{n-3} + \dots + (n-(n-1))k_{n-1}x^{n-n} \quad (70)$$

$$\begin{aligned} \frac{d^2f(x)}{dx^2} = & (n-2)(n-1)k_1x^{n-3} + (n-3)(n-2)k_2x^{n-4} + \dots \\ & + (n-(n-1))(n-(n-2))k_{n-2}x^{n-n} \end{aligned} \quad (71)$$

where

$f(x)$  is the polynomial function

$k_i$  is the  $i$ -th coefficient from the `coeffs` array

$n$  is the number of coefficients in the `coeffs` array

$x$  is the global  $x$  coordinate

With the framework introduced, it is possible to derive many more analytical function contact segments without much effort. For  $C^2$  continuous 2D functions, the only necessity is to implement classes calculating function values and derivatives. With other functions, it is necessary to inherit from `FunctionContactSegment`. However, that also only necessitates an algorithm for finding the projection vector from given coordinates. Everything else is already provided for.

### 3.4 Avenues of Further Development

In its current state described on the previous pages, the OOFEM contact implementation is capable of simulating node-to-node and node-to-segment frictionless contact in two-dimensional space. A choice of either penalty-based or Lagrangian-multiplier-based contact condition is presented in both cases. However, there still remain vast options for further implementations.

The node-to-segment contact conditions, as elaborated on in section 3.3.2, are generally independent on the dimension of the task domain. Implementation of contact segment classes for three-dimensional tasks is the logical next step. As of now, only the circle contact segment is usable to simulate a rigid sphere (see section 3.3.4). Other analytical function segments could be formulated. It is also easy to imagine an element surface contact segment - a class that would treat surfaces of 3D elements in the same way the class `ElementEdgeContactSegment` treats boundaries of 2D elements. In the case of node-to-node contact, the current implementation takes some liberties to ease particular computations of the normal vectors, for which two-dimensional space has to be assumed. It is, however, generally possible to redevelop these algorithms and make the boundary conditions for node-to-node dimension-independent as well.

In the current state, the only element contact segment available is the class `ElementEdgeContactSegment`, which is limited to using linearly formulated elements. The reason for this is mainly the projection algorithm described in section 3.3.3, which assumes a linear edge bounded by only two nodes. An overture has been made in the code to implement a class `QElementEdgeContactSegment`, which would work with quadratic elements. The projection algorithm, and especially its generalization to work for higher-order elements as well, is proving to pose a significant challenge, however<sup>13</sup>.

This all could be also expanded upon by introducing other ways to enforce the non-penetration conditions, e.g. the Nitsche method. More importantly, maybe, an entire new batch of classes could implement various approaches to contact with friction.

---

<sup>13</sup>The algorithms presented in literature, like [Konyukhov and Izi, 2015] or [Yastrebov, 2013], mostly utilize the Newton-Raphson method and thus require second-order derivatives of the element base functions, which is reasonable for quadratic elements, where those are known and constant, but problematic for higher-order elements.

### 3.4 Avenues of Further Development

---

In conclusion, while the current implementation as described and existing is sufficient to be tested and presented in this thesis, it only lays a foundation in the OOFEM code to be extended into numerous related directions in the future.

## 4 Numerical Experiments

In this section, the completed OOFEM implementation of contact conditions is subjected to testing. Various examples are used, validating OOFEM results against results of similar tasks in literature or against known analytical solutions of specific contact cases.

Section 4.1 reviews the examples that were used during code development to test each newly implemented feature. In section 4.2, an analytical solution of a simple task involving node-to-node contact with Lagrangian multipliers is compared with OOFEM results. In section 4.3, the same mesh is reused to recreate an experiment from literature involving size of the penalty parameter. This is expanded upon in section 4.4, where a comprehensive set of experiments is performed to study penalty size in relation to various other parameters of computation. Section 4.5 tests OOFEM contact on a large mesh simulating the rigid flat punch problem, which has a known analytical solution. Finally in section 4.6, the famous Hertz contact problem is recreated in OOFEM and compared with an analytical solution.

For the visualisation of OOFEM results as can be seen throughout this section, the ParaView open-source software tool was used [Ayachit, 2015].

### 4.1 Development Testing

During the course of new code development, it has been necessary to test each new implemented feature. OOFEM input files of simple tasks were developed to easily debug the new code. In this section, we shall review several of them, as they provide easy and simple proofs of functionality of the implemented contact algorithms.

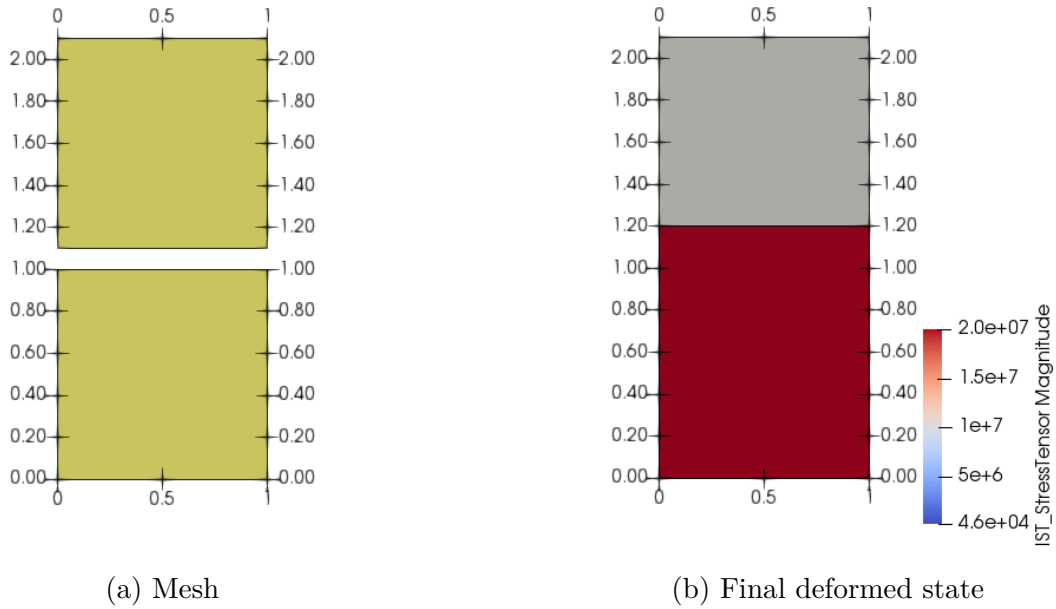
The very first implemented feature was the boundary conditions for node-to-node contact<sup>14</sup>. All the necessary code is encased in the classes `Node2NodePenaltyContact` (for detailed discussion of the implemented code please refer to section 3.2.2) and `Node2NodeLagrangianMultiplierContact` (see section 3.2.3). The boundary conditions contain in their present state some assumptions of two-dimensional domain, and therefore to debug the code, a task of two square elements in a plane stress domain was considered, instead of a comparatively easier one-dimensional case. The elements in the task are positioned about each other. The nodes which form the master-slave pairs for the boundary condition are directly adjacent to each other and their displacement is directed to be in the exact normal direction. This is to respect the fact that node-to-node contact in OOFEM is limited to small deformations.

Two versions of loading were considered. In the first case, loading was driven by displacements. The top edge of the lower element was raised in several steps by a prescribed

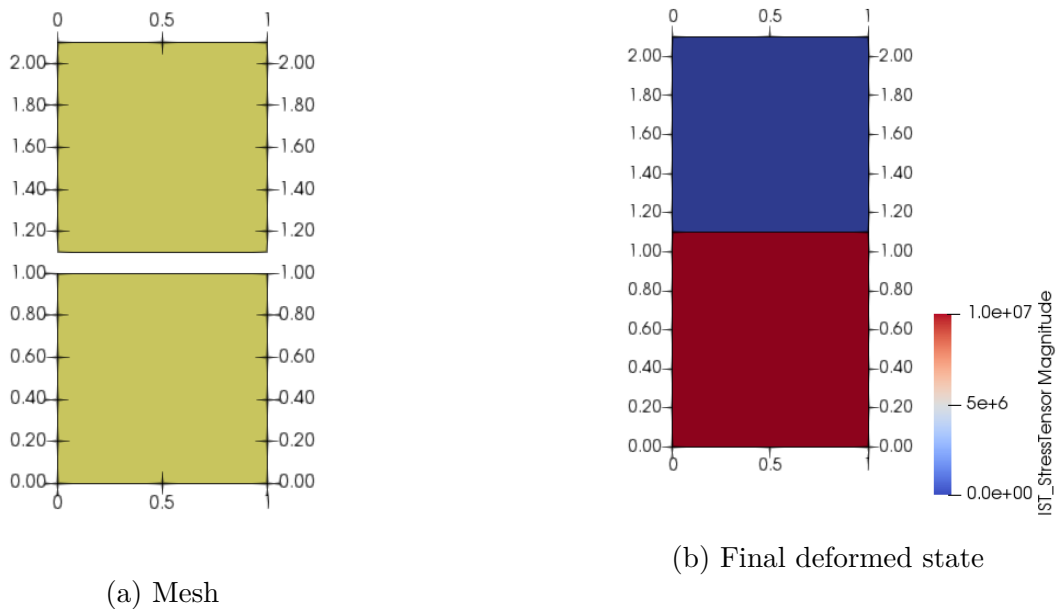
---

<sup>14</sup>The tests were performed separately for both the penalty condition and the Lagrangian multiplier condition. The results presented here in the form of visualisations look the same for both cases, however, so only one figure per test is included

displacement. The expected result is for the upper element to be compressed as the two bodies come into contact. The experiment and its satisfactory conclusion is pictured in figure 10.



**Figure 10:** A displacement-driven development test of node-to-node contact



**Figure 11:** A force-driven development test of node-to-node contact

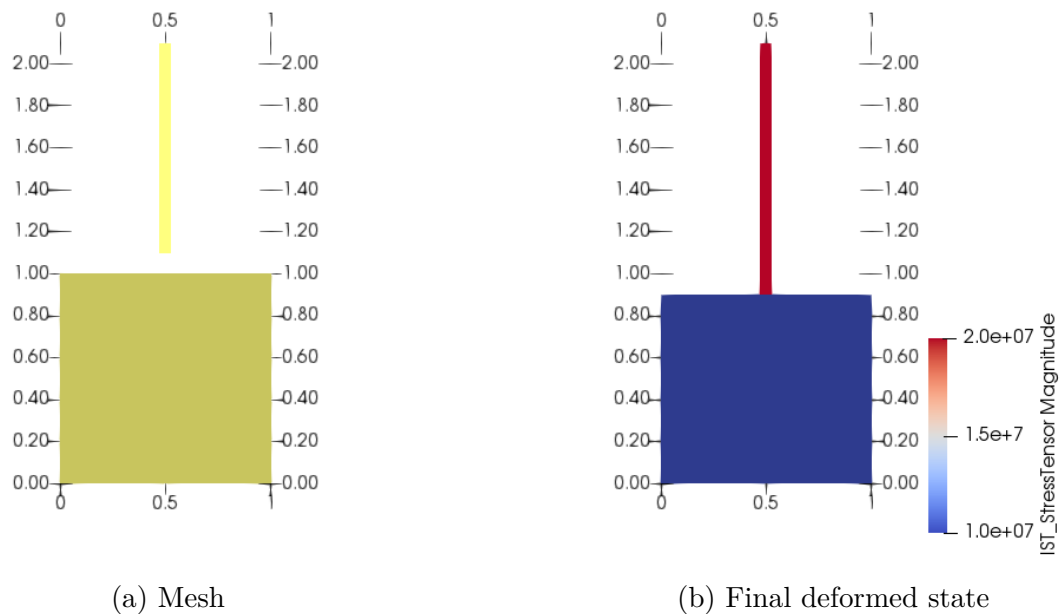
In the second loading case, the lower element was subjected to a force load instead of a prescribed displacement. The upper element, on the other hand, had all nodes fixed in



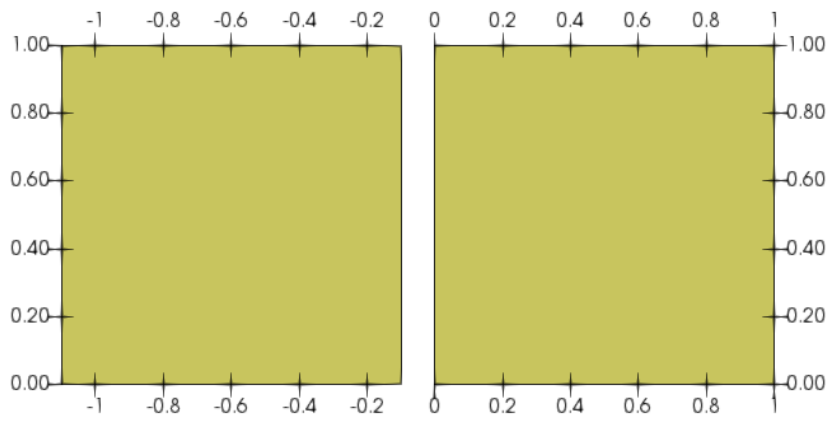
position. The expected result, which can be seen in figure 11, is for the lower element's protraction to be halted by the contact condition and a reaction force to develop along the contact surface.

After the previous experiments ended with positive results, some basic work was undertaken to ensure that those positive results are not limited to this single special case. Figure 13 shows an experiment testing a similar configuration of two elements, only now loaded in the  $x$  direction instead of the original  $y$  direction. Figure 14 demonstrates an experiment which tested the contact conditions on a larger task with 8 elements. Only the central nodes of the lower structure are loaded in this case and accordingly, the response of both structures is more complicated, achieving contact in only some of the specified nodes with differing levels of penetration. Nevertheless, the contact condition is still successfully enforced.

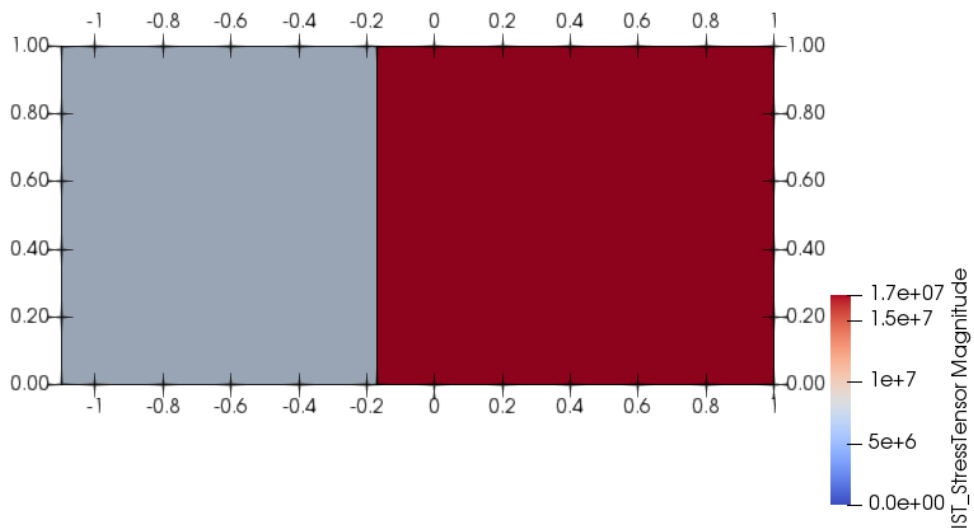
With node-to-node conditions successfully coded and tested, the focus moved on to the development of element contact segments for node-to-segment contact. This is a majorly more complicated coding exercise (details are described in section 3.3.3). Nevertheless, the basic structure of the test task was retained. Now, only, the upper square element has been replaced by a truss, the lower node of which has been positioned against the upper edge of the lower square element. This edge has thus formed the contact segment in contact with said node. A displacement-driven experiment is illustrated in figure 12. This time it is the truss which is subjected to prescribed displacement. In accordance with the aim of the defined contact condition, the square element follows suit.



**Figure 12:** A displacement-driven development test of node-to-segment contact

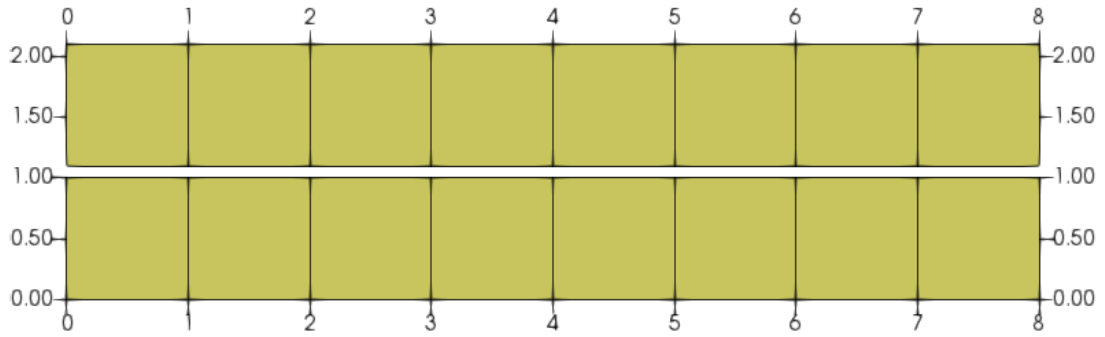


(a) Mesh

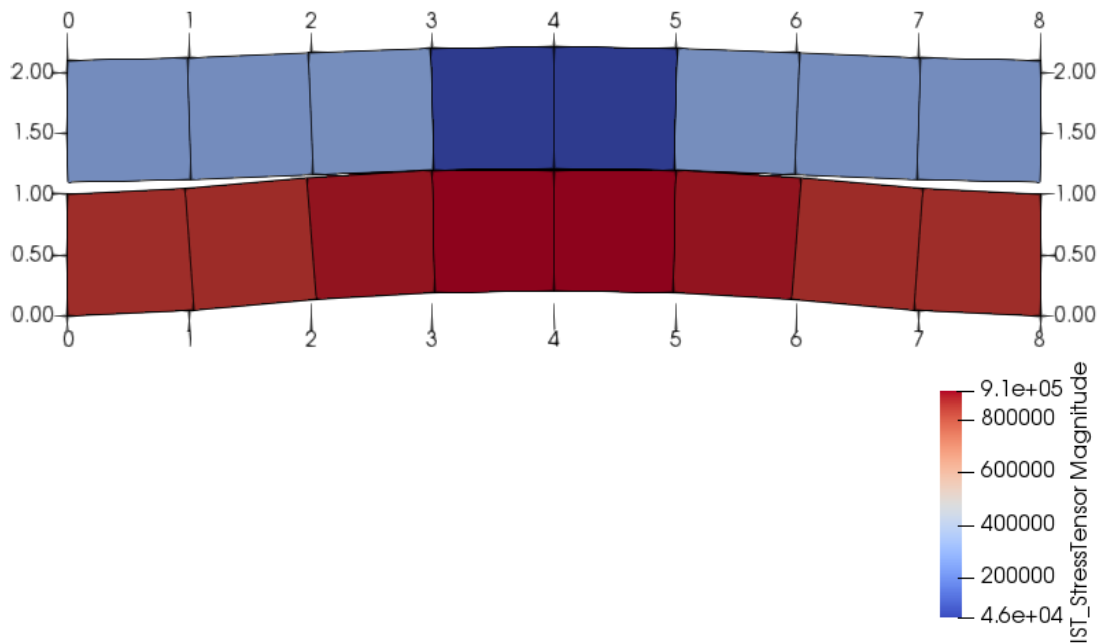


(b) Final deformed state

**Figure 13:** A verification test of node-to-node contact in the X direction



(a) Mesh



(b) Final deformed state

**Figure 14:** A verification test of node-to-node contact with 8 elements

After the element edge contact segment, other contact segments were developed, defined by analytical functions rather than elements. In the test cases, the square element was removed in favor of the analytically defined function. As it is impractical to demonstrate these tests by visualisation, which cannot show the analytical function, table 1 is presented here to show how the displacement of the truss node, despite being loaded by a constant force load in each element, is halted by the contact condition. Data from two experiments are shown, one involving a circular function (OOFEM class `CircleContactSegment`) and the other a quadratic function (OOFEM class `PolynomialContactSegment`). Both functions were positioned so that they intersected with the path of the monitored node at  $y = 1.0$ ,  $w = -0.1$ . In the case of the quadratic function, its vertex was moved slightly to the side so that it did not form the contact point. This was to test the Newton-Raphson iteration on which the `PolynomialContactSegment` class is based.

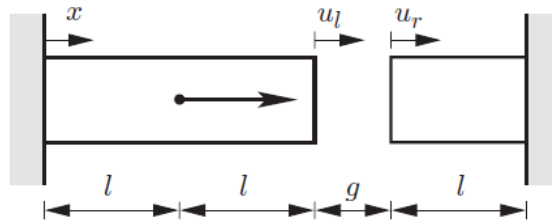
**Table 1:** A verification test of analytical function contact segments: Displacements before and after activation of a penalty contact condition

Step	Displacement $w_c$ (circle)	Displacement $w_p$ (parabola)
1	-3.00000000e-02	-3.00000000e-02
2	-6.00000000e-02	-6.00000000e-02
3	-9.00000000e-02	-9.00000000e-02
4	-1.00198020e-01	-1.00205864e-01
5	-1.00298990e-01	-1.00310919e-01

This overview of development tests is obviously not decisive proof that the contact conditions work as expected, if for no other reason, then because there is no hard, verifiable numerical data. The experiments that shall provide those are described further in this thesis. The aim of those simple examples was only to prove at a glance that the contact conditions are not simply ignored by OOFEM, and above all to provide something for OOFEM to compute which activates the newly written code, allowing for its debugging.

## 4.2 Contact of Two Bars

This is a rather simple test of contact functionality. An example of two bars in one-dimensional space is considered. An initial gap is located between them, as denoted in figure 15. The test is inspired by the example discussed in [Wriggers, 2006], where it is presented as proof of nonlinearity introduced by contact.



**Figure 15:** Contact of two bars: Initial geometry, taken from [Wriggers, 2006]

The bars are both made of a linear elastic material. The right bar is discretized by one element only; the left bar consists of two elements. All elements are of the size  $l$  and cross-section area  $A$ . In the shared node between the two left elements, a nodal load of size  $F$  is applied in the positive  $x$  direction. A node-to-node Lagrangian multiplier contact boundary condition is introduced, binding the facing nodes of the two bars, which are distanced from each other by the initial gap  $g_0$ . The relevant section of the OOFEM input file for the task is included in listing 9. The OOFEM mesh is presented in figure 16. Displacement values shall be further referred to as  $u_1$ ,  $u_2$  and  $u_3$ , for the non-fixed nodes 2, 3 and 4, respectively.

```

ndofman 5 nelem 3 ncrosssect 1 nmat 1 nbc 4 nic 0 nltf 2 nset 3
### Nodes
# Nodes of Element 1 and 2 (left bar)
node 1 coords 2 0.0 0.0
node 2 coords 2 1.0 0.0
node 3 coords 2 2.0 0.0
# Nodes of Element 3 (right bar)
node 4 coords 2 2.1 0.0
node 5 coords 2 3.1 0.0
### Elements
truss2d 1 nodes 2 1 2 crossSect 1 mat 1 cs 1
truss2d 2 nodes 2 2 3 crossSect 1 mat 1 cs 1
truss2d 3 nodes 2 4 5 crossSect 1 mat 1 cs 1
### CrossSection
SimpleCS 1 area 1
### Materials
isoLE 1 d 0. E 2e6 n 0.0 talpha 0.
### Boundary Conditions
BoundaryCondition 1 loadTimeFunction 1 values 2 0.0 0.0 dofs 2 1 2 set 1
BoundaryCondition 2 loadTimeFunction 1 values 1 0.0 dofs 1 2 set 3
NodalLoad 3 loadTimeFunction 1 components 1 1.e5 dofs 1 1 set 2
n2nlagrangianmultipliercontact 4 loadTimeFunction 1 masterset 1 3
  slaveset 1 4 usetangent

```

**Listing 9:** Contact of two bars: Node, element, material and boundary condition definitions from the OOFEM input file



**Figure 16:** Contact of two bars: OOFEM mesh

The input values of the parameters  $F$ ,  $l$ ,  $A$ ,  $E$  and  $g_0$  used for the experiment are summarized in table 2. The force  $F$  was applied in four steps, the value shown is the value of one increment only. In total, four times that value was applied at the end of the task.

**Table 2:** Contact of two bars: Input values

Parameter	Value
$F$	100 kN
$l$	1 m
$A$	1 m <sup>2</sup>
$E$	2000 kPa
$g_0$	0.1 m

With the use of these input values, an analytical solution of the task can be constructed. Disregarding the FEM formulation, the first two steps can be easily computed by traditional principles of mechanics. As indicated by the results, contact does not apply in these steps and therefore the solution is valid.

As shown in figure 15, the force  $F$  applies in the middle of the left bar. Therefore, only the stiffness of the leftmost element is resisting it. The second element is neither loaded nor subject to any deformation. It follows that  $u_1 = u_2$ . The value of those displacements in the first step can be determined as

$${}^{(1)}u_1 = {}^{(1)}u_2 = \frac{{}^{(1)}Fl}{EA} = \frac{100 \cdot 1}{2000 \cdot 1} = 0.05 \text{ m} \quad (72)$$

The gap has not been closed. For that, the second step is necessary, where the total value of force applied doubles:

$${}^{(2)}u_1 = {}^{(2)}u_2 = \frac{{}^{(2)}Fl}{EA} = \frac{200 \cdot 1}{2000 \cdot 1} = 0.1 \text{ m} \quad (73)$$

The two bars are now in contact (though penetration has not yet occurred and therefore no contact boundary condition has been activated). For any further steps, the pre-

ceding simple solution is not possible. It is now necessary to employ the FEM formulation with Lagrangian multiplier contact condition.

It has been discussed and determined in [Wriggers, 2006] that the FEM formulation of this particular contact problem takes the shape of the following system of equations:

$$\begin{bmatrix} 2\frac{EA}{l} & -\frac{EA}{l} & 0 & 0 \\ -\frac{EA}{l} & \frac{EA}{l} & 0 & -1 \\ 0 & 0 & \frac{EA}{l} & 1 \\ 0 & -1 & 1 & 0 \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \lambda \end{pmatrix} = \begin{pmatrix} F \\ 0 \\ 0 \\ -g_0 \end{pmatrix} \quad (74)$$

where

$\lambda$  is the Lagrangian multiplier of the contact condition

Solving this equation system leads to a direct solution for  $u_2$  [Wriggers, 2006]:

$${}^{(i)}u_2 = \frac{1}{3} \left( 2g_0 + \frac{{}^{(i)}Fl}{EA} \right) \quad (75)$$

where

$i$  is the index of incremental loading step, assuming the values 3 or 4

Using formulas (72) and (75), it is now possible to determine analytically the values of displacement  $u_2$  in all solution steps. Those are compared with the solution provided by OOFEM calculation in table 3. The deformed state of the OOFEM mesh, showing also magnitudes of displacements, can be seen in figure 17. It can be concluded that the test was satisfactory. The OOFEM calculation has in this simple case returned exactly the results predicted by the analytical solution.

**Table 3:** Contact of two bars: Comparison of Analytical and OOFEM results

Step	Analytical $u_2$	OOFEM $u_2$
1	0.0500	0.0500
2	0.1000	0.1000
3	0.1167	0.1167
4	0.1333	0.1333

### 4.3 Two Bars with Penalty Condition

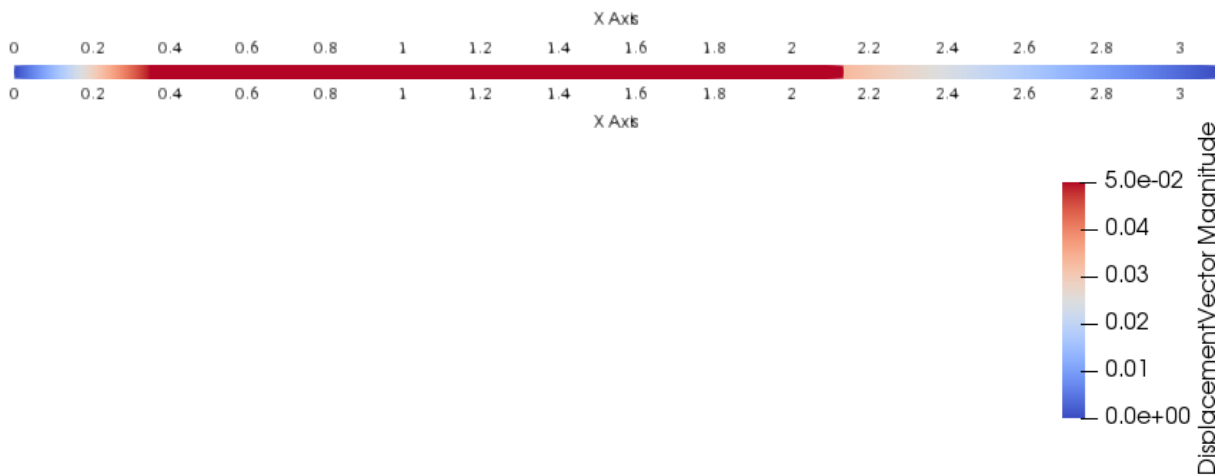


Figure 17: Contact of two bars: OOFEM deformed state

### 4.3 Two Bars with Penalty Condition

The OOFEM mesh used in the previous section 4.2 was largely reused to simulate another example from literature. This time it is a contact problem described in [Konyukhov and Izi, 2015] and solved there by means of the penalty parameter.

Similarly to the example in [Wriggers, 2006] which was used for the test 4.2, two bars are considered in two-dimensional space, one of them consisting of two truss elements and the other one from only one. The loading force remains in the middle of the larger bar as well. The mesh has slightly different dimensions; its altered initial state is pictured in figure 18.

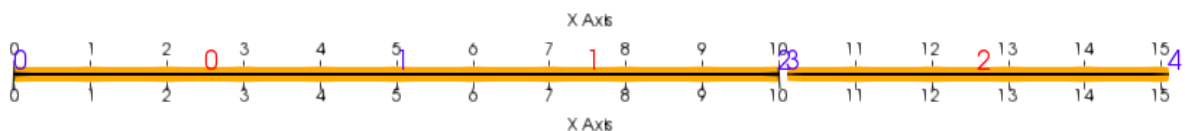


Figure 18: Two bars with penalty condition: OOFEM mesh

The length  $l$  of the finite elements is newly 5 m and also other material and computational parameters have changed. The summary is given in table 4. Compare directly with table 2.



**Table 4:** Two bars with penalty condition: Input values

Parameter	Value
$F$	50 N
$l$	5 m
$A$	1 m <sup>2</sup>
$E$	100 Pa
$g_0$	0.1 m

The analysis, also, now proceeds in 5 steps rather than 4. An attempt to compute the deformation in the first step by formula (72) proves that penetration shall now occur in the very first step:

$${}^{(1)}u_1 = {}^{(1)}u_2 = \frac{{}^{(1)}Fl}{EA} = \frac{50 \cdot 5}{1000 \cdot 1} = 0.25 \text{ m} > g_0 = 0.1 \text{ m} \quad (76)$$

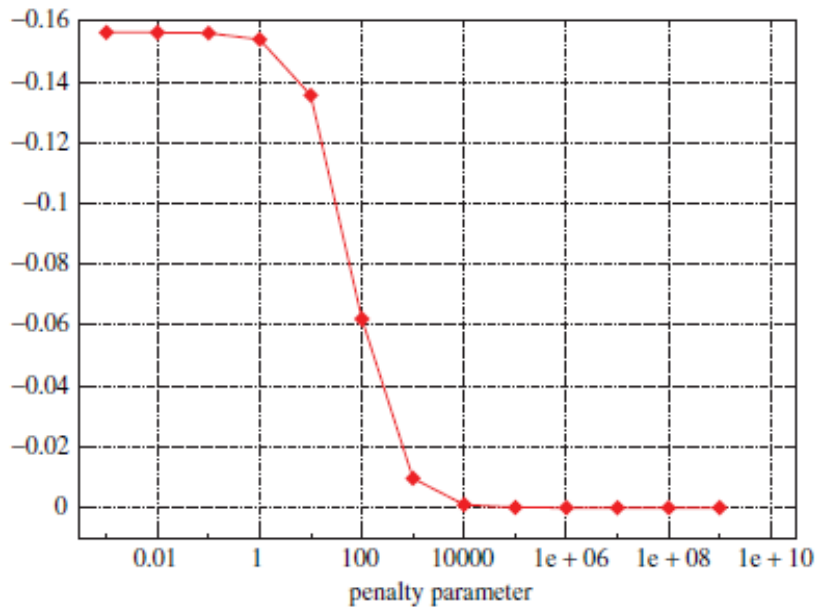
In [Konyukhov and Izi, 2015], this example is used to illustrate the effect of penalty size on the result, namely on the residual penetration of element 2 into element 3. The same experiment was performed in OOFEM as well. The results of [Konyukhov and Izi, 2015] are pictured in figure 19; the results from OOFEM are similarly plotted in figure 20. The plotted values are values of penetration in the last computed step. Negative values mean penetration, i.e. node 3 having a higher  $x$  coordinate than node 4.

Note that the plot from [Konyukhov and Izi, 2015] displays the result of the first step of the computation, which explains the differences in penetration values among the two plots.

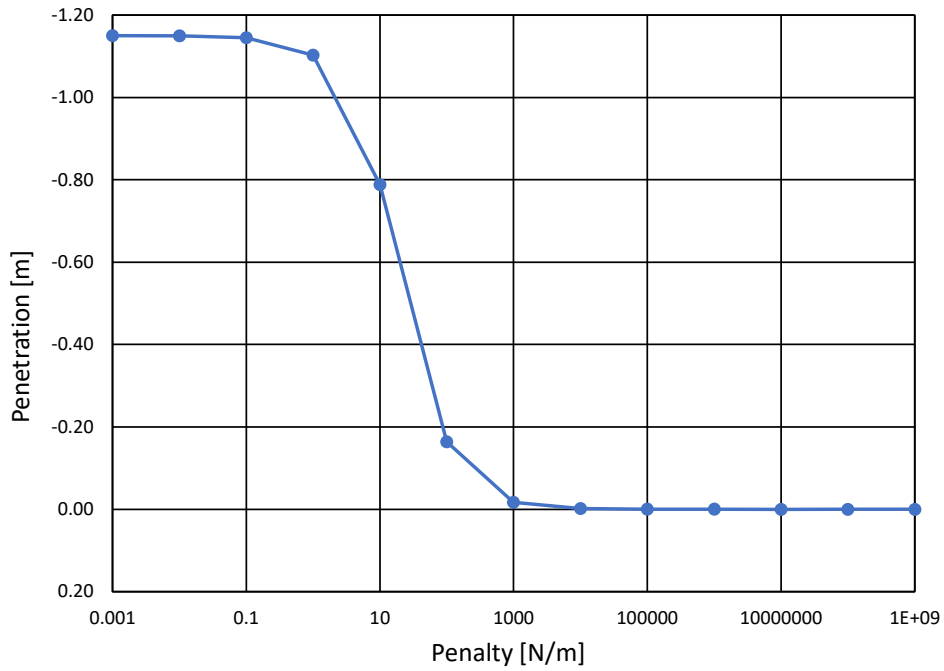
Visual comparison of the two figures serves as proof that the OOFEM calculation performs as expected. For very low penalty values, the penetration approaches values from a no-contact case; for penalty values in the order of  $E$  and higher, the penetration disappears almost completely (approaching a perfect contact case) [Konyukhov and Izi, 2015].

### 4.3 Two Bars with Penalty Condition

---



**Figure 19:** Two bars with penalty condition: Results from literature [Konyukhov and Izi, 2015]



**Figure 20:** Two bars with penalty condition: Results from OOFEM

## 4.4 Penalty Size Study

In the penalty-based approach to contact computation, the size of the penalty parameter is a question of high importance. In the OOFEM implementation, the penalty is an external input decided by the user for each task independently. In this section, it shall be examined empirically how the size of the penalty chosen reflects on the error of the computation result.

The penalty can be understood as a measure of the contact force when related to penetration. This force, when the bodies are interfering with each other, shall be infinite. From that follows the conclusion that an ideal solution of a contact problem arises when the penalty parameter is itself infinite [Konyukhov and Izi, 2015]. Infinite penalty parameters are naturally impossible in the practical implementation of FEM; very large penalty parameters also pose many problems related to the shape of the resulting systems of equations [Zienkiewicz and Taylor, 2000]. Therefore a classical problem presents itself - the demands for ease of solution and for its precision are contradictory to each other.

In this simple test, an uncomplicated example of two single element bodies in two-dimensional space is used. Two identical square-shaped elements are positioned above each other, separated by an initial gap of a 0.1 fraction of their height. The top nodes of the bottom element and bottom nodes of the top element align with each other and are defined as master and slave nodes, respectively, for a node-to-node contact boundary condition. The geometry of this setup, as well as the node indexing henceforth referenced in this text, is presented in figure 21<sup>15</sup>.

All nodes except nodes 3 and 4 are fixed in position. Nodes 3 and 4 are fixed in the direction of the X axis; in the direction of the Y axis, a force load of size  $F$  is applied in three identical steps. Both elements are declared to be of the same isotropic linear elastic material, defined by Young modulus  $E$  and Poisson ratio  $\nu = 0$ .

Numerous experimental computations were conducted with alternating values of the parameters  $F$  and  $E$ , as well as the penalty parameter  $p$ . Each resulted in a different error  $\varepsilon$ , manifesting as penetration at the point of contact in the last computational step. For obvious reasons, only such combinations of  $F$  and  $E$  which resulted in achieving contact within the three steps computed were considered.

The results of these experiments are summarized in Table 5.

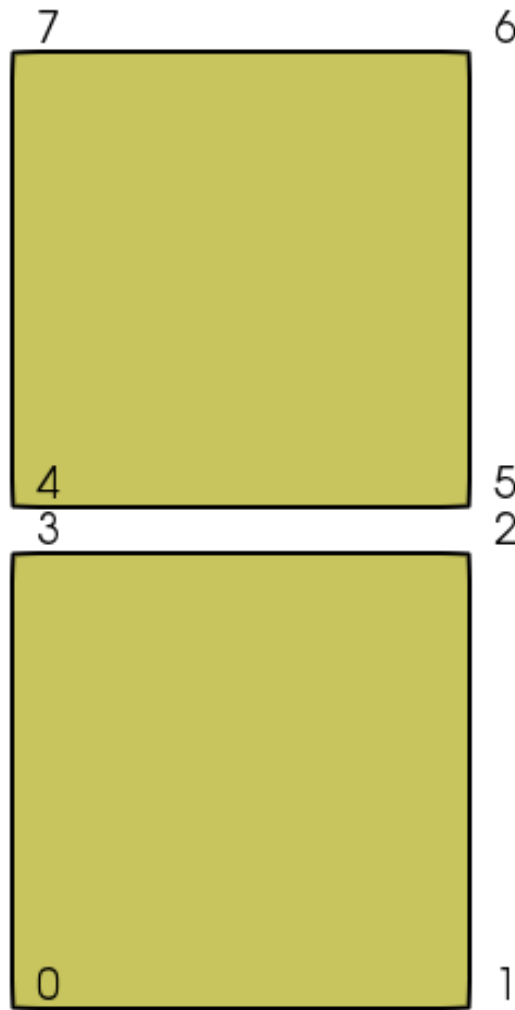
The experiments numbered 1-6 show the relation between penalty size  $p$  and error  $\varepsilon$  under the circumstances of all other variables remaining constant. Disregarding the apparent rounding errors affecting the error computation, it is confidently demonstrated that this is an inverse linear relation. With increasing penalty, error diminishes proportionately.

---

<sup>15</sup>The node ids are zero-based in the figure and therefore lower by one than as defined and mentioned further in the text

**Table 5:** Penalty size study: Experiment data

$i$	$E$	$F$	$p$	$\varepsilon$
[-]	[Pa]	[N]	[N/m]	[-]
1	1.00E+08	3.00E+06	1.00E+10	0.000299995
2	1.00E+08	3.00E+06	2.00E+10	0.000149999
3	1.00E+08	3.00E+06	5.00E+10	0.000060000
4	1.00E+08	3.00E+06	1.00E+11	0.000030000
5	1.00E+08	3.00E+06	1.00E+12	0.000003000
6	1.00E+08	3.00E+06	1.00E+13	0.000000300
7	1.00E+08	3.00E+06	1.00E+10	0.000299995
8	1.00E+08	4.00E+06	1.00E+10	0.000399997
9	1.00E+08	8.00E+06	1.00E+10	0.000799988
10	1.00E+08	1.00E+07	1.00E+10	0.000999988
11	1.00E+08	3.00E+07	1.00E+10	0.002999987
12	1.00E+08	3.00E+08	1.00E+10	0.029999984
13	1.00E+08	3.00E+10	1.00E+10	2.999999620
14	2.00E+08	3.00E+06	1.00E+10	0.000198020
15	1.00E+08	3.00E+06	1.00E+10	0.000299995
16	5.00E+07	3.00E+06	1.00E+10	0.000299998
17	1.00E+07	3.00E+06	1.00E+10	0.000300000
18	1.00E+06	3.00E+06	1.00E+10	0.000300000
19	1.00E+08	3.00E+06	1.00E+10	0.000299995
20	1.00E+08	6.00E+06	2.00E+10	0.000299998
21	1.00E+08	3.00E+07	1.00E+11	0.000300000
22	1.00E+08	1.50E+06	5.00E+09	0.000198020



**Figure 21:** Penalty size study: Initial geometry of the task used

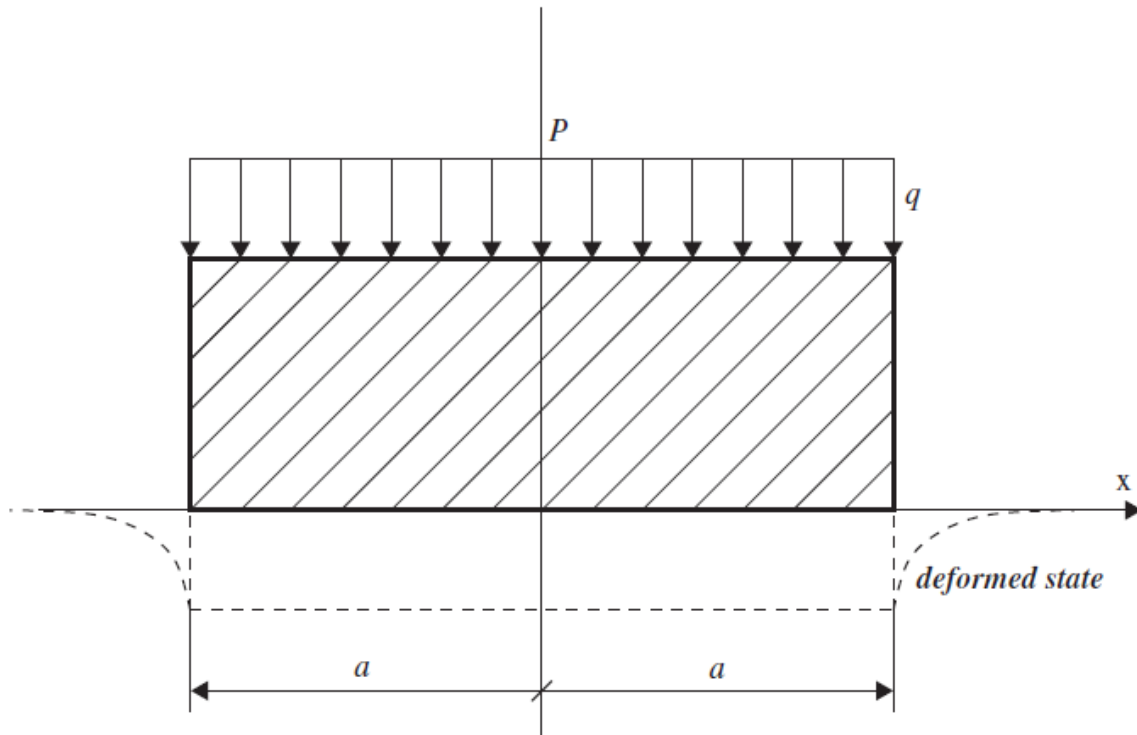
The further set of experiments, numbered 7-13, attempts a similar comparison with the applied nodal load  $F$ . Here the relation is also linear, albeit directly proportional. As the magnitude of the load nears the magnitude of the penalty, the error increases to very significant values. It is apparent that for the computation to proceed with a bearable degree of precision, the size of the penalty parameter shall be kept several orders of magnitude greater than the maximum size of the loads applied. This is further reinforced by experiments 19-21, where  $F$  and  $p$  change proportionately to each other, keeping the error constant.

Experiments 14-18 inspect the relationship between the Young modulus of the material  $E$  and the error. The change in the Young modulus implies a change in the size of most regular members of the model's stiffness matrix. The phenomenon investigated is therefore the influence of the irregularities in this matrix introduced by the penalty member.

## 4.5 Rigid Flat Punch Problem

The "rigid flat punch" problem (also known as "stamping problem") is a benchmark contact problem which can be verified by comparing the results to an existing analytical solution. The formulation of the problem for the purposes of the following experiments has been taken from [Konyukhov and Izi, 2015].

The problem is formulated in two-dimensional plane strain space. As indicated by the problem name, a rigid flat body (a "stamp") of a width  $2a$  with sharp corners is considered. Below this body, an elastic solid half-space is positioned, its boundary aligned with the bottom of the rigid body. The top of the rigid body is pressed into the elastic half-space by means of loading its top boundary with a uniformly distributed force load of size  $q$ . The whole assembly is pictured in figure 22.



**Figure 22:** Rigid flat punch problem: Sketch of situation, taken from [Konyukhov and Izi, 2015]

In the described case, the stress distribution along the contact surface can be determined for the case of frictionless contact as [Konyukhov and Izi, 2015]:

$$p(x) = \frac{P}{\pi\sqrt{a^2 - x^2}} \quad (77)$$

where

$p(x)$  is the stress distribution function

$P$  is the total force applied, equivalent to  $2aq$

$a$  is half of the rigid body width

$x$  is the  $x$  coordinate along the contact surface such that its origin lies in the middle of said surface

The formula describes a symmetrical curve, which at the edge of the rigid body ( $|x| = a$ ) approaches infinity. This is consistent with the shape of deformation depicted in figure 22, where there is a sharp corner in the elastic material, implying a stress singularity. This is, however, a state unattainable by a FEM computation for several reasons, e.g. inability to mesh the corner area with infinitesimal elements and the simple fact that infinite stress is principally incompatible with computer processing. Various attempts at meshing the critical areas and defining the contact conditions may produce results exhibiting a different degree of correlation with the analytical solution. Several such attempts have been the topic of this experiment.

The FEM model for the experiment has been defined using square plane stress elements and the node-to-segment penalty condition. It shall be noted that especially in the case of larger meshes, the solver was having trouble with large penalty parameters. Therefore the penalty parameter used is quite small, which results in some (mostly invisible) penetration along the contact surface. As this penetration is uniform, it should only mean a loss of a uniform value of contact force along the contact surface and therefore not have any effect on the shape of the pressure distribution.

Various different meshes were considered. For the meshing, a simple script in the MATLAB software [MATLAB, 2017] was used.

In all meshes, the rigid body is modelled by a single row of square elements. Below it, the elastic "half-space" is represented by a significantly larger field of square elements. In all cases, the elastic half-space is fixed along its bottom edge. In some meshes, it is also fixed along its sides. A linear elastic material is prescribed. The rigid body is fixed against movement in the  $x$  direction.

Loading occurs by means of prescribed displacement assigned to the lower nodes of the rigid body, i.e. the nodes along the contact surface. This has been chosen as the easiest way to model the rigidity of the stamp. In such configuration, naturally, there is no force  $P$  present to be used in the calculation of the analytical solution. The value of the force is obtained post-fact as a sum of all reactions in the  $y$  direction in the contacting nodes. From these reactions the pressure curve is constructed as well.

The parameters of the task common for all mesh representations are summarized in table 6. The individual meshes and differences among them are listed in table 7.

**Table 6:** Rigid flat punch problem: Task parameters common for all mesh configurations

Description	Parameter	Value
Stiffness modulus of the elastic halfspace	$E$	1 kPa
Half-length of the contact surface	$a$	10 m
Depth of the elastic half-space model	$h$	6 m
Prescribed depression of stamp into half-space	$w$	0.395 m
Penalty parameter	$p$	1 kN/m

**Table 7:** Rigid flat punch problem: Overview of mesh configurations

Mesh	Elem's of stamp	Elem's of half-space	Steps	Overhang	Segments in	Sides
20-198F	20	198	20	6.5 m	half-space	free
20-198FX	20	198	20	6.5 m	half-space	fixed
20-198RX	20	198	20	6.5 m	stamp	fixed
20-804FX	20	804	20	6.75 m	half-space	fixed
40-804FX	40	804	20	6.75 m	half-space	fixed
40-3192FX	40	3192	20	6.625 m	half-space	fixed
40-3192RX	40	3192	20	6.625 m	stamp	fixed



The *Elements of stamp* and *Elements of half-space* columns in table 7 refer to the number of elements the particular mesh devotes to the rigid body and the elastic half-space, respectively. The *Steps* column describes the number of solution steps in which the solution was computed<sup>16</sup>.

Different meshing sometimes resulted in small differences in the size of the elastic half-space model, which is reflected in the *Overhang* column. The value represents the distance by which the half-space model is wider than the rigid body on each side, which can have some (later proven to be negligible) influence on the precision of the stress distribution in the elastic model.

Two versions of the contact condition were considered. Either the nodes of the rigid body were paired with element edges on the corresponding boundary of the elastic half-space, indicated in the table by stating "half-space" in the *Segments in* column, or vice-versa, nodes of the boundary were paired with element edges on the rigid body, indicated by "stamp" in the same column.

Finally, the sides of the elastic half-space model were either fixed in place or free; similarly as the size of the overhang, this had a negligible influence, yet is stated for the sake of completeness.

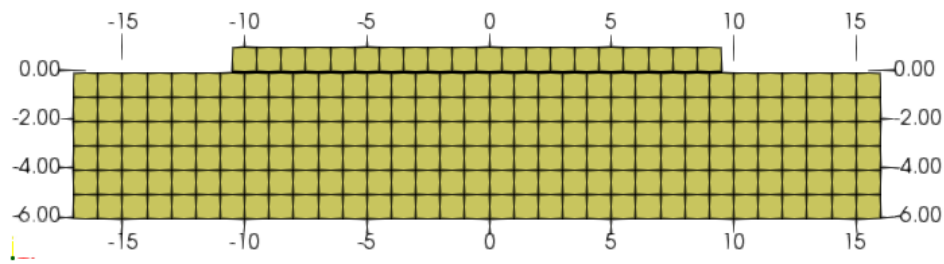
On all the described meshes, computation of the described task was performed. As expected, the achieved results differ significantly from each other. In figures 23 through 29, one mesh is displayed in each, showing the initial state, deformed state, stress distribution in elements and a graph comparing the achieved stresses on a contact surface with the analytical solution. The task is symmetrical, therefore only a half of the contact surface is plotted. Since, as already described, the force used to calculate the analytical solution is itself obtained from the analysis results, the analytical solutions slightly differ as well. An overview of this difference is presented in table 8. A plot of all solutions compared with an average analytical solution is then included in figure 30.

**Table 8:** Rigid flat punch problem: Computed loading forces for each mesh configuration

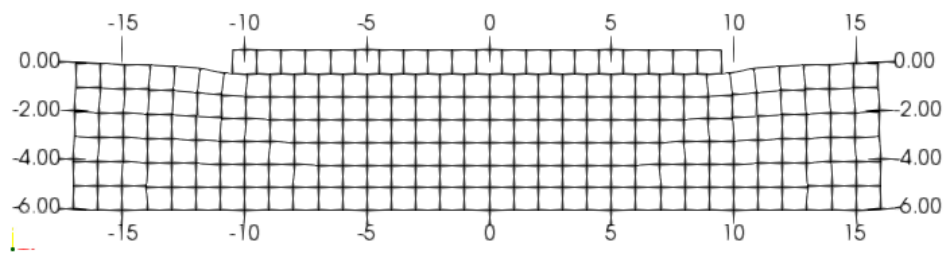
Mesh	Force $P$	Mesh	Force $P$
20-198F	1.541 kN	40-804FX	1.511 kN
20-198FX	1.543 kN	40-3192FX	1.479 kN
20-198RX	1.415 kN	40-3192RX	1.453 kN
20-804FX	1.505 kN	<b>Average</b>	<b>1.491 kN</b>

<sup>16</sup>The total prescribed displacement remains the same for all meshes.

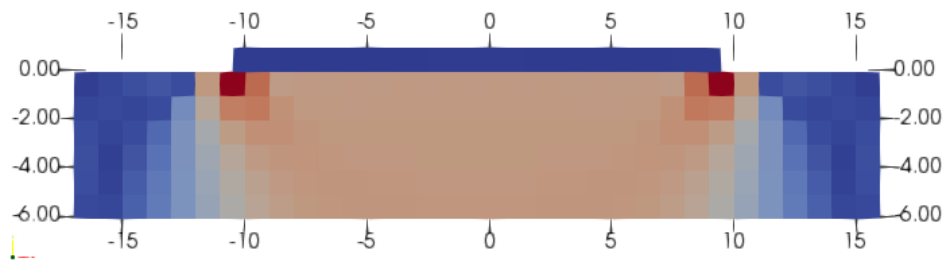
## 4.5 Rigid Flat Punch Problem



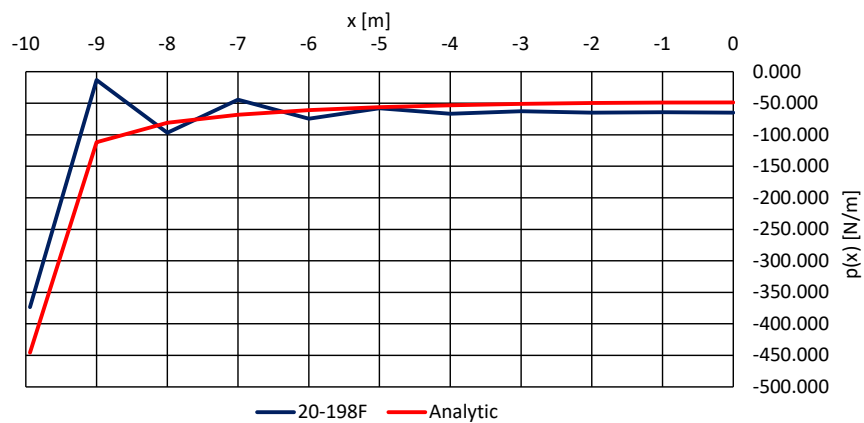
(a) Initial state



(b) Deformed state

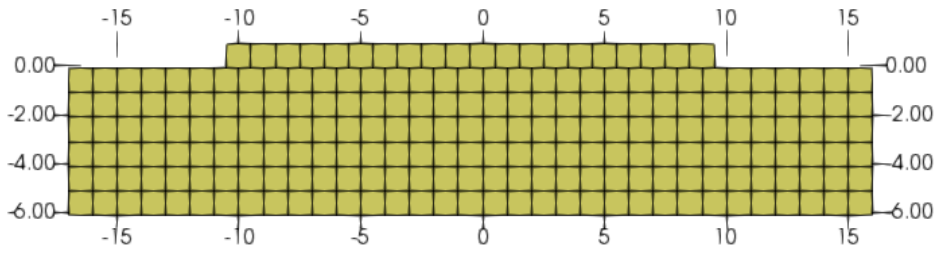


(c) Magnitude of stress

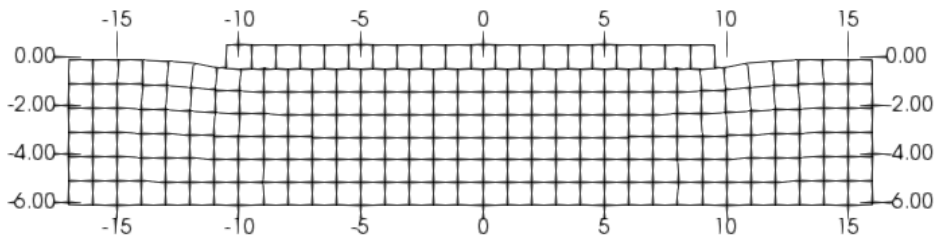


(d) Stress distribution along the contact surface

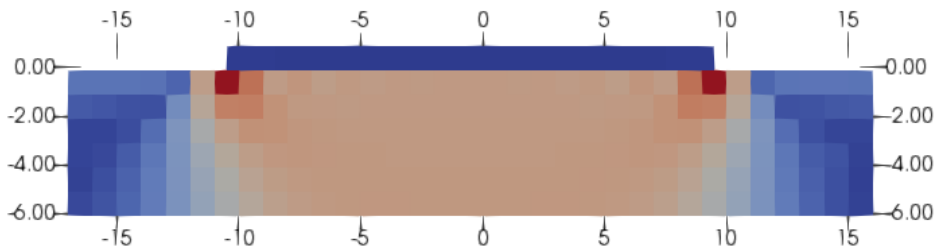
**Figure 23:** Rigid flat punch problem: Mesh 20-198F



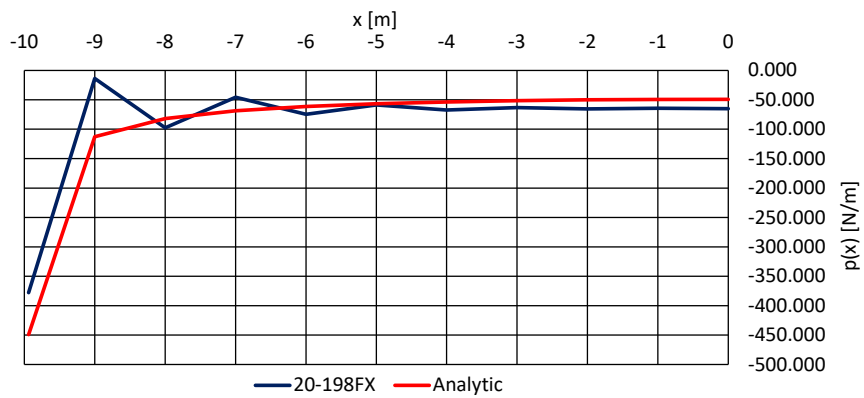
(a) Initial state



(b) Deformed state



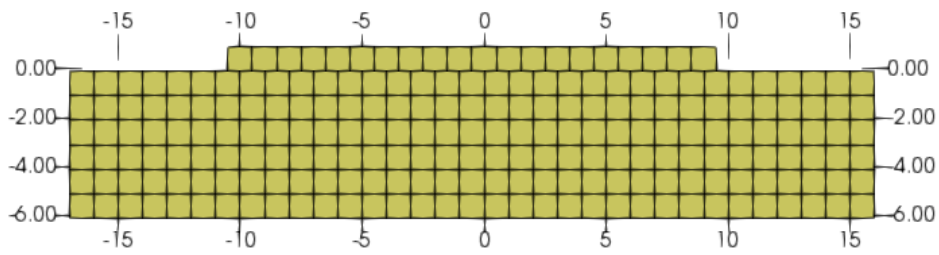
(c) Magnitude of stress



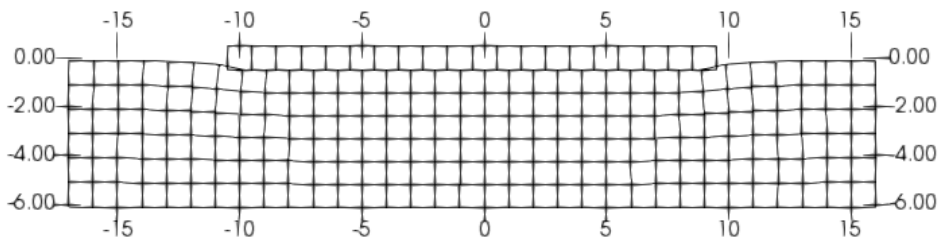
(d) Stress distribution along the contact surface

**Figure 24:** Rigid flat punch problem: Mesh 20-198FX

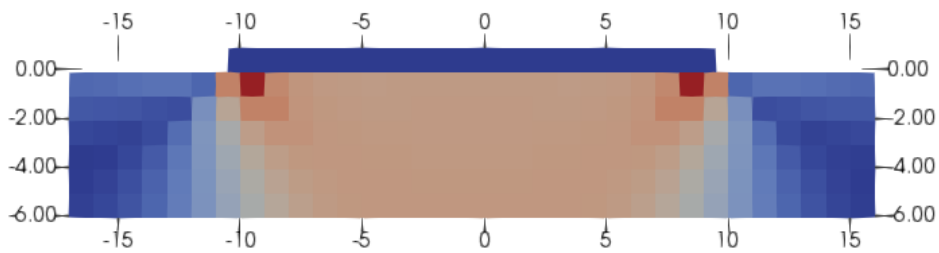
## 4.5 Rigid Flat Punch Problem



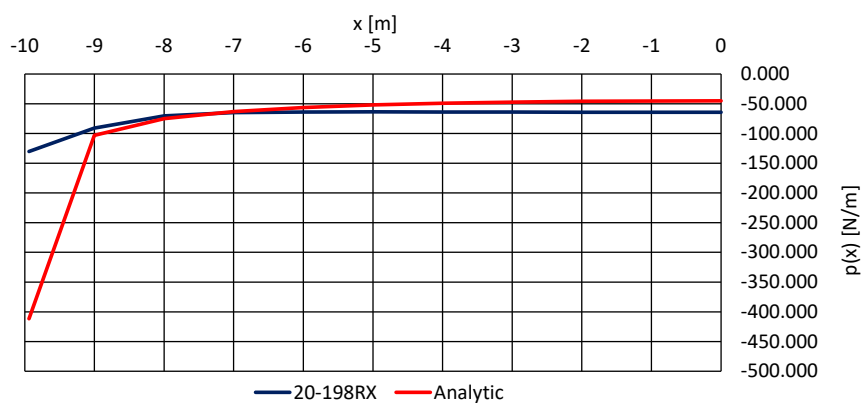
(a) Initial state



(b) Deformed state

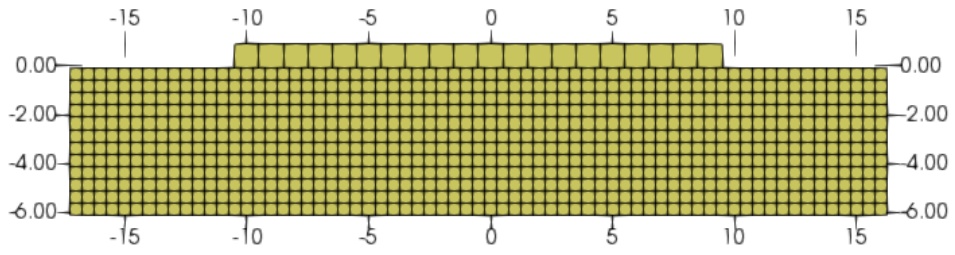


(c) Magnitude of stress

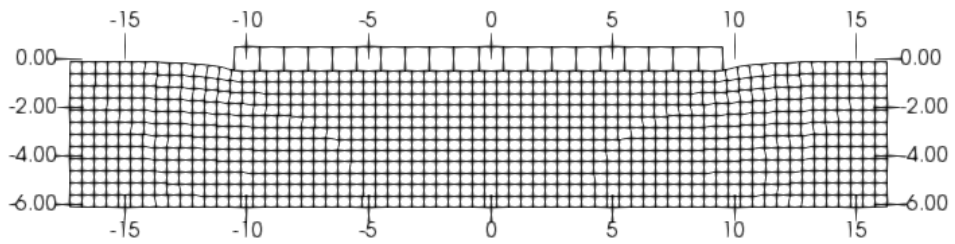


(d) Stress distribution along the contact surface

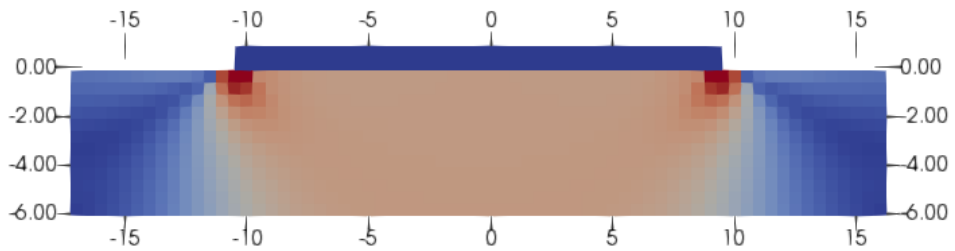
**Figure 25:** Rigid flat punch problem: Mesh 20-198RX



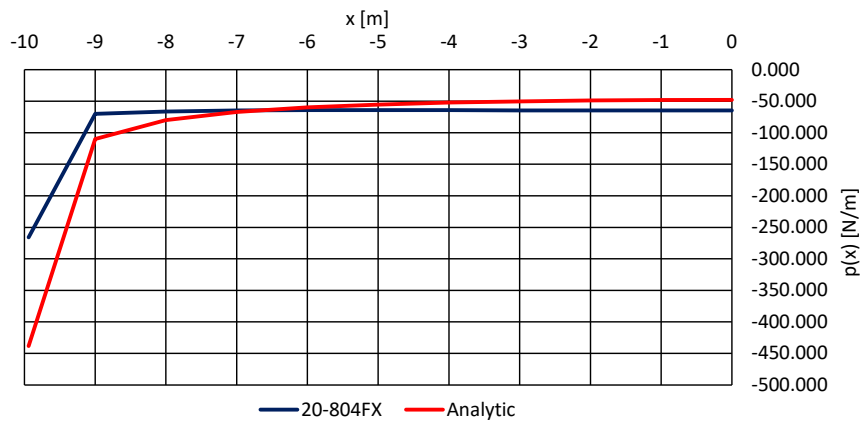
(a) Initial state



(b) Deformed state



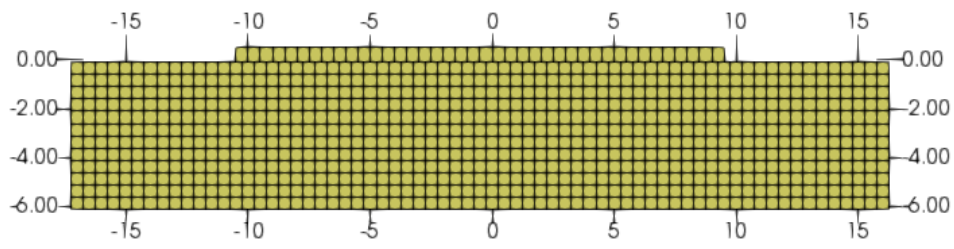
(c) Magnitude of stress



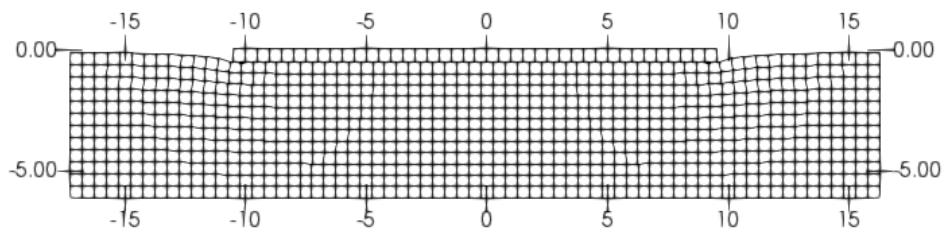
(d) Stress distribution along the contact surface

Figure 26: Rigid flat punch problem: Mesh 20-804FX

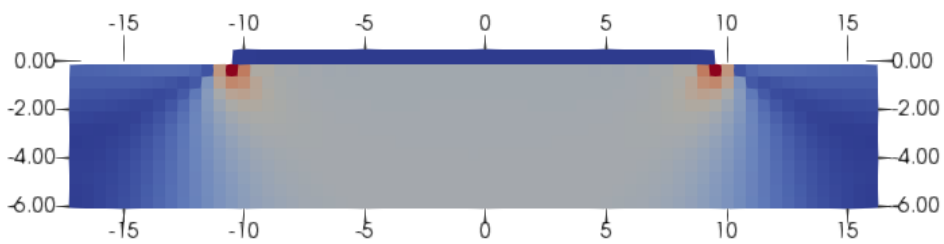
## 4.5 Rigid Flat Punch Problem



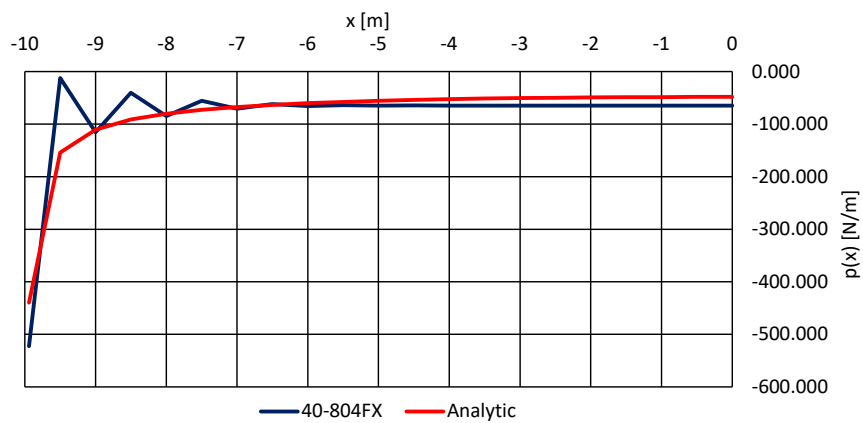
(a) Initial state



(b) Deformed state

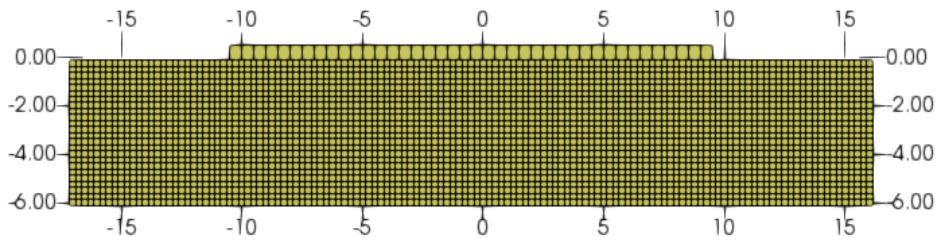


(c) Magnitude of stress

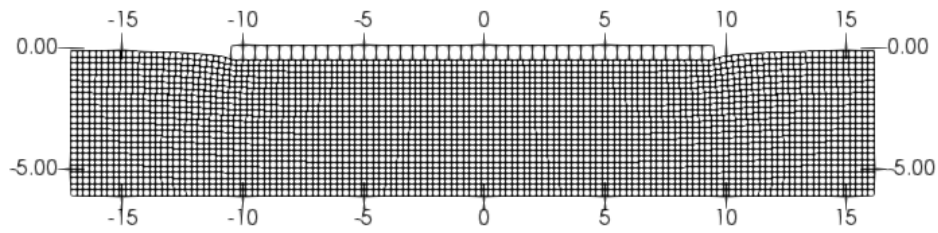


(d) Stress distribution along the contact surface

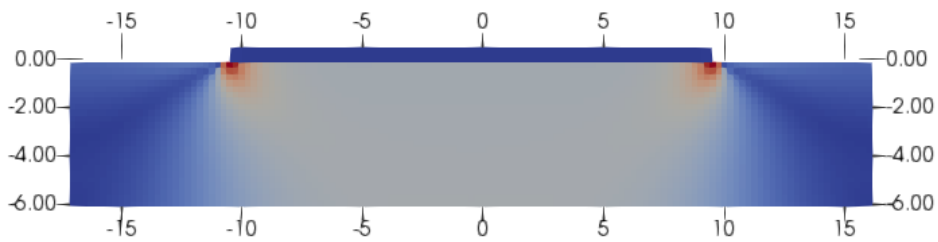
**Figure 27:** Rigid flat punch problem: Mesh 40-804FX



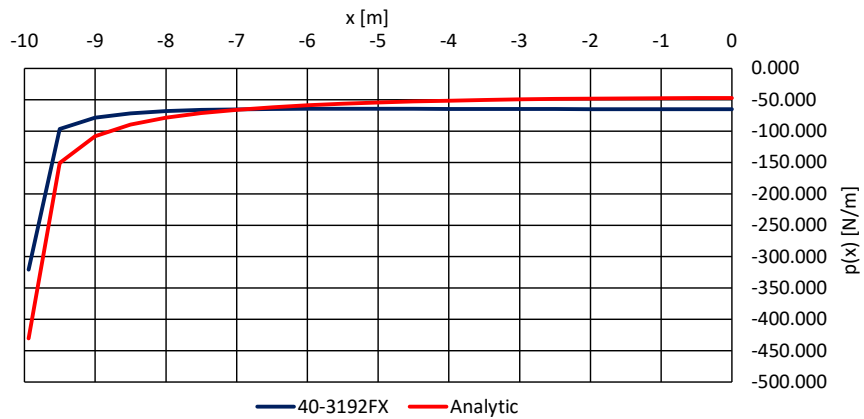
(a) Initial state



(b) Deformed state



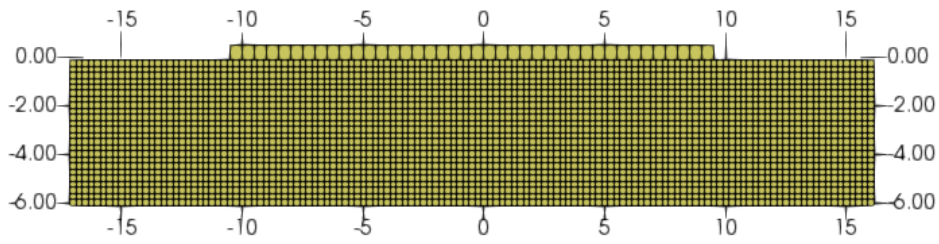
(c) Magnitude of stress



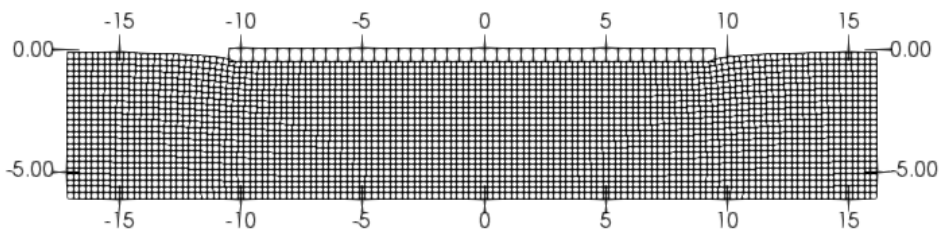
(d) Stress distribution along the contact surface

Figure 28: Rigid flat punch problem: Mesh 40-3192FX

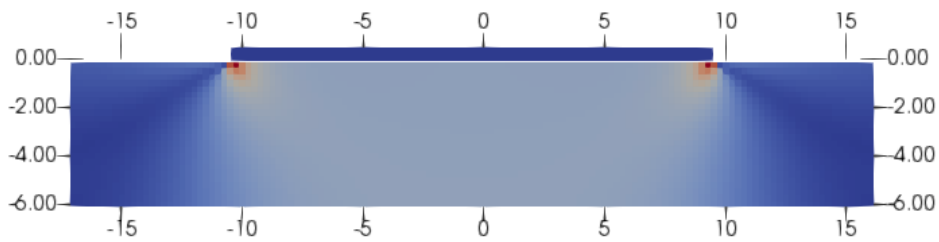
## 4.5 Rigid Flat Punch Problem



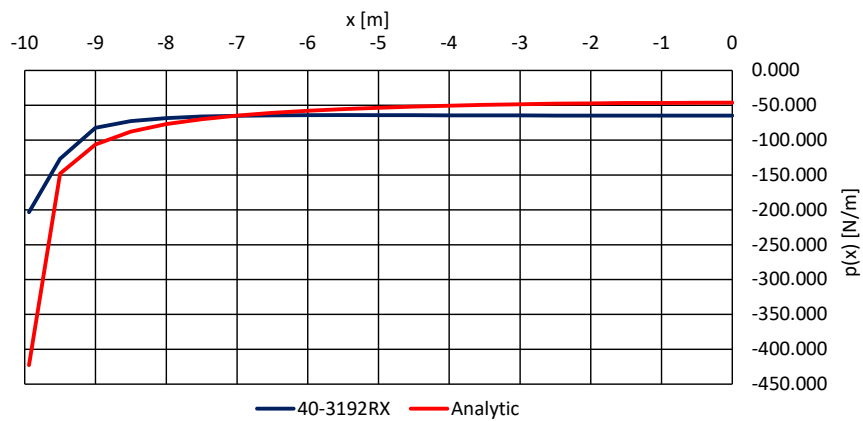
(a) Initial state



(b) Deformed state



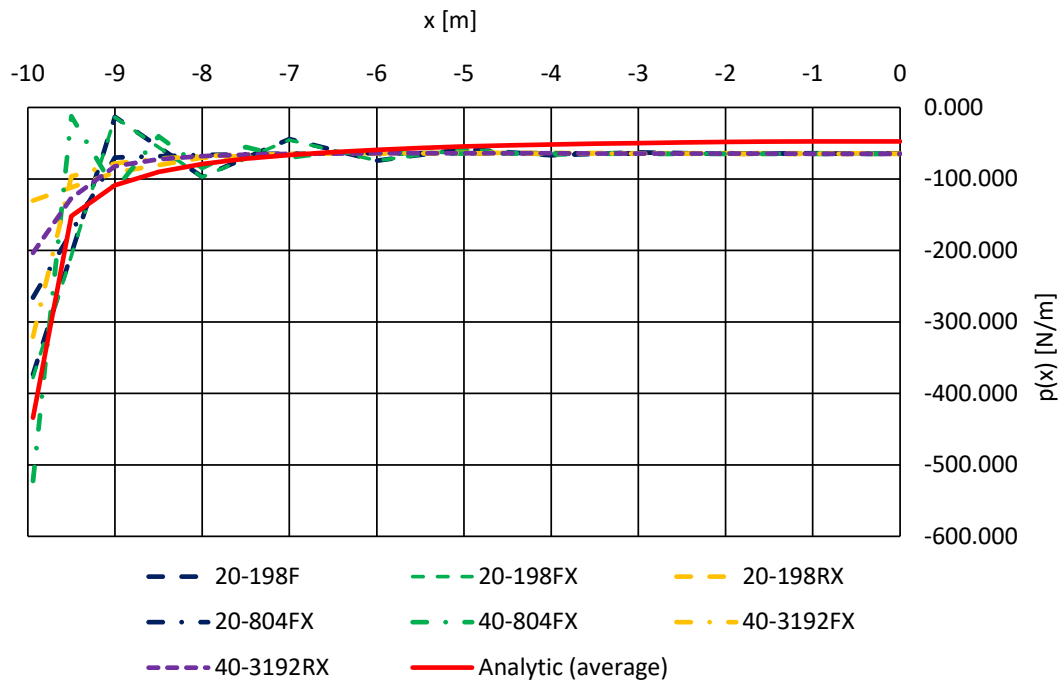
(c) Magnitude of stress



(d) Stress distribution along the contact surface

**Figure 29:** Rigid flat punch problem: Mesh 40-3192RX





**Figure 30:** Rigid flat punch problem: Comparison of all numerical results against an average analytical result

From the presented results, many interesting conclusions may be drawn. Firstly, however, a comparison of results of meshes 20-198F and 20-198FX (pictured in figures 23 and 24), which differ only in the fixation, respectively non-fixation of the sides of the half-space model, proves that this has barely any effect on the final result. Therefore, for all each subsequent meshes, the sides always remained fixed.

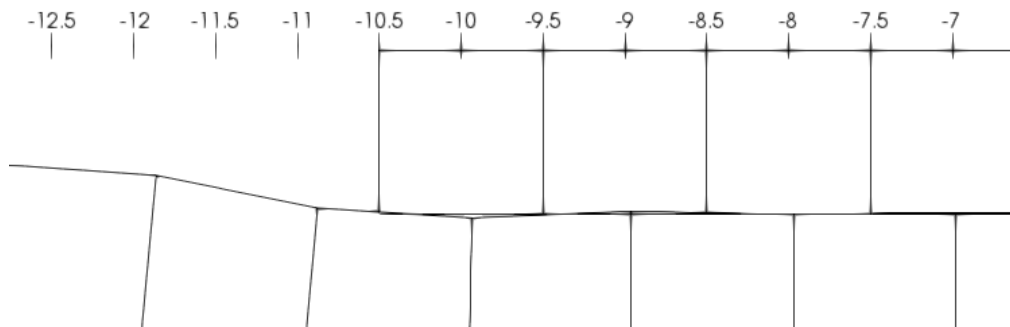
The rough meshes have apparently a tendency to produce an oscillating pressure curve. This is naturally a computation error, which can be explained upon closer examination of the deformed state. Due to the large elements used for the discretization a situation arises where, especially closer to the end of the contact surface, the elements of the half-space arrange into a sort of zig-zag pattern, seen in figure 31. This is due to the limited number of points for which contact is enforced. Due to this phenomenon, every alternate node is actually *not* in full contact in the final deformed state, and reports a much smaller reaction as a consequence.

If we compare meshes 20-804FX and 40-804FX (in figures 26 and 27), we find that 40-804FX, despite being finer than 20-804FX, results in a much worse behavior in this regard. It can be concluded from that that this phenomenon is not actually tied as much to the element size itself, but rather to whether the element size in both stamp and half-

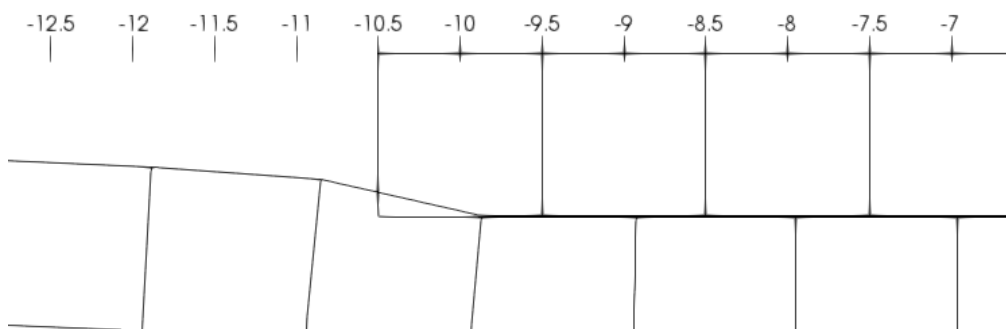
## 4.5 Rigid Flat Punch Problem

space agrees. Mesh 40-804FX, same as the 20-198 family of meshes, has all the elements of the same dimensions, while in 20-804FX, the half-space elements are half the size of the stamp elements.

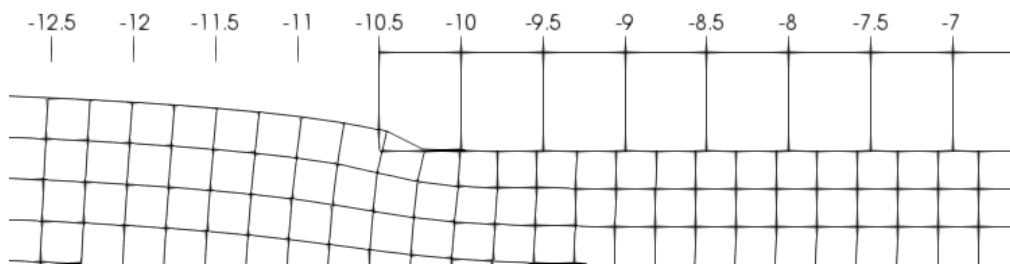
It shall be noted that despite half the reactions being very small, this is in sum compensated by the other reactions and the total magnitude of loading force remains comparable with the numbers reported by the meshes unaffected by the oscillating pressure (compare in table 8).



**Figure 31:** Rigid flat punch problem: A detail of deformed elements near the edge of the contact surface (mesh 20-198FX)



(a) Mesh 20-198RX



(b) Mesh 40-3192RX

**Figure 32:** Rigid flat punch problem: A detail of penetration failure of the R meshes at the edge of contact surface

Another interesting difference can be observed between meshes which define contact segments on the stamp (marked with the letter R) and the rest, which defines contact

segments in the half-space (marked with F). The R meshes perform slightly better in the middle of the contact surface, avoiding totally the oscillation problems described in the previous paragraph, however they exhibit significant failures near its edge. This is because the edge itself is not even a contact point in this configuration, the actual last contact point being half a half-space element inside the contact surface. A slight penetration occurs (pictured in figure 32) and the model is unable to properly catch the rise of the pressure curve at the end. Compare results of 20-198RX and 40-3192RX (figures 25 and 29) to see that this disadvantage diminishes with the finer refinement of the half-space, but remains still noticeable in comparison with the F meshes. Also thanks to this, the R meshes report noticeably lower total force loads in table 8.

In conclusion, apart from proving reasonable agreement between OOFEM results and the benchmark analytical solution, this set of experiments allows to make some important observations about the consequences of meshing contact tasks. It is apparent that modelling meshes and defining contact conditions may have significant effects on the resulting solution and therefore all such actions shall be treated with the utmost care in each case.

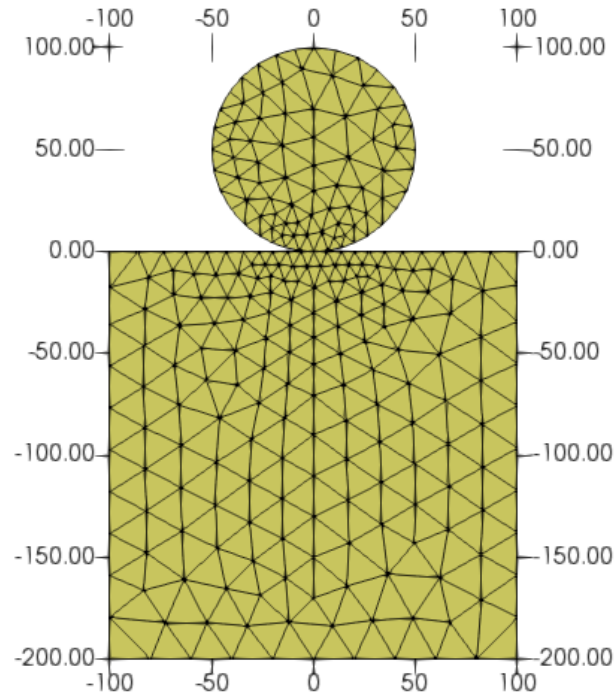
## 4.6 Hertz Contact Problem

In 1881, Heinrich Hertz published his article *On the contact of elastic solids* [Hertz, 1881], laying the foundation of the scientific field of contact mechanics. The problem described and solved in the article is now known as the *Hertz contact problem*, being one of the typical, if not the most typical, problem used as a benchmark for contact software computations [Konyukhov and Izi, 2015].

Frictionless contact of two elastic solid bodies is considered. There are some rather restrictive conditions which have to be fulfilled [Konyukhov and Izi, 2015], namely

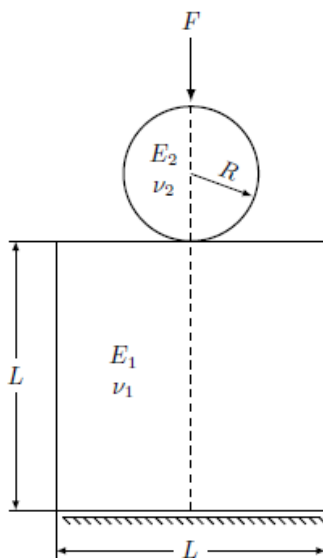
- convexity and smoothness into the second derivative of the contacting surfaces
- contact of opposite convex surfaces
- contact area  $a$  is much smaller than the size of either of the bodies
- the radius of curvature of either contact surface is much larger than  $a$
- contact is frictionless
- both bodies are of an elastic material, geometrically and materially linear

Several similar experiments were performed. The problems involve a cylindrical body modelled with two-dimensional elements contacting with a flat surface. The surface is either rigid or itself part of an elastic body. The mesh used is a relatively rough mesh of triangular elements for the cylindrical body, complemented when appropriate by a similarly rough mesh for the flat body beneath it (see figure 33).



**Figure 33:** Hertz contact problem: Mesh

The first experiment is inspired by an experiment conducted in [JuliaFEM, 2019]. The rough mesh is used to model an elastic cylinder being pressed into an elastic block by application of a force on top of the cylinder. The situation is pictured in figure 34. Table 9 summarizes the input data.



**Figure 34:** Hertz contact problem: Sketch of the cylinder and block, taken from [JuliaFEM, 2019]

**Table 9:** Hertz contact problem: Input data for two elastic bodies

Parameter	Value
$F$	350 kN
$R$	50 mm
$L$	200 mm
$E_1$	210 GPa
$E_2$	70 GPa
$\nu_1$	0.3
$\nu_2$	0.3

According to the Hertz formulas, the average stiffness  $E$ , the maximum contact pressure  $p_0$  and the size of the contact area  $a$  can be determined for these input values as [JuliaFEM, 2019]

$$E = \frac{2E_1E_2}{E_2(1 - \nu_1^2) + E_1(1 - \nu_2^2)} = 115\,385 \text{ MPa} \quad (78)$$

$$p_0 = \sqrt{\frac{FE}{2\pi R}} = 11\,337 \text{ MPa} \quad (79)$$

$$a = \sqrt{\frac{8FR}{\pi E}} = 19.64 \text{ mm} \quad (80)$$

Two computations were performed in OOFEM. The first used node-to-node contact discretization, taking advantage of a specially for this case programmed feature allowing the user to pre-select the projection vector. Therefore the projection vector was fixed to always be in the vertical direction, avoiding issues with nodes of the cylinder not being in perfect alignment with the nodes of the flat surface. This boundary condition uses the penalty method, with the penalty parameter set to  $p = 10\,000 \text{ kN/mm}$ .

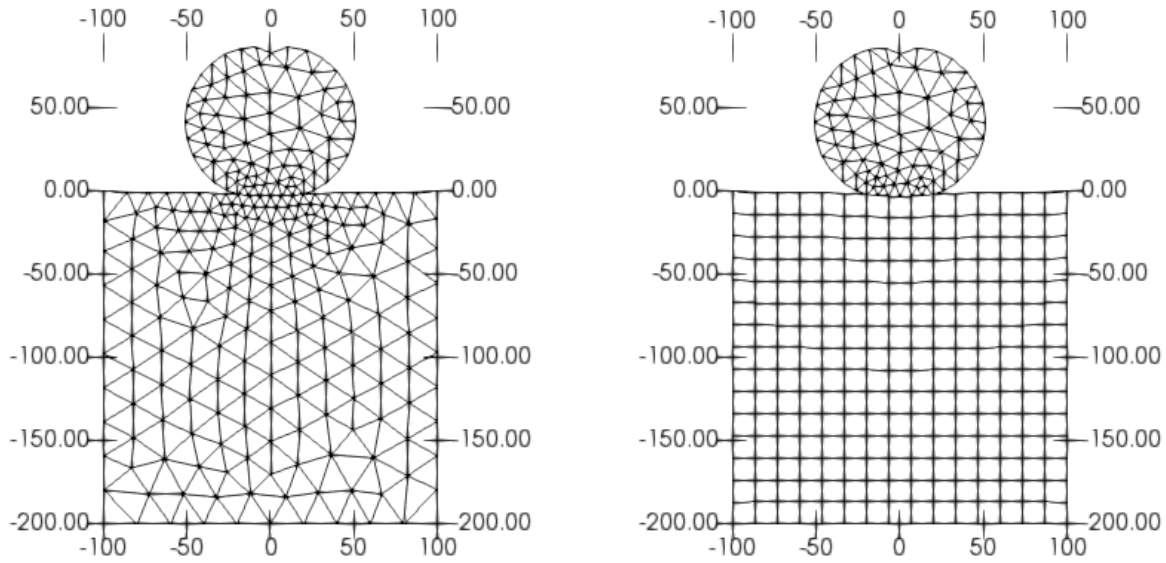
The second computation uses a regular node-to-segment discretization with a different (square-based) mesh for the block to improve model performance. For the same reason, the node-to-segment model is loaded by prescribed displacement (so tuned that the reaction in the loaded node matches the prescribed loading force). The boundary condition uses Lagrangian multipliers.

From each computation, the maximum contact pressure and the size of the contact area were collected<sup>17</sup>. Table 10 compares those results with the analytical solution. Figure 35 shows the deformed state of the mesh for both computations.

**Table 10:** Hertz contact problem: Results for two elastic bodies

Parameter	Analytical	Node-to-node (penalty)	Node-to-segment (LMs)
$p_0$	11 337 MPa	11 142 MPa	10 647 MPa
$a$	19.64 mm	19.59 mm	19.55 mm

<sup>17</sup>defined as the largest pressure in the  $y$  direction found in the nodes of the contact surface and as the position of the last node of the cylinder for which the contact condition was activated, respectively



(a) The node-to-node discretization

(b) The node-to-segment discretization

**Figure 35:** Hertz contact problem: Deformed state of two elastic bodies

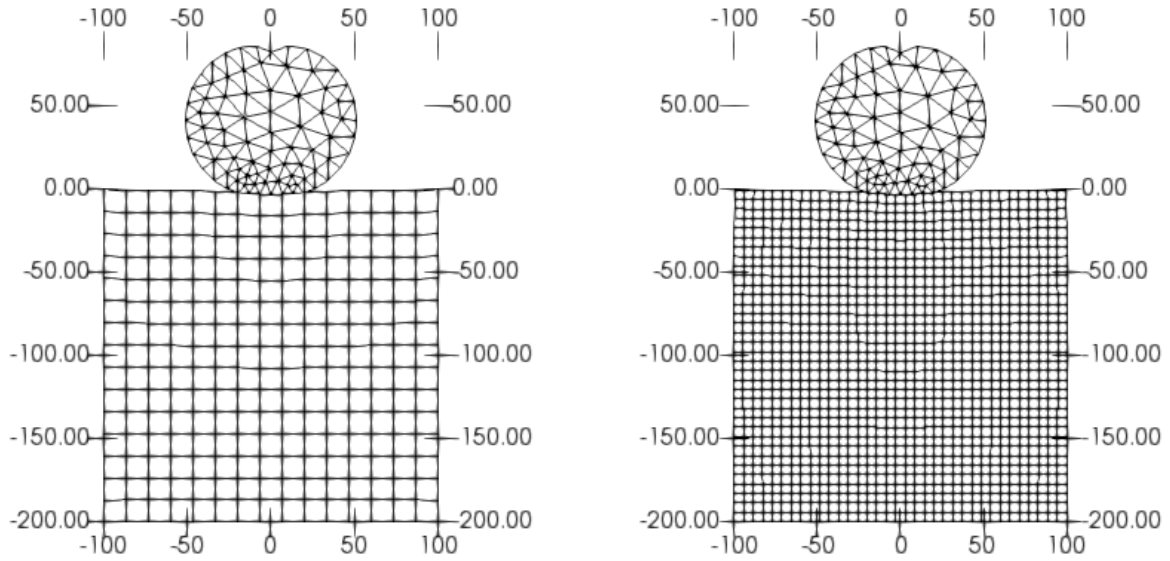
Despite the relative roughness of meshing, the results are very satisfactory. The node-to-segment discretization performs slightly worse in this case. This minor difference can be attributed to the different mesh configuration, however.

To test that that is indeed the case, a verification was performed using a finer mesh for the block in the node-to-segment case. The expectation is that the result shall improve and converge to the analytical solution.

A comparison between the results of the original rough mesh and the finer mesh is shown in table 11 and visually in figure 36. It is apparent that indeed the results improved as expected.

**Table 11:** Hertz contact problem: Results of verification for two elastic bodies

Parameter	Analytical	Original mesh (penalty)	Finer mesh (LMs)
$p_0$	11 337 MPa	10 647 MPa	11 152 MPa
$a$	19.64 mm	19.55 mm	19.57 mm



(a) Original node-to-segment computation      (b) Verifying computation with a finer mesh

**Figure 36:** Hertz contact problem: Verification of results for two elastic bodies

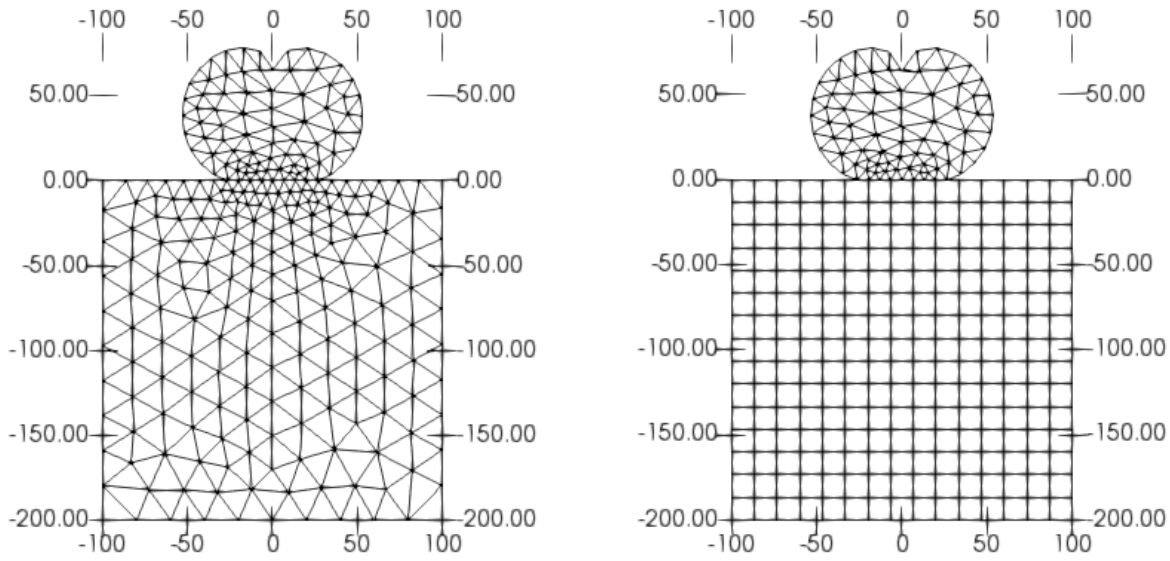
In the next experiment, the node-to-node and node-to-segment (with the original mesh) computations shall be repeated. Now, however, the block shall be fixed in place, mimicking an infinitely stiff rigid barrier. To obtain the analytical solution, expression (78) can be reconsidered with the assumption that  $E_1 \rightarrow \infty$ , yielding

$$E = \lim_{E_1 \rightarrow \infty} \frac{2E_1E_2}{E_2(1 - \nu_1^2) + E_1(1 - \nu_2^2)} = \frac{2E_2}{1 - \nu_2^2} = 153\,846 \text{ MPa} \quad (81)$$

To increase the size of the contact area and allow the mesh to properly portray the deformation, the loading force has been increased to  $F = 900 \text{ kN}$ . Both computations use the penalty approach. The resulting analytical solutions for the maximum of pressure and for the contact area are in table 12 together with the results of the computations. The deformed state is pictured in figure 37.

**Table 12:** Hertz contact problem: Results for an elastic-rigid configuration

Parameter	Analytical	Node-to-node (penalty)	Node-to-segment (penalty)
$p_0$	20 994 MPa	19 995 MPa	20 556 MPa
$a$	27.22 mm	26.55 mm	26.54 mm



(a) The node-to-node discretization

(b) The node-to-segment discretization

**Figure 37:** Hertz contact problem: Deformed state of an elastic-rigid configuration

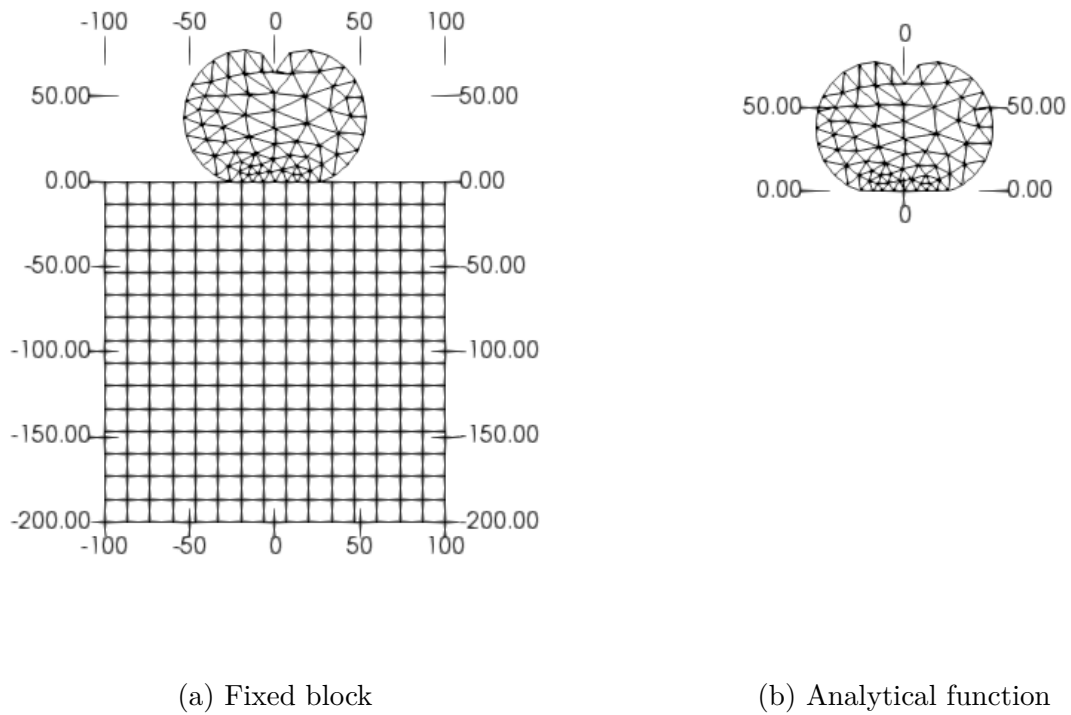
The computed values are again lower than the analytical solution and the accuracy is slightly worse compared with the previous experiment. Most notably, opposite to the previous case, the node-to-segment discretization performs better than the node-to-node discretization now. That could be explained by two factors. Firstly, the segments cannot deform any longer and thus any disadvantages of their mesh do not manifest anymore. Secondly, the deformations are larger now, which, in the OOFEM implementation, is more suited to the node-to-segment approach.

As a last experiment, the use of an analytical function shall be used to compute the same elastic-rigid task. The block elements are removed and substituted by an analytically defined barrier. If the contact conditions in OOFEM work properly, the results should be exactly the same as the results for the fixed segments. The comparison satisfying this expectation is provided in table 13 and figure 38.

**Table 13:** Hertz contact problem: Results of verification for an elastic-rigid configuration

Parameter	Analytical	Fixed block (penalty)	Analytical barrier (penalty)
$p_0$	20 994 MPa	20 556 MPa	20 556 MPa
$a$	27.22 mm	26.54 mm	26.54 mm





**Figure 38:** Hertz contact problem: Comparison of deformed state of the elastic-rigid case - fixed block vs an analytical function

## 5 Conclusion

It was the goal of this thesis to introduce contact mechanics into the OOFEM software. In this regard, it can be declared that the goal was successfully fulfilled.

Contact mechanics is a very wide field encompassing many different physical problems and algorithms intended to tackle them. The ambition was never to implement every single one of those in the restricted time frame of the thesis. Originally, there was no contact code in OOFEM whatsoever. Now, as of the submission date of this thesis, the existing implementation is focused on two-dimensional cases of frictionless contact, using the two formulations discussed in the theoretical section - the penalty method and the Lagrangian multiplier method. Section 3.4 discusses in detail how future additions and extensions may tie into this framework.

The experiment section 4 provides several examples of experiments verifying the proper work of the contact algorithms implemented. It has been successfully demonstrated that reliable results are achieved both in detailed small cases as well as for large structures involving significant numbers of elements.

To conclude, the aims of the thesis have been met. The modified OOFEM code now includes a basic implementation of contact mechanics, which is open and ready for future expansions.

## List of Figures

1	A spring-mass system in contact, taken from [Wriggers, 2006] . . . . .	7
2	A simplified diagram of object relations within OOFEM . . . . .	18
3	Implementation of node-to-node penalty contact within OOFEM environ- ment . . . . .	21
4	Implementation of node-to-node Lagrangian multiplier contact within OOFEM environment . . . . .	24
5	An overview of all classes pertinent to node-to-segment contact and their hierarchy . . . . .	28
6	Implementation of node-to-segment boundary conditions within OOFEM environment . . . . .	30
7	Implementation of an element edge contact segment within OOFEM envi- ronment . . . . .	33
8	Projection of a point to a straight line: geometrical illustration of the <code>computeDistanceVector()</code> function . . . . .	34
9	Implementation of analytical function contact segments within the OOFEM environment. <code>PolynomialContactSegment</code> and <code>CircleContactSegment</code> presented as examples of derived classes . . . . .	37
10	A displacement-driven development test of node-to-node contact . . . . .	44
11	A force-driven development test of node-to-node contact . . . . .	44
12	A displacement-driven development test of node-to-segment contact . . . . .	45
13	A verification test of node-to-node contact in the X direction . . . . .	46
14	A verification test of node-to-node contact with 8 elements . . . . .	47
15	Contact of two bars: Initial geometry, taken from [Wriggers, 2006] . . . . .	49
16	Contact of two bars: OOFEM mesh . . . . .	50
17	Contact of two bars: OOFEM deformed state . . . . .	52
18	Two bars with penalty condition: OOFEM mesh . . . . .	52
19	Two bars with penalty condition: Results from literature [Konyukhov and Izi, 2015] . . . . .	54
20	Two bars with penalty condition: Results from OOFEM . . . . .	54
21	Penalty size study: Initial geometry of the task used . . . . .	57
22	Rigid flat punch problem: Sketch of situation, taken from [Konyukhov and Izi, 2015] . . . . .	58
23	Rigid flat punch problem: Mesh 20-198F . . . . .	62
24	Rigid flat punch problem: Mesh 20-198FX . . . . .	63
25	Rigid flat punch problem: Mesh 20-198RX . . . . .	64
26	Rigid flat punch problem: Mesh 20-804FX . . . . .	65
27	Rigid flat punch problem: Mesh 40-804FX . . . . .	66

28	Rigid flat punch problem: Mesh 40-3192FX . . . . .	67
29	Rigid flat punch problem: Mesh 40-3192RX . . . . .	68
30	Rigid flat punch problem: Comparison of all numerical results against an average analytical result . . . . .	69
31	Rigid flat punch problem: A detail of deformed elements near the edge of the contact surface (mesh 20-198FX) . . . . .	70
32	Rigid flat punch problem: A detail of penetration failure of the R meshes at the edge of contact surface . . . . .	70
33	Hertz contact problem: Mesh . . . . .	72
34	Hertz contact problem: Sketch of the cylinder and block, taken from [Juli- aFEM, 2019] . . . . .	72
35	Hertz contact problem: Deformed state of two elastic bodies . . . . .	74
36	Hertz contact problem: Verification of results for two elastic bodies . . . .	75
37	Hertz contact problem: Deformed state of an elastic-rigid configuration . .	76
38	Hertz contact problem: Comparison of deformed state of the elastic-rigid case - fixed block vs an analytical function . . . . .	77

---

## List of Tables

1	A verification test of analytical function contact segments: Displacements before and after activation of a penalty contact condition . . . . .	48
2	Contact of two bars: Input values . . . . .	50
3	Contact of two bars: Comparison of Analytical and OOFEM results . . . .	51
4	Two bars with penalty condition: Input values . . . . .	53
5	Penalty size study: Experiment data . . . . .	56
6	Rigid flat punch problem: Task parameters common for all mesh configurations . . . . .	60
7	Rigid flat punch problem: Overview of mesh configurations . . . . .	60
8	Rigid flat punch problem: Computed loading forces for each mesh configuration . . . . .	61
9	Hertz contact problem: Input data for two elastic bodies . . . . .	72
10	Hertz contact problem: Results for two elastic bodies . . . . .	73
11	Hertz contact problem: Results of verification for two elastic bodies . . . .	74
12	Hertz contact problem: Results for an elastic-rigid configuration . . . . .	75
13	Hertz contact problem: Results of verification for an elastic-rigid configuration . . . . .	76

## List of Listings

1	An excerpt from an OOFEM input file initializing a node-to-node penalty contact boundary condition . . . . .	20
2	An excerpt from an OOFEM input file initializing a node-to-node Lagrangian multiplier contact boundary condition . . . . .	25
3	Contact segment definition block within the OOFEM input file . . . . .	28
4	An excerpt from an OOFEM input file initializing a node-to-segment penalty contact boundary condition . . . . .	29
5	An excerpt from an OOFEM input file initializing a node-to-segment Lagrangian multiplier contact boundary condition . . . . .	29
6	An excerpt from an OOFEM input file initializing an element edge contact segment, including the initialization of the set of element edges referred to	33
7	An excerpt from an OOFEM input file initializing a circular analytical function contact segment . . . . .	38
8	An excerpt from an OOFEM input file initializing a polynomial function contact segment . . . . .	40
9	Contact of two bars: Node, element, material and boundary condition definitions from the OOFEM input file . . . . .	49

---

## References

- [Ayachit, 2015] Ayachit, U. (2015). *The ParaView Guide: A Parallel Visualization Application*. Kitware, 1. edition.
- [Felippa, 1975] Felippa, C. A. (1975). Solution of linear equations with skyline-stored symmetric matrix. *Computers & Structures*, 5(1):13–29.
- [Gould et al., 2007] Gould, N. I., Scott, J. A., and Hu, Y. (2007). A numerical evaluation of sparse direct solvers for the solution of large sparse symmetric linear systems of equations. *ACM Transactions on Mathematical Software (TOMS)*, 33(2):10.
- [Hertz, 1881] Hertz, H. (1881). On the contact of elastic solids. *Z. Reine Angew. Mathematik*, 92:156–171.
- [JuliaFEM, 2019] JuliaFEM (2019). Documentation. <http://www.juliafem.org/JuliaFEM.jl/latest/>. Accessed: 31-12-2019.
- [Konyukhov and Izi, 2015] Konyukhov, A. and Izi, R. (2015). *Introduction to computational contact mechanics*. Wiley, Chichester, West Sussex.
- [MATLAB, 2017] MATLAB (2017). *Version 9.3.0 (R2017b)*. The MathWorks Inc., Natick, Massachusetts.
- [Patzák, 2000] Patzák, B. (2000). OOFEM home page. <http://www.oofem.org>.
- [Patzák, 2019] Patzák, B. (2019). Numerická analýza konstrukcí 2. University Lecture.
- [Virus, 2018] Virus, M. (2018). *Programování v C++*. Grada Publishing, Prague, 1st edition.
- [Wriggers, 2006] Wriggers, P. (c2006). *Computational contact mechanics*. Springer, New York, 2nd ed edition.
- [Yastrebov, 2013] Yastrebov, V. A. (2013). *Numerical methods in contact mechanics*. Wiley, Hoboken, NJ.
- [Zienkiewicz and Taylor, 2000] Zienkiewicz, O. C. and Taylor, R. L. (2000). *The finite element method*. Butterworth-Heinemann, Boston, 5th ed edition.