PhD-FSTM-2025-011
The Faculty of Science, Technology and Medicine

DISSERTATION

Presented on 09/01/2025 in Esch-sur-Alzette

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG
EN Sciences de l'Ingénieur

by

Diego R. HURTADO CATHALIFAUD
Born on 23 August 1989 in Coquimbo, Chile

Optimization solvers and machine learning to enhance quasi-static unilateral contact simulations

Dissertation defence committee

Dr. Lars A. A. Beex	Supervisor	Université du Luxembourg
Dr. Bernhard Peters	Chair	Université du Luxembourg
Dr. Ondrej Rokos	Vice chair	Eindhoven University of Technology
Dr. Reza Talemi	Member	Katholieke Universiteit Leuven
Dr. Marco Magliulo	Member	External examiner

Acknowledgements

Completing this PhD has been a long and intense journey, and I would not have reached this milestone without the invaluable support of many people.

First and foremost, I want to express my deepest gratitude to my supervisor, Dr. Lars A. A. Beex. Over these past five years, our relationship has been a mix of both challenge and growth. There were moments of struggle and moments of great collaboration, but through it all, Lars has been a reliable source of technical and emotional support. He always pushed me forward and provided guidance through the hardest times.

I am also sincerely grateful to Prof. Bernhard Peters for his role as Chair in my defense committee for his involvement and kind support in this process. My appreciation extends as well to Dr. Ondrej Rokos and Dr. Reza Talemi for the time and effort they dedicated to reviewing my thesis and for the insightful discussions following my defense. Their participation and thoughtful feedback during the process were greatly appreciated.

I also want to express my gratitude to Dr. Marco Magliulo, whom I was fortunate to meet during my master's studies and with whom I shared an office during the early days of my PhD. Marco gave me helpful guidance in coding and technical aspects early on, which contributed to my work. His support throughout the years has been very valuable.

Additionally, I would like to thank Prof. Andreas Zilian for allowing me to join the research team that would eventually become my PhD group. He made this journey possible from the very beginning and provided valuable feedback during our CET meetings, for which I am very grateful.

To my colleagues and office mates—Marco, Li, Camilo, and Cem—thanks for sharing this journey with me, for the fun discussions, and for making the office a good place to be. I also want to thank all my colleagues with whom I shared not just lunch but also plenty of activities, beers, sports, and great moments. You all made the tough times easier.

A special thank you goes to my girlfriend, Pauli, who I love and has been by my side throughout my PhD, celebrating my victories and supporting me through the tough times. Her patience, encouragement, and support mean the world to me. From climbing to beautiful travels, I look forward to many more adventures together.

To my closest friends—Rawezh, Camila, Kay, Seba, Marielle, Diego K., Natsu, Natha, and Amanda—thank you for being my family away from home. You've been there for the good and the bad, and I'm grateful for every moment we've shared.

I also want to thank my climbing friends, who have always been around, making sure I got some fresh air and a break from work.

Last but certainly not least, I want to thank my family—my parents, my sister, and my niece—whose love and support have always given me the strength to keep going. Their encouragement has meant everything to me. I also want to acknowledge my extended family—my cousins, aunts, and uncles—who have always shown great interest in my journey and celebrated my milestones with me.

This PhD has been a demanding yet rewarding chapter of my life, and I am grateful to everyone who has played a part in making it possible. Obviously, there are more people to thank, and I hope they understand that even if I don't mention everyone, they have also been part of this. Thank you all.

Financial support

This work is part of the Doctoral Training Unit Data-driven computational modelling and applications (DRIVEN) funded by the Luxembourg National Research Fund under the PRIDE programme (PRIDE17/12252781).

Abstract

This thesis focuses on computational methods to describe the mechanics of deformable bodies in unilateral contact. This field of research aims to develop numerical simulations that predict the mechanics that occur when a deformable body comes in contact with another deformable body, a rigid body or with itself. A diverse range of questions in this research domain remains open to this day. The goal of this thesis is to help answer two of them. The first part of this thesis investigates if optimization solvers can be exploited to predict the mechanics of a quasi-static, rate-independent deformable solid body when it experiences snap-back due to its contact with a rigid body. The second part of this thesis investigates if machine learning can be exploited to accelerate unilateral contact simulations.

The first part of this thesis revolves around snap-back occurring due to unilateral contact. Snap-back occurs when (a part of) the energy stored in a quasi-static deformable body is suddenly released. Quasi-static deformable bodies in contact are particularly prone to experience snap-back. Conventional quasi-static simulations are not able to treat snap-back. A well-known remedy is to enhance the quasi-static simulation with an arc-length method. Another remedy is to perform a dynamical simulation instead of a quasi-static simulation. Both remedies come with their own advantages, disadvantages and limitations. To this purpose, the first part of this thesis investigates the capabilities of four optimization algorithms to describe the mechanics of a quasi-static, rate-independent deformable body when it experiences contact-induced snap-back. The presented investigation considers 2D and 3D scenarios, an elastic and elastoplastic mechanical model for the deformable body and different refinement levels to spatially discretize the deformable body.

The machine learning algorithms under investigation in the second part of this thesis are neural networks. First, an existing neural network is investigated for its ability to replace the entire contact detection algorithm of the simulations. Second, a new neural network is proposed to replace only the most time-consuming tasks of the conventional contact detection algorithm. This hybrid strategy of only replacing a part of the existing algorithm and not interfering with the other part makes use of the speed of the neural network but also allows for the fact that neural networks are not perfect. The reason is that the final task of the conventional contact detection algorithm, which is responsible for the high accuracy, is the part that remains untouched. The proposed network is a multi-task neural network that simultaneously classifies and emulates functions. Because the network's classifier and function emulator exchange information during each forward pass, its accuracy is sufficient for use in true contact simulations.

Contents

Chapter 1

Introduction

The topic of the thesis in front of you is situated in the research domain of computational mechanics of solids. This domain aims to develop fast, accurate and robust numerical simulations to predict the mechanical behaviour of deformable solid bodies. These simulations are crucial for industries such as the manufacturing industry and the construction industry because they enable companies to virtually assess the manufacturing process of existing and yet-to-be-developed products, as well as the mechanical behaviour of the products during service life.

This thesis focuses on computational methods to describe the mechanics of deformable bodies in unilateral contact. This field of research aims to develop numerical simulations that predict the mechanics that occur when a deformable body comes in contact with another deformable body, a rigid body or with itself. Unilateral contact is thus not only important for contact between several bodies [1], it can also be essential for a single body, for instance if delamination [2] or a crack occurs [3, 4].

An important part of the research in this domain of the last 25 years has aimed at tackling the following problem. A deformable body requires a spatial discretization in order to describe its mechanical behaviour in numerical simulations. The spatial discretization usually entails that the body's surface is C^0-continuous. The lack of surface continuity (i.e. surface smoothness) poses substantial problems for simulations that utilize the fastest and most conventional contact framework, the node-to-segment framework [5, 6].

The first class of studies aims to solve the problem described in the previous paragraph by improving the body's surface smoothness [7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]. Thus, these studies formulate new surface descriptions for deformable bodies such that the robustness of node-segment frameworks improves.

Instead, the second class of studies aims to solve the aforementioned problem by replacing the conventional node-to-segment contact framework with segment-to-segment frameworks, also known as mortar methods [19, 20, 21, 22, 23, 24, 25]. It is reasonable to state that a relevant part of the research community is convinced that segment-to-segment frameworks outperform, or will outperform conventional node-to-segment frameworks in terms of simulation robustness. Nevertheless, mortar methods are more time-consuming than conventional node-to-segment approaches. Moreover, commercially available software packages, such as Abaqus, use node-to-segment approaches as the default contact framework.

The principle aims of this thesis are not intended to improve node-to-segment frameworks or segment-to-segment approaches. Somewhat remarkably nevertheless, the approach developed in the first part of this thesis does improve the robustness of node-to-segment contact frameworks. Furthermore, the developments of the second part of this thesis are formulated for node-to-segment frameworks, but can also be exploited by certain mortar methods. The two main problems and associated aims under investigation in this thesis can be concisely described as follows.

- When in contact, a minor change in a deformable body's translation or rotation can induce a sudden release of the energy stored in the body. This phenomenon is called 'snap-back' and conventional quasi-static simulations cannot treat this, entailing that the simulation cannot proceed. The two remedies that could be applied in such a scenario come with their own drawbacks. The first aim of this thesis is to investigate if optimization solvers can be used as an alternative remedy to treat snap-back in quasi-static rate-independent unilateral contact simulations.

- The second aim of this thesis is related to the recent increase of artificial intelligence in society. The second aim is to investigate whether neural networks can be used to replace the part of the contact simulations that is responsible for treating contact. To this purpose, both the capabilities of an existing neural network are investigated, and a new neural network is proposed.

The remainder of this introduction chapter consists of two subsections that consecutively explain the two objectives of the thesis in more detail. They also relate the two aims and the associated investigations to the existing literature. Consequently, the reader will be confronted with an increase in technical and scientific terminology in the next subsections.

The remainder of this thesis, after the current chapter, is divided into three chapters. Chapter 2 focuses on optimization solvers to treat snap-back. Chapter 3 focuses on two neural networks to emulate the algorithm responsible for evaluating contact in conventional contact simulations. Finally, Chapter 4 presents several conclusions.

1.1 Aim of Chapter 2 and its relation to existing literature

Newton's method is the preferred solver for quasi-static mechanical (contact) simulations (and for many others, for that matter) because of its quadratic convergence, i.e. its simulation speed. The problem of interest in Chapter 2 is that Newton's method does not converge if one or more deformable bodies undergo snap-back. Deformable bodies in contact are particularly prone to experience snap-back, and snap-through for that matter because the bodies are compressed when in contact.

If snap-back or snap-through is expected to occur, one course of action for contact simulations could be to change a quasi-static simulation to a dynamical one [24], even if the mechanical behaviour, and not the dynamical behaviour, is of principal interest. However, dynamical simulations are slower and require additional parameters to be tuned or identified (e.g. densities and viscosities).

Another course of action is to stick to the quasi-static paradigm and Newton solvers and to make use of arc length methods [26, 27, 28, 29, 30]. However, a disadvantage of arc length methods is that Newton's method is not guaranteed to converge if the snap-back is so severe that the pre-snap-back force response and the snap-back force response are significantly similar. Another disadvantage is that they require boundary conditions to be in place because invertible stiffness matrices are required in Newton's method and because arc length methods modify the boundary conditions. Consequently, arc length methods cannot be used if no boundary conditions are in place (see e.g. the multibody problem in [1]).

The aim of Chapter 2 is to investigate the capabilities of optimization solvers as alternative solvers to treat snap-back in quasi-static unilateral contact simulations. Optimization solvers could be a good alternative to the two aforementioned strategies to treat snap-back, because 1) the simulations remain quasi-static, thus circumventing the need to tune any dynamical parameters, and 2) the potential problems of arc length methods may be avoided (e.g. the lack of convergence in case of severe snap-back and problems if no boundary conditions are applied).

Investigating optimization solvers for unilateral contact simulations is relatively new. The work of Chapter 2 is most closely related to the recent work of Tupek and Talamini [31] and to a lesser extent to the recent study of Houssein et al. [32]. The reason is that these two studies also focus on optimization solvers for quasi-static unilateral contact simulations. Of these two previous studies, Chapter 2 matches the least that of Houssein et al. [32], because Chapter 2 investigates the potential of optimization solvers when snap-back occurs, whilst the aim of [32] is to investigate if the interior point method can be used for frictional contact if Coulomb friction is regularized using a series of Tresca models [33, 34]. In other words, Chapter 2 focuses on mechanically non-linear bodies in frictionless contact exhibiting snap-back, whereas [32] focuses on frictional contact between infinitesimally elastic bodies that do not exhibit snap-back, nor snap-through.

Chapter 2 is most closely related to the study of Tupek and Talamini [31]. The most important difference is that [31] focuses on snap-through, whereas Chapter 2 focuses on snap-back. Other differences are that [31] considers hyperelasticity in 2D scenarios using linear triangular finite elements (FE), while Chapter 2 considers both hyperelasticity and hyperelastoplasticity in 2D scenarios using bilinear quadrilateral FEs and in 3D scenarios using trilinear hexagonal FEs. Whereas [31] only investigates the preconditioned trust region (TR) method as optimization solver, Chapter 2 investigates four optimization algorithms: the BFGS method [35, 36, 37, 38, 39], its limited storage variant (L-BFGS) [40, 41, 39], the original TR method [42, 43, 44, 39, 45, 46] and the preconditioned TR method [44, 39, 47, 31]. As the results of Chapter 2 will indicate, the two quasi-Newton methods perform differently than the two TR methods, not just in terms of simulation speed.

Furthermore, Tupek and Talamini are obliged to use their optimization solver for each time increment, because snap-through does not yield failure of Newton's method. On the other hand, this thesis makes use of the fact that Newton's method fails if snap-back is encountered. Hence, simulations in this thesis only deploy an optimization solver for problematic time increments. Another difference is that the thesis only focuses on contact between a deformable body and a rigid body, whereas [31] also considers deformable-to-deformable contact (albeit only in 2D scenarios). Furthermore, this thesis constructs the preconditioner in the preconditioned TR method for each TR iteration independently, whereas [31] only changes the preconditioner if the number of conjugate gradient iterations of the last global TR iteration increases a user-defined threshold. Another difference is that the thesis focuses on frictionless contact, whereas Tupek and Talamini also consider frictional contact, although large sliding of contact surfaces does not occur in their simulations. Moreover, both [31] and this thesis use a node-to-segment framework, but the thesis uses Gregory patches to smoothen the leader surface, whereas Tupek and Talamini smoothen the closest-point projection (in a fashion that is only applicable in 2D scenarios). Finally, the thesis uses the penalty method to enforce contact, while [31] uses augmented Lagrange multipliers.

Newton's method is not a true minimization method because it can converge to any type of stationary point. Hence, switching from Newton's method to a true minimization method that aims to find minimizers is strictly speaking only possible if the mechanical problem can be written as a minimization problem. This entails that Lagrange multipliers can, for instance, not be used to enforce contact. Furthermore, rate-dependent constitutive models can also not be used, strictly speaking. However, hyperelasticity [48, 49, 50], variational plasticity [51, 52, 53, 54] and variational damage [55, 56, 57, 58, 59] can be used.

Chapter 2 investigates optimization solvers for two widely used hyperelastic and hyperelastoplastic constitutive models. The neat aspect is that true incremental variational plasticity in the strictest sense, which considers the plastic variables as independent variables that must be optimized together with the displacements, can be avoided. This entails that the displacement-driven nature of this thesis' simulations is retained. In turn, this has the advantage that the efforts to implement the optimization solvers are somewhat limited if a standard Newton's method is already implemented.

The thesis uses a straightforward node-to-segment contact framework [5] and smoothens the rigid leader-surface using Gregory patches [60], although numerous other smoothening methods exist [7, 8, 9, 10, 11, 13] - that are in certain scenarios completely unavoidable [12, 14, 15, 16, 17, 18]. As a consequence of the employed node-to-segment framework, snap-back does not only occur this thesis' simulations because it is physically supposed to, it also occurs as a side effect of the discreteness intrinsic to the node-to-segment framework. In other words, switching Newton's method for an optimization solver may also be beneficial for node-to-segment contact frameworks.

Thus, it is reasonable to assume that if this thesis would have considered a mortar method [19, 20, 21, 22, 23, 24, 25] instead of a node-to-segment framework, the optimization solvers would need to work substantially less. The reason is that segment-to-segment mortar methods are purposefully formulated to avoid the discreteness of node-to-segment methods. This discreteness is caused by the fact that they enforce non-penetration for each follower-vertex individually. Mortar methods avoid this by integrating the signed distance over the entire contact surface and enforcing this integral to be equal to zero.

Nevertheless, snap-back is physically supposed to happen in numerous contact scenarios, regardless of the contact framework employed, as this thesis demonstrates. Hence, all unilateral contact simulations, including those employing mortar methods, need a strategy to deal with snap-back.

1.2 Aim of Chapter 3 and its relation to existing literature

Chapter 3 contains the second main body of this thesis' work. It evolves around the capabilities of neural networks (NNs) to accelerate existing contact simulations, by only replacing the contact part of the simulations.

NNs are these days regularly investigated and exploited in the domain of computational mechanics. They are, for instance, frequently used to learn constitutive laws from microstructural representative volume element computations [61, 62, 63] and/or the relevant microstructural fields [64, 65, 66]. NNs can furthermore learn to replace entire simulations that are governed by partial differential equations [67, 68, 69, 70, 71, 72], with [73] as the most famous study. NNs can also be taught to predict the coefficients of the basis vectors in reduced order models [74, 75, 76]. ML can furthermore classify groups of solution fields for reduced order models [77, 78] or rapidly select appropriate solutions to generate the most suitable reduced basis [79].

Computational mechanics' subdomain of unilateral contact mechanics has not seen so many ML enhancements yet. An example is the study by Sahin et al. [80]. Sahin et al. propose a physically-informed NN (PINN) for the displacement field and stress field of 2D deformable bodies in quasi-static contact simulations. The aspect that specializes the PINN of [80] in treating frictionless contact is that the loss function incorporates the Karush-Kuhn-Tucker equations that arise in unilateral contact scenarios. Goodbrake et al. [81] also develop a PINN for contact involving a deformable body. The difference is that Goodbrake et al. deal with a single 3D hyperelastic body in contact with a rigid sphere, whereas Sahin et al. consider 2D infinitesimally elastic bodies in contact with each other.

The study of Chapter 3 differs from those of Sahin et al. [80] and Goodbrake et al. [81], because Chapter 3 only replaces the contact detection algorithm within an otherwise conventional contact simulation, while the PINNs of [80] and [81] replace the entire simulation. A previous study that also only replaces the contact detection algorithm is that of Lai et al. [82]. Lai et al. focus on contact between 2D ellipsoidal rigid particles in a discrete element framework. The first NN of [82] outputs a binary value that classifies whether or not two ellipsoids are in contact. The second NN of [82] outputs a scalar that quantifies the amount of penetration. The inputs are two shape parameters for each ellipsoidal particle, the displacement vector between the two particles' centers and the rotation of each particle. In [83], three of the authors of [82] extend the approach to irregular particles in 3D.

The goal of the contact detection algorithm in the simulations of this thesis, which are limited to quasi-static node-to-segment contact between a deformable follower-body and a rigid leader-body, is to determine the signed distance between a point in 3D space and the rigid leader-body, as well as the signed distance's gradient and Hessian with respect to the location of the point of interest. The signed distance is the shortest distance between a point of interest and the rigid-leader body, where the sign indicates if the point is located inside (negative) or outside the body (positive). Thus, the signed distance is used to determine whether or not the point of interest penetrates the leader-surface. However, it is also necessary to determine the contact force and contact stiffness in the contact simulations. In addition to the signed distance, its gradient is also necessary to construct the contact force. The signed distance, its gradient and Hessian are furthermore all three required to construct the contact stiffness.

Since the signed distance is a crucial result of contact detection algorithms, it is only natural to borrow an NN tailored to the signed distance from the field of computer vision [84, 85, 86, 87, 88]. In computer vision, signed distance NNs are used to rapidly visualize shapes.

The approach of [89], named 'Neural-Pull', is identified as a suitable candidate to potentially replace the conventional contact detection algorithm. The reason is that besides the fact that Neural-Pull's output is the signed distance, its cleverly constructed loss function also trains the NN for the signed distance's gradient. In other words, Neural-Pull's input scalars are the three components of a point in 3D and its single output scalar is the signed distance between the point and a 3D body. Yet, if Neural-Pull is differentiated with respect to the three components of the 3D point, the resulting NN should rather accurately output the signed distance's gradient. (If the resulting NN is again differentiated, the new NN should emulate the signed distance's Hessian) Nevertheless, Neural-Pull's loss function consists of a single contribution.

Besides investigating the capabilities of Neural-Pull to replace the conventional contact detection algorithm, Chapter 4 also develops its own NN. Whereas Neural-Pull is envisioned to replace the entire contact detection algorithm, the second NN only emulates the most time-consuming tasks of the conventional contact detection algorithm. However, the task that is most critical to the accuracy of the contact detection algorithm is left untouched. Hence, this hybrid approach is expected to yield excellent accuracy as well as a substantial acceleration.

The conventional contact detection algorithm consists of a broad phase and a narrow phase. The latter is split in two tasks. The broad phase calculates a subset of patches of the leader-surface that contain the closest-point projection (i.e. the point on the leader-surface closest to the point of interest). The first task of the narrow phase is to determine a suitable initial guess of the surface coordinates for each patch that is identified by the broad phase. The second task of the narrow phase is to deploy Newton's method in order to calculate the parametric surface coordinates that exactly match the closest-point projection, for each patch independently. To this purpose, the second task uses the initial guesses that are determined by the narrow phase's first task. (Subsequently, the parametric surface coordinates yield the signed distance, its gradient and its Hessian almost instantaneously.)

The second NN of Chapter 3 replaces the broad phase, as well as the narrow phase's first task. Hence, the NN classifies a subset of patches that contain the closest-point projection and simultaneously provides a suitable initial guess for the parametric surface coordinates of each patch in the subset. Nevertheless, the second task of the narrow phase, i.e. Newton's method, is left intact. The idea behind this hybrid approach is that the broad phase and the narrow phase's first task are the most time-consuming, yet they require the least accuracy. Thus, they are ideal for replacement by the NN. On the other hand, the narrow phase's second task is relatively fast but must be performed highly accurately. Thus, this task is neither very suitable nor useful for emulation by a NN.

The NN that is developed in Chapter 3 is a multi-task NN, because it both classifies patches and approximates parametric surface coordinates. It does so simultaneously, which means that the classification part of the network and the approximation part of the network exchange information during each forward pass. Furthermore, the multi-task NN also approximates the signed distance, but its accuracy is not great. Consequently, the signed distance approximated by the NN is, therefore, only used as an initial filter, but not as the precise signed distance that is required in the governing equations.

The accuracy of the multi-task NN's parametric surface coordinate approximation, particularly near patch edges, is improved using a segmented regression layer with a customized loss function. This loss function selectively propagates gradients based on patch-specific conditions.

Chapter 2

Optimization solvers to treat snap-back in unilateral contact simulations

2.1 Introduction

The current chapter investigates the capabilities of optimization solvers. The optimization solvers are only used to treat time increments for which Newton's method does not converge. In the contact simulations of this thesis, if Newton's method does not converge, it is generally caused by snap-back. In some cases, snap-back may be perceived as a side effect of the node-to-segment contact framework. In other cases, it is visually confirmed that snap-back is supposed to occur and is completely unrelated to the employed contact framework.

The thesis limits itself to 2D and 3D scenarios in which a deformable body comes in and out of frictionless contact with a rigid body. The rigid body consistently serves as the leader-body and the deformable body functions as the follower-body. The deformable body is discretized using bilinear quadrilateral finite elements (FEs) in the 2D scenarios and trilinear hexagonal FEs in the 3D scenarios. The rigid leader-surface is smoothed using Gregory patches [60].

Non-penetration is enforced using the penalty method. Three variants are investigated for Newton's method. After a brief investigation of the results section of this chapter, the bilateral variant with a constant penalty parameter is deemed the most robust and is used for the remainder of the simulations. In the bilateral variant, Newton's method keeps applying the contact constraints until it has converged - even if a follower-vertex is temporarily out of the rigid leader-body. After each time increment has converged, the algorithm checks if the active set of follower-vertices must be updated or not. If an update indeed takes place, the time increment is repeated. In the optimization algorithms, the unilateral variant with a constant penalty stiffness is employed because the optimization algorithms are more robust than Newton's method (yet slower).

Both a hyperelastic and a hyperelastoplastic constitutive law are used to describe the mechanical deformation of the deformable body. Since the hyperelastic model employed is the same as the hyperelastoplastic model used, but with a substantially large initial yield stress, only the elastoplastic one is described in the next section.

The four optimization algorithms under investigation are the quasi-Newton method with the BFGS update of the approximation of the inverted Hessian, its limited-storage sibling (i.e. L-BFGS), the TR method exploiting the original Steihaug-Point conjugate gradient method and finally the TR method that used the same but precondition conjugate gradient method. The optimization algorithms are explained in the third section of this chapter.

The outline for the remainder of the chapter is as follows. The next section discusses the unilateral contact simulation, with Newton's method as the solution algorithm. Section 3 provides the objective function for the mechanical contact simulations and discusses the minimisation algorithms that are applied in case Newton's method fails. Section 4 briefly discusses the smoothing of surfaces using Gregory patches. The numerical investigation in section 5 elaborately compares the capabilities and performances of the different optimization solvers. Finally, a short conclusion is provided in section 6. First, a part of the mathematical notation is summarized in Table. 2.1.

Table 2.1: Some of the mathematical nomenclature.

a	scalar	$\underline{\mathbf{a}}$	column of vectors
$\mathbf{a}$	vector	$\underline{\underline{\mathbf{A}}}$	matrix of tensors
$\mathbf{A}$	2nd-order tensor	$(\bullet)^c$	conjugate transpose of 2nd-order tensor
$\underline{a}$	column	$(\bullet)^T$	transpose of matrix
$\underline{\underline{A}}$	matrix	$(\bullet)^{-1}$	inverse of matrix or 2nd-order tensor
$\mathbf{A}^{-c}$	conjugate transpose of the inverse of 2nd-order tensor (which equals the inverse of the conjugate transpose)		
$(\bullet)^{c_{ab}}$	conjugate transpose of higher-order tensor where the a^{th} and b^{th} base vector change sequence		
$(\bullet)^{cT}$	transposed of a matrix of 2nd-order tensors, where each tensor is conjugate-transposed		
$\underline{\underline{\mathbf{A}}}^{-cT}$	defined according to $\underline{\underline{\mathbf{A}}}^{-cT} \cdot \underline{\underline{\mathbf{A}}}^{cT} = \underline{\underline{\mathbf{I}}}$, where $\underline{\underline{\mathbf{I}}}$ denotes a diagonal matrix with identity tensors on its diagonal		
$\underline{b} = \frac{\mathrm{d}\,a(\underline{x})}{\mathrm{d}\,\underline{x}}$	the component of $\underline{b}$ at row i is defined as $\frac{\mathrm{d}\,a(\underline{x})}{\mathrm{d}\,(\underline{x})_i}$, where i denotes the component of $\underline{x}$ at location i.		
$\underline{\underline{B}} = \frac{\mathrm{d}^2 a(\underline{x},\underline{y})}{\mathrm{d}\underline{y}\,\mathrm{d}\underline{x}}$	the component of $\underline{\underline{B}}$ at row i, column j is defined as $\frac{\mathrm{d}}{\mathrm{d}(\underline{y})_j}\frac{\mathrm{d}a(\underline{x},\underline{y})}{\mathrm{d}(\underline{x})_i}$.		

2.2 Displacement driven Newton's method

The current section discusses the most frequently employed solver in computational mechanics: displacement-driven Newton's method. The displacement-driven nature entails that Newton's method computes a correction to the previous estimate of the displacement vectors. Subsequently, additional computations may need to be performed to update other variables, that directly or indirectly depend on the displacement vectors. In the case of quasi-static simulations involving frictionless, penalty-based, node-to-segment contact between a rate-independent elastoplastic follower-body and a rigid leader-body, these additional variables constitute (i) the parametric leader-surface coordinates associated with the shortest distance between a follower-vertex and the leader-surface, and (ii) the internal (plastic) variables of the elastoplastic constitutive model.

The first subsection discusses the governing equations which involve the signed distance quantification, the constitutive model and the discretized weak form. The second subsection discusses Newton's method to solve the governing equations and pays attention to the return mapping algorithm and the consistent tangent stiffness.

2.2.1 Governing equations

The goal in the simulations of interest is to compute the unknown displacement vector field, $\mathbf{u}$, the unknown field of the internal (plastic) variables, $\boldsymbol{\zeta}$, the parametric surface coordinates on the leader-surface associated with the closest distance between the follower-surface and the leader-surface, $\underline{\xi}$ and the field of normal pressure, p, such that the following strong form holds for the deformable follower-body:

$$\begin{aligned}
\nabla \cdot \boldsymbol{\sigma}(\mathbf{u}^*, \boldsymbol{\zeta}^*) &= \mathbf{0} && \text{in} \quad \Omega, && (2.1)\\
\boldsymbol{\sigma}(\mathbf{u}^*, \boldsymbol{\zeta}^*) - \boldsymbol{\sigma}^c(\mathbf{u}^*, \boldsymbol{\zeta}^*) &= \mathbf{0} && \text{in} \quad \Omega, && (2.2)\\
\frac{1}{J(\mathbf{u}^*)} \Big(P_{\text{mat}}(\mathbf{u}^*, \boldsymbol{\zeta}^*) - \dot{\mathcal{E}}(\mathbf{u}^*, \boldsymbol{\zeta}^*) - \dot{\mathcal{D}}(\boldsymbol{\zeta}^*) \Big) &= \mathbf{0} && \text{in} \quad \Omega, && (2.3)\\
\mathbf{u}^* &= \bar{\mathbf{u}} && \text{on} \quad \Gamma_{\text{D}}, && (2.4)\\
\boldsymbol{\sigma}(\mathbf{u}^*, \boldsymbol{\zeta}^*) \cdot \mathbf{n}(\mathbf{u}^*) &= \bar{\mathbf{t}} && \text{on} \quad \Gamma_{\text{N}}, && (2.5)\\
\boldsymbol{\sigma}(\mathbf{u}^*, \boldsymbol{\zeta}^*) \cdot \mathbf{n}(\mathbf{u}^*) - p^* \, \mathbf{n}(\mathbf{u}^*) &= \mathbf{0} && \text{on} \quad \Gamma_{\text{C}}, && (2.6)\\
g(\mathbf{u}^*, \underline{\xi}^*(\mathbf{u}^*)) \geq 0, \quad p^* \leq 0, \quad g(\mathbf{u}^*, \underline{\xi}^*(\mathbf{u})^*) \, p^* &= 0 && \text{on} \quad \Gamma_{\text{C}}. && (2.7)
\end{aligned}$$

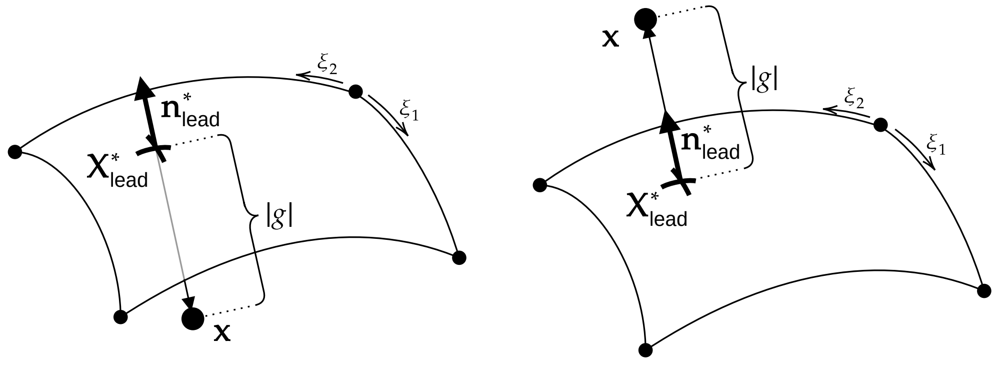
Eq. (2.1) describes the balance of linear momentum in the absence of body forces within the deformable body in the deformed configuration, Ω. The balance must be iteratively solved. ∇, $\boldsymbol{\sigma}$, and $(\cdot)$ denote the gradient operator with respect to the deformed configuration, the Cauchy stress tensor and the inner product, respectively. Asterisks are used as superscripts when the associated variables satisfy some computational problem. Thus, here asterisks are used because the associated variables satisfy the strong form.

Eq. (2.2) describes the balance of angular momentum. The constitutive model is formulated such that it automatically holds locally.

Eq. (2.3) describes energy conservation. The constitutive model is formulated such that it automatically holds locally. P_{mat}, $\mathcal{E}$ and $\mathcal{D}$ denote the internal power, the stored energy and the dissipated energy in the reference (i.e. original or undeformed) configuration, respectively. The dot on top of a symbol denotes the time derivative. J denotes the volume change with respect to the undeformed configuration as a ratio.

Eq. (2.4) specifies Dirichlet boundary conditions $\bar{\mathbf{u}}$ on surface Γ_{D} of the deformable body. Eq. (2.5) specifies Neumann boundary conditions on surface Γ_{N} using surface tractions $\bar{\mathbf{t}}$ and follower-surface's outward pointing normal vector $\mathbf{n}$. Both are defined in the deformed configuration.

Eqs. (2.6) and (2.7) together describe the contact interactions, i.e. the Signorini problem.



ξ_2
ξ_1
$\mathbf{n}^*_{\text{lead}}$
$\mathrm{X}^*_{\text{lead}}$
$|g|$
$\mathbf{x}$
$\mathbf{x}$
$|g|$
ξ_2
ξ_1
$\mathbf{n}^*_{\text{lead}}$
$\mathrm{X}^*_{\text{lead}}$

Figure 2.1: Illustration of the quantities involved in the computation of the signed distance. Left: penetration, i.e. $g < 0$. Right: non-penetration, i.e. $g \geq 0$.

The conditions in Eq. (2.7) are commonly known as the Karush-Kuhn-Tucker conditions. Eqs. (2.6) and (2.7) must be incorporated in the computational procedure that solves the balance of linear momentum. $\underline{\xi}^*$ denotes the column of length two that stores the parametric surface coordinates on the leader-body associated with the shortest distance between the leader-surface and the follower-surface. g denotes the penetration of the deformable body in the rigid body. p^* denotes the normal pressure needed to prevent penetration of the bodies.

One may recognise that the three aforementioned parts of the domain boundary of Ω, (Γ_{D}, Γ_{N} and Γ_{C}) cannot overlap. Furthermore, displacement vector field $\mathbf{u}$ is a function of the location in the undeformed domain, i.e. $\mathbf{u}(\mathbf{X})$.

Closest-point projection

In the strong form, signed distance g and parametric surface coordinates $\underline{\xi}$ are introduced. The formulation of g and the computation to find the parametric surface coordinates that minimize the distance between a point on the follower-surface and the rigid leader-surface are now discussed.

For a point on the follower-surface, indicated by $\mathbf{X}$ in the undeformed configuration, signed distance g in the strong form is quantified as follows:

$$g(\mathbf{u}, \underline{\xi}^*(\mathbf{u})) = (\mathbf{x}(\mathbf{u}) - \mathbf{X}^*_{\text{lead}}) \cdot \mathbf{n}^*_{\text{lead}}, \tag{2.8}$$

with

$$\mathbf{x}(\mathbf{u}) = \mathbf{X} + \mathbf{u}, \tag{2.9}$$

where $\mathbf{X}^*_{\text{lead}}$ denotes the point on the leader-surface that is closest to $\mathbf{x}$ (see Fig. 2.1). Furthermore, $\mathbf{n}^*_{\text{lead}}$ denotes the normal outward-pointing vector of the leader-surface at $\mathbf{X}^*_{\text{lead}}$.

In this case, the asterisks are associated with the solution of the following minimization problem:

$$\underline{\xi}^* = \underset{\underline{\xi}}{\operatorname{argmin}} \, \|\mathbf{x}(\mathbf{u}) - \mathbf{X}_{\text{lead}}(\underline{\xi})\|, \tag{2.10}$$

(where $\| \bullet \|$ denotes the L^2-norm) and consequently, they implicitly depend on displacement vector $\mathbf{u}$ (at $\mathbf{X}$):

$$\mathbf{X}^*_{\text{lead}} = \mathbf{X}_{\text{lead}}(\underline{\xi}^*(\mathbf{u})), \tag{2.11}$$

$$\mathbf{n}^*_{\text{lead}} = \mathbf{n}_{\text{lead}}(\underline{\xi}^*(\mathbf{u})). \tag{2.12}$$

Constitutive model

In the strong form, the expressions of the stresses and how the stresses, together with the history of the deformation, govern the evolution of the internal variables are not provided. These points are defined by the constitutive model. The constitutive model is now discussed. It is important to acknowledge that the constitutive model is defined such that the balance of angular momentum of Eq. (2.2) holds locally, as well as the energy balance of Eq. (2.3).

The total deformation gradient tensor is locally defined in terms of the displacement vector according to:

$$\mathbf{F}(\mathbf{u}(\mathbf{X})) = \mathbf{I} + (\nabla_0 \mathbf{u}(\mathbf{X}))^c, \tag{2.13}$$

where $\mathbf{I}$ denotes the identity tensor and ∇_0 denotes the gradient operator with respect to the reference configuration.

A multiplicative decomposition of the total deformation gradient tensor into elastic deformation gradient tensor $\mathbf{F}_\text{e}$ and plastic deformation gradient tensor $\mathbf{F}_\text{p}$ is furthermore assumed:

$$\mathbf{F}(\mathbf{u}(\mathbf{X})) = \mathbf{F}_\text{e} \cdot \mathbf{F}_\text{p}. \tag{2.14}$$

The following expression for the strain energy density is used:

$$W(\mathbf{F}_\text{e}) = \mu_1 \big(\text{tr}(\mathbf{F}^c_\text{e} \cdot \mathbf{F}_\text{e}) - 3 - \mu_2 \ln(J_\text{e})\big) + \mu_2 \big(\ln(J_\text{e})\big)^2, \tag{2.15}$$

with

$$\operatorname{tr}(\mathbf{F}_\mathrm{e}^c \cdot \mathbf{F}_\mathrm{e}) = \mathbf{I} : (\mathbf{F}_\mathrm{e}^c \cdot \mathbf{F}_\mathrm{e}), \tag{2.16}$$

$$J_\mathrm{e} = \det(\mathbf{F}_\mathrm{e}), \tag{2.17}$$

$$\mu_1 = \frac{E}{4(1+\nu)}, \tag{2.18}$$

$$\mu_2 = \frac{E\nu}{2(1+\nu)(1-2\nu)}, \tag{2.19}$$

where (:) denotes a double inner product and E and ν denote Young's modulus and Poisson's ratio, respectively. This strain energy density yields the following expression for the first Piola-Kirchoff stress tensor:

$$\mathbf{P} = \frac{\mathrm{d}\,W}{\mathrm{d}\,\mathbf{F}_\mathrm{e}} = 2\mu_1 \mathbf{F}_\mathrm{e} + 2(\mu_2 \ln(J_\mathrm{e}) - \mu_1)\mathbf{F}_\mathrm{e}^{-c}. \tag{2.20}$$

The employed Von Mises yield function reads:

$$y = \sqrt{\frac{3}{2}\mathbf{M}_\mathrm{dev} : \mathbf{M}_\mathrm{dev}} - M_\mathrm{y0} - h_\mathrm{hard}\lambda^{m_\mathrm{hard}}, \tag{2.21}$$

where M_y0, h_hard and m_hard denote the initial yield stress, the hardening modulus and the hardening exponent, respectively. Furthermore, λ denotes the plastic multiplier and deviatoric Mandel stress tensor $\mathbf{M}_\mathrm{dev}$ reads:

$$\mathbf{M}_\mathrm{dev} = \mathbf{M} - \frac{\operatorname{tr}(\mathbf{M})}{3}\mathbf{I}, \tag{2.22}$$

where the Mandel stress tensor is expressed as follows:

$$\mathbf{M} = \mathbf{F}_\mathrm{e}^c \cdot \mathbf{P}, \tag{2.23}$$

which can be shown to be symmetric with the use of Eq. (2.20).

The associative flow rule used here reads:

$$\dot{\mathbf{F}}_\mathrm{p} = \dot{\lambda}\frac{\mathrm{d}\,y}{\mathrm{d}\,\mathbf{M}} \cdot \mathbf{F}_\mathrm{p}, \tag{2.24}$$

which ignores plastic spin. Finally, the following Karush-Kuhn-Tucker relations are in place:

$$y \leq 0, \qquad \dot{\lambda} \geq 0, \qquad y\dot{\lambda} = 0. \tag{2.25}$$

Thus, the internal variables are entirely local, and they only depend on the deformation at $\mathbf{X}$. Locally, the internal variables consist of the nine components of the plastic deformation gradient tensor $\mathbf{F}_\mathrm{p}$ and the plastic multiplier λ. Furthermore, the rate of the plastic deformation gradient tensor is isochoric due to the use of the deviatoric Mandel stress tensor in the yield function. Consequently, all volume changes are due to elastic deformation:

$$\begin{aligned} \det\left(\mathbf{F}(\mathbf{u})\right) &= \det(\mathbf{F}_\mathrm{e})\det(\mathbf{F}_\mathrm{p}), && (2.26)\\ J(\mathbf{u}) &= J_\mathrm{e}\,J_\mathrm{p}(\mathbf{F}_\mathrm{p}), && (2.27)\\ &= J_\mathrm{e}. && (2.28) \end{aligned}$$

Energy balance

Next, it will be demonstrated that the constitutive model automatically satisfies the energy balance of Eq. (2.3). For this purpose, first, the internal power for an infinitesimal material point in the reference configuration can be written as follows:

$$\begin{aligned} P_\mathrm{mat} &= J\,\boldsymbol{\sigma} : \mathbf{L} && (2.29)\\ &= J_\mathrm{e}\,\boldsymbol{\sigma} : \left(\dot{\mathbf{F}}\cdot\mathbf{F}^{-1}\right), && (2.30)\\ &= \left(J_\mathrm{e}\,\boldsymbol{\sigma}\cdot\mathbf{F}^{-c}\right) : \dot{\mathbf{F}}^{c}, && (2.31)\\ &= \left(J_\mathrm{e}\,\boldsymbol{\sigma}\cdot\mathbf{F}_\mathrm{e}^{-c}\cdot\mathbf{F}_\mathrm{p}^{-c}\right) : \dot{\mathbf{F}}^{c}, && (2.32) \end{aligned}$$

On the other hand, free energy function $\mathcal{E}$ is postulated to depend on the elastic deformation gradient tensor and the plastic multiplier so that the rate of the free energy reads:

$$\begin{aligned} \dot{\mathcal{E}} &= \frac{\mathrm{d}\,\mathcal{E}}{\mathrm{d}\,\mathbf{F}_\mathrm{e}} : \dot{\mathbf{F}}_\mathrm{e}^{c} + \frac{\mathrm{d}\,\mathcal{E}}{\mathrm{d}\,\lambda}\,\dot{\lambda}, && (2.33)\\ &= \frac{\mathrm{d}\,\mathcal{E}}{\mathrm{d}\,\mathbf{F}_\mathrm{e}} : \left(\mathbf{F}_\mathrm{p}^{-c}\cdot\dot{\mathbf{F}}^{c}\right) + \frac{\mathrm{d}\,\mathcal{E}}{\mathrm{d}\,\mathbf{F}_\mathrm{e}} : \left(\dot{\mathbf{F}}_\mathrm{p}^{-c}\cdot\mathbf{F}^{c}\right) + \frac{\mathrm{d}\,\mathcal{E}}{\mathrm{d}\,\lambda}\,\dot{\lambda}, && (2.34)\\ &= \left(\frac{\mathrm{d}\,\mathcal{E}}{\mathrm{d}\,\mathbf{F}_\mathrm{e}}\cdot\mathbf{F}_\mathrm{p}^{-c}\right) : \dot{\mathbf{F}}^{c} - \frac{\mathrm{d}\,\mathcal{E}}{\mathrm{d}\,\mathbf{F}_\mathrm{e}} : \left((\mathbf{F}_\mathrm{p}^{-c}\mathbf{F}_\mathrm{p}^{-c})^{c24} : \dot{\mathbf{F}}_\mathrm{p}^{c}\cdot\mathbf{F}^{c}\right) + \frac{\mathrm{d}\,\mathcal{E}}{\mathrm{d}\,\lambda}\,\dot{\lambda}, && (2.35)\\ &= \left(\frac{\mathrm{d}\,\mathcal{E}}{\mathrm{d}\,\mathbf{F}_\mathrm{e}}\cdot\mathbf{F}_\mathrm{p}^{-c}\right) : \dot{\mathbf{F}}^{c} - \left(\mathbf{F}_\mathrm{e}^{c}\cdot\frac{\mathrm{d}\,\mathcal{E}}{\mathrm{d}\,\mathbf{F}_\mathrm{e}}\cdot\mathbf{F}_\mathrm{p}^{-c}\right) : \dot{\mathbf{F}}_\mathrm{p}^{c} + \frac{\mathrm{d}\,\mathcal{E}}{\mathrm{d}\,\lambda}\,\dot{\lambda}, && (2.36)\\ &= \left(\frac{\mathrm{d}\,\mathcal{E}}{\mathrm{d}\,\mathbf{F}_\mathrm{e}}\cdot\mathbf{F}_\mathrm{p}^{-c}\right) : \dot{\mathbf{F}}^{c} - \left(\mathbf{F}_\mathrm{e}^{c}\cdot\frac{\mathrm{d}\,\mathcal{E}}{\mathrm{d}\,\mathbf{F}_\mathrm{e}}\cdot\mathbf{F}_\mathrm{p}^{-c}\right) : \left(\mathbf{F}_\mathrm{p}^{c}\cdot\frac{\mathrm{d}\,y}{\mathrm{d}\,\mathbf{M}}\right)\dot{\lambda} + \frac{\mathrm{d}\,\mathcal{E}}{\mathrm{d}\,\lambda}\,\dot{\lambda}, && (2.37)\\ &= \left(\frac{\mathrm{d}\,\mathcal{E}}{\mathrm{d}\,\mathbf{F}_\mathrm{e}}\cdot\mathbf{F}_\mathrm{p}^{-c}\right) : \dot{\mathbf{F}}^{c} + \left(\frac{\mathrm{d}\,\mathcal{E}}{\mathrm{d}\,\lambda} - \mathbf{F}_\mathrm{e}^{c}\cdot\frac{\mathrm{d}\,\mathcal{E}}{\mathrm{d}\,\mathbf{F}_\mathrm{e}} : \frac{\mathrm{d}\,y}{\mathrm{d}\,\mathbf{M}}\right)\dot{\lambda}. && (2.38) \end{aligned}$$

Using this result together with Eq. (2.32), the rate of dissipation reads:

$$\dot{\mathcal{D}} = P_{\mathrm{mat}} - \dot{\mathcal{E}}, \tag{2.39}$$

$$= \left(J_{\mathrm{e}}\, \boldsymbol{\sigma} \cdot \mathbf{F}_{\mathrm{e}}^{-c} \cdot \mathbf{F}_{\mathrm{p}}^{-c} - \frac{\mathrm{d}\,\mathcal{E}}{\mathrm{d}\,\mathbf{F}_{\mathrm{e}}} \cdot \mathbf{F}_{\mathrm{p}}^{-c} \right) : \dot{\mathbf{F}}^{c} + \left(\mathbf{F}_{\mathrm{e}}^{c} \cdot \frac{\mathrm{d}\,\mathcal{E}}{\mathrm{d}\,\mathbf{F}_{\mathrm{e}}} : \frac{\mathrm{d}\,y}{\mathrm{d}\,\mathbf{M}} - \frac{\mathrm{d}\,\mathcal{E}}{\mathrm{d}\,\lambda} \right) \dot{\lambda}. \tag{2.40}$$

With the following postulate:

$$\frac{\mathrm{d}\,\mathcal{E}}{\mathrm{d}\,\mathbf{F}_{\mathrm{e}}} = J_{\mathrm{e}}\, \boldsymbol{\sigma} \cdot \mathbf{F}_{\mathrm{e}}^{-c}, \tag{2.41}$$

and assuming that:

$$\frac{\mathrm{d}\,\mathcal{E}}{\mathrm{d}\,\mathbf{F}_{\mathrm{e}}} = \frac{\mathrm{d}\,W}{\mathrm{d}\,\mathbf{F}_{\mathrm{e}}} \; (= \mathbf{P}), \tag{2.42}$$

the dissipation rate reduces to:

$$\dot{\mathcal{D}} = \left(\mathbf{M} : \frac{\mathrm{d}\,y}{\mathrm{d}\,\mathbf{M}} - \frac{\mathrm{d}\,\mathcal{E}}{\mathrm{d}\,\lambda} \right) \dot{\lambda}. \tag{2.43}$$

This can further be reduced using:

$$\frac{\mathrm{d}\,y}{\mathrm{d}\,\mathbf{M}} = \frac{3\,\mathbf{M}_{\mathrm{dev}}}{\sqrt{6\,\mathbf{M}_{\mathrm{dev}} : \mathbf{M}_{\mathrm{dev}}}}, \tag{2.44}$$

to:

$$\dot{\mathcal{D}} = \left(\sqrt{\frac{3}{2} \mathbf{M}_{\mathrm{dev}} : \mathbf{M}_{\mathrm{dev}}} - \frac{\mathrm{d}\,\mathcal{E}}{\mathrm{d}\,\lambda} \right) \dot{\lambda}. \tag{2.45}$$

If it is further assumed that:

$$\frac{\mathrm{d}\,\mathcal{E}}{\mathrm{d}\,\lambda} = h_{\mathrm{hard}} \lambda^{m_{\mathrm{hard}}}, \tag{2.46}$$

the dissipation rate can be ultimately reduced to (with the use of the yield criterion):

$$\dot{\mathcal{D}} = M_{\mathrm{y0}}\, \dot{\lambda}. \tag{2.47}$$

This expression entails that energy is only dissipated if plastic deformation evolves. Furthermore, because the plastic multiplier can only grow (as already included in the Karush-Kuhn-Tucker conditions of Eq. (2.25)), the dissipation rate is nonnegative, as physically required.

The only remaining issue is to define free energy function $\mathcal{E}$ in such a way that Eqs. (2.42) and (2.46) hold. The following expression accomplishes this:

$$\mathcal{E}(\mathbf{F}_{\mathrm{e}}, \lambda) = W(\mathbf{F}_{\mathrm{e}}) + \frac{h_{\mathrm{hard}}}{m_{\mathrm{hard}} + 1} \lambda^{\, m_{\mathrm{hard}}+1}, \tag{2.48}$$

where the strain energy density W is defined in Eq. (2.15). Finally, it may be noted that not all the energy stored in $\mathcal{E}$ is recoverable. Only the energy in the strain energy density function can be recovered.

The fact that this constitutive model also maximizes the dissipation (i.e. it abides by the second law of thermodynamics) is not further detailed. However, it is worth mentioning that the second law of thermodynamics is not a prerequisite for switching from a Newton solver to a minimization solver.

Balance of angular momentum

In addition, it can be shown that this constitutive model abides by the balance of angular momentum of Eq. (2.2). To this purpose, Eqs. (2.41), (2.42) and (2.20) together yield the following expression for the Cauchy stress tensor:

$$\begin{aligned} \boldsymbol{\sigma} &= \frac{1}{J_{\mathrm{e}}} \mathbf{P} \cdot \mathbf{F}_{\mathrm{e}}^{c}, && (2.49) \\ &= \frac{1}{J_{\mathrm{e}}} \Big(2\mu_1 \mathbf{F}_{\mathrm{e}} \cdot \mathbf{F}_{\mathrm{e}}^{c} + 2\big(\mu_2 \ln(J_{\mathrm{e}}) - \mu_1\big)\mathbf{I} \Big), && (2.50) \end{aligned}$$

which is indeed symmetric.

Weak form

Since the constitutive model intrinsically accounts for the balance of angular momentum of Eq. (2.2) and the energy balance of Eq. (2.3), they do not have to be solved for. In order to solve the remaining equations of the strong form, the continuity requirements of the strong form are reduced by transforming them into the weak form.

By application of the weighted residuals method to the balance of linear momentum of the strong form and subsequently, the divergence theorem to incorporate Neumann boundary conditions and the normal pressure on the potential contact surface, the strong form yields the following weak form:

$$\begin{aligned} \int_{\Omega} (\nabla \boldsymbol{\phi})^{c} : \boldsymbol{\sigma}(\mathbf{u}^*, \boldsymbol{\zeta}^*) \,\mathrm{d}V - \int_{\Gamma_{\mathrm{N}}} \boldsymbol{\phi} \cdot \bar{\mathbf{t}} \,\mathrm{d}A - \int_{\Gamma_{\mathrm{C}}} \boldsymbol{\phi} \cdot p^* \mathbf{n}^* \,\mathrm{d}A = 0, && (2.51) \\ g(\mathbf{u}^*, \underline{\xi}^*(\mathbf{u}^*)) \geq 0, \quad p^* \leq 0, \quad g(\mathbf{u}^*, \underline{\xi}^*(\mathbf{u})^*)\, p^* = 0, && (2.52) \end{aligned}$$

where $\boldsymbol{\phi}(\mathbf{X})$ denotes some test vector field defined in the reference configuration. Here, the Dirichlet boundary conditions are ignored in the notation.

As this thesis uses the penalty method to enforce contact constraints, normal pressure p^* is replaced by $-k\langle -g\rangle$, where k denotes the penalty stiffness and $\langle \bullet \rangle$ the Macaulay brackets. Incorporating this in the weak form, together with the fact that the outward-pointing normal of the deformable follower-body, $\mathbf{n}$, and the outward pointing normal of the rigid leader-body, $\mathbf{n}_{\text{lead}}$, are opposites:

$$\mathbf{n} = -\mathbf{n}_{\text{lead}}, \tag{2.53}$$

yields the following result:

$$\int_{\Omega} (\nabla\boldsymbol{\phi})^c : \boldsymbol{\sigma}(\mathbf{u}^*, \boldsymbol{\zeta}^*)\,\mathrm{d}V - \int_{\Gamma_{\mathrm{N}}} \boldsymbol{\phi}\cdot\bar{\mathbf{t}}\,\mathrm{d}A - \int_{\Gamma_{\mathrm{C}}} \boldsymbol{\phi}\cdot k\big\langle -g(\mathbf{u}^*, \underline{\xi}^*(\mathbf{u}^*))\big\rangle \mathbf{n}^*_{\text{lead}}\,\mathrm{d}A = 0, \tag{2.54}$$

which avoids the need of Eq. (2.52).

The issue with this weak form (and with the strong form for that matter) is that externally applied traction $\bar{\mathbf{t}}$ in the deformed configuration is hard to apply, but its equivalent in the reference configuration, $\bar{\mathbf{t}}_0$, is straightforward to apply. Furthermore, the contribution due to the balance of linear momentum is also more straightforward to express in the reference contribution, but not the contribution due to contact. For this reason, using the following two relations:

$$\nabla = \mathbf{F}^{-c}\cdot\nabla_0, \tag{2.55}$$

$$\mathrm{d}V = J\,\mathrm{d}V_0 = J_{\mathrm{e}}\,\mathrm{d}V_0, \tag{2.56}$$

(where ∇_0 and $\mathrm{d}V_0$ denote the gradient operator with respect to the original configuration and the integral over the original configuration, respectively), the weak form is partially expressed in the reference configuration and partially in the deformed configuration as follows:

$$\int_{\Omega_0} (\nabla_0\boldsymbol{\phi})^c : \left(\mathbf{F}_{\mathrm{p}}^{-1}\cdot\mathbf{P}^c(\mathbf{u}^*, \boldsymbol{\zeta}^*)\right)\,\mathrm{d}V_0 - \int_{\Gamma_{\mathrm{N}0}} \boldsymbol{\phi}\cdot\bar{\mathbf{t}}_0\,\mathrm{d}A_0 - \int_{\Gamma_{\mathrm{C}}} \boldsymbol{\phi}\cdot k\big\langle -g(\mathbf{u}^*, \underline{\xi}^*(\mathbf{u}^*))\big\rangle \mathbf{n}^*_{\text{lead}}\,\mathrm{d}A = 0. \tag{2.57}$$

Discretized governing equations

By spatially discretizing the weak form with the same test functions as the trial functions (here with trilinear hexagonal FEs) and numerically integrating it (here with eight Gauss quadrature points per hexagonal FE), a system of non-linear vector equations is obtained which, in case it holds, can be written as follows:

$$\underline{\mathbf{f}}(\underline{\mathbf{u}}^*, \underline{\underline{\xi}}^*, \underline{\underline{\zeta}}^*) = \underline{\mathbf{0}}, \tag{2.58}$$

where $\underline{\mathbf{f}}$ and $\underline{\mathbf{u}}$ denote the columns containing the force vector that acts on each FE vertex and the displacement vector of each FE vertex, respectively. The matrix $\underline{\underline{\xi}}$ stores the leader-body's parametric surface coordinates associated with the shortest distance of each follower-vertex. It is thus of size $2 \times n_{\text{con}}$ for 3D simulations (where n_{con} denotes the number of follower-vertices in contact). The matrix $\underline{\underline{\zeta}}$ stores the internal variables. For the constitutive model of interest in 3D, the matrix with internal variables contains the nine components of the plastic deformation gradient tensor and the plastic multiplier for each quadrature point. Thus, it is of size $10 \times 8n_{\text{el}}$, where n_{el} denotes the number of FEs.

The column $\underline{\mathbf{f}}$ is additively constructed based on the three integrals of the weak form as follows:

$$\underline{\mathbf{f}}(\underline{\mathbf{u}}, \underline{\underline{\xi}}^*, \underline{\underline{\zeta}}) = \underline{\mathbf{f}}_{\text{mat}}(\underline{\mathbf{u}}, \underline{\underline{\zeta}}) - \underline{\mathbf{f}}_{\text{ext}} - \underline{\mathbf{f}}_{\text{con}}(\underline{\mathbf{u}}, \underline{\underline{\xi}}^*), \tag{2.59}$$

where $\underline{\mathbf{f}}_{\text{ext}}$ denotes the column with external force vectors and:

$$\underline{\mathbf{f}}_{\text{mat}}(\underline{\mathbf{u}}, \underline{\underline{\zeta}}) = \sum_{i=1}^{8n_{\text{el}}} \eta_i \nabla_0 \underline{\mathcal{N}}_0 \cdot \left(\mathbf{F}_{\text{p}/i}^{-1} \cdot \mathbf{P}^c \left(\underline{\mathbf{u}}, \underline{\zeta}_i \right) \right), \tag{2.60}$$

$$\underline{\mathbf{f}}_{\text{con}}(\underline{\mathbf{u}}, \underline{\underline{\xi}}^*) = \sum_{i \in C} k_i \Big\langle - g_i \left(\mathbf{u}_i, \underline{\xi}_i^*(\mathbf{u}_i) \right) \Big\rangle \underline{\mathbf{n}^*_{\text{lead}/i}}, \tag{2.61}$$

where η_i denotes the weight of the quadrature point i and $\underline{\mathcal{N}}_0$ denotes the column of shape functions (where the subscript 0 is included to indicate that the shape functions are defined with respect to the undeformed configuration, i.e. $\mathbf{X}$). Moreover, the following abbreviations are used:

$$\underline{\zeta}_i = \left(\underline{\underline{\zeta}}\right)_{:,i} \qquad \underline{\xi}_i = \left(\underline{\underline{\xi}}\right)_{:,i} \qquad \mathbf{u}_i = (\underline{\mathbf{u}})_i, \tag{2.62}$$

Thus, the plastic variables of integration point i, $\underline{\zeta}_i$, constitute the plastic multiplier and the components of plastic deformation gradient tensor $\mathbf{F}_{\text{p}/i}$.

In Eq. (2.61) furthermore, C denotes the set that contains the FE follower-vertices that may be in contact. The signed distance of the follower-vertex i is indicated by g_i and is given in Eq. (2.8). It only depends on the displacement vector of follower-vertex i, i.e. $\mathbf{u}_i$. $\underline{\mathbf{n}^*_{\text{lead}/i}}$ denotes a zero column, with the exception of the component associated with the follower-vertex i according to Eq. (2.12).

2.2.2 Newton's method

Displacement-driven Newton's method uses a first-order Taylor expansion around the estimated solution of iteration it, $\underline{\mathbf{u}}^{it}$, to obtain a system of linear vector equations. Once the system of linear vector equations is transformed into a system of linear scalar equations, it can be computed numerically.

The system of linear vector equations can be expressed as follows:

$$\underline{\mathbf{f}}\big|_{\underline{\mathbf{u}}^{it},\underline{\underline{\zeta}}^{*it},\underline{\underline{\xi}}^{*it}} + \underline{\underline{\mathbf{K}}}\big|_{\underline{\mathbf{u}}^{it},\underline{\underline{\zeta}}^{*it},\underline{\underline{\xi}}^{*it}} \cdot \underline{\mathbf{h}} = \underline{\mathbf{0}}, \tag{2.63}$$

where $\underline{\mathbf{h}}$ denotes the correction to the current estimate of displacements $\underline{\mathbf{u}}^{it}$ that is to be computed. Furthermore, $\underline{\underline{\mathbf{K}}}$ denotes the Jacobian, which is a matrix of second-order tensors.

Particular in displacement-driven Newton's method is that additional variables $\underline{\underline{\zeta}}$ and $\underline{\underline{\xi}}$ are recomputed to perfectly hold for each new estimate $\underline{\mathbf{u}}^{it}$. For this reason, the asterisks are included in the superscripts of $\underline{\underline{\zeta}}$ and $\underline{\underline{\xi}}$ in Eq. (2.63). Recomputing the additional variables is relatively efficient because the computations are entirely local; the closest point projections in $\underline{\underline{\xi}}$ can be computed for each follower-vertex independently, and the computations of the new plastic variables can be computed for each integration point independently.

The computation of the closest projection point was already discussed in the text surrounding Eq. (2.10). The computation of the plastic variables is now discussed.

Return mapping

The computation of the updated internal variables starts with evaluating the yield function for the existing internal variables. In case the yield function is not exceeded in a quadrature point, the internal variables associated with the quadrature point do not change. The reason is that Karush-Kuhn-Tucker conditions in Eq. (2.25) are not violated.

However, if the yield function is exceeded at the quadrature point, the internal variables do change. In that case, the equations that need to hold in the integration point can be written as follows:

$$y(\mathbf{F}_\mathrm{p}, \dot{\lambda} \big| \mathbf{F}_\mathrm{p/prev}, \lambda_\mathrm{prev}, \mathbf{F}^{it}) = 0, \tag{2.64}$$

$$\mathbf{F}_\mathrm{p} = \exp\left(\dot{\lambda}\frac{\mathrm{d}\,y}{\mathrm{d}\,\mathbf{M}}\right) \cdot \mathbf{F}_\mathrm{p/prev}, \tag{2.65}$$

$$\dot{\lambda} > 0, \tag{2.66}$$

where $\mathbf{F}_\mathrm{p}$ and $\dot{\lambda}$ denote the variables of the problem. Furthermore, $\mathbf{F}_\mathrm{p/prev}$ and λ_prev together quantify the previous plastic state of the integration point and

$$\mathbf{F}^{it} = \mathbf{F}(\underline{\mathbf{u}}^{it}). \tag{2.67}$$

It may furthermore be clear that an eigenproblem must be solved to express:

$$\exp\left(\dot{\lambda}\frac{\mathrm{d}\,y}{\mathrm{d}\,\mathbf{M}}\right) = \sum_{i=1}^{3} \exp(\dot{\lambda}s_i)\,\mathbf{n}_i\mathbf{n}_i, \tag{2.68}$$

where s_i and $\mathbf{n}_i$ denote the i^th eigenvalue and eigenvector of tensor $\frac{\mathrm{d}\,y}{\mathrm{d}\,\mathbf{M}}$, respectively.

The said return mapping problem can be simplified using the concept of 'closest point projection', which in the case of Von Mises yield criterion leads to the radial return algorithm. The idea is to insert Eq. (2.65) in Eq. (2.64) so that the new plastic deformation gradient tensor is omitted as a variable and the only equation to solve is:

$$y(\dot{\lambda}|\mathbf{F}_{\mathrm{p/prev}}, \lambda_{\mathrm{prev}}, \mathbf{F}^{it}) = 0. \tag{2.69}$$

Newton's method is then applied as follows:

$$y(\dot{\lambda}^{it_{\mathrm{RM}}}) + \left.\frac{\mathrm{d}\,y}{\mathrm{d}\,\dot{\lambda}}\right|_{\dot{\lambda}^{it_{\mathrm{RM}}}} \Delta\dot{\lambda} = 0, \tag{2.70}$$

where $\dot{\lambda}^{it_{\mathrm{RM}}}$ denotes the previous estimate of the change of the plastic multiplier and $\Delta\dot{\lambda}$ the correction of the previous estimate of the change of the plastic multiplier that is to be computed in the current iteration. Furthermore, the derivative in the previous equation reads:

$$\frac{\mathrm{d}\,y}{\mathrm{d}\,\dot{\lambda}} = \frac{\mathrm{d}\,y}{\mathrm{d}\,\mathbf{M}} : \frac{\mathrm{d}\,\mathbf{M}^c}{\mathrm{d}\,\mathbf{F}_\mathrm{e}} : \frac{\mathrm{d}\,\mathbf{F}_\mathrm{e}^c}{\mathrm{d}\,\mathbf{F}_\mathrm{p}} : \frac{\mathrm{d}\,\mathbf{F}_\mathrm{p}^c}{\mathrm{d}\,\dot{\lambda}} - m_{\mathrm{hard}} h_{\mathrm{hard}} (\lambda_{\mathrm{prev}} + \dot{\lambda})^{m_{\mathrm{hard}}-1}, \tag{2.71}$$

where

$$\frac{\mathrm{d}\,y}{\mathrm{d}\,\mathbf{M}} = \frac{3\,\mathbf{M}_{\mathrm{dev}}}{\sqrt{6\,\mathbf{M}_{\mathrm{dev}} : \mathbf{M}_{\mathrm{dev}}}}, \tag{2.72}$$

$$\frac{\mathrm{d}\,\mathbf{M}^c}{\mathrm{d}\,\mathbf{F}_\mathrm{e}} = (\mathbf{IP})^{c_{14}} + \left(\mathbf{F}_\mathrm{e}^c \cdot \left(\frac{\mathrm{d}\,\mathbf{P}^c}{\mathrm{d}\,\mathbf{F}_\mathrm{e}}\right)^{c_{12}}\right)^{c_{12}}, \tag{2.73}$$

$$\frac{\mathrm{d}\,\mathbf{F}_\mathrm{e}^c}{\mathrm{d}\,\mathbf{F}_\mathrm{p}} = -\left(\left(\mathbf{F}_\mathrm{p}^{-c}\,\mathbf{F}^{it} \cdot \mathbf{F}_\mathbf{p}^{-1}\right)^{c_{23}}\right)^{c_{34}}, \tag{2.74}$$

$$\frac{\mathrm{d}\,\mathbf{F}_\mathrm{p}^c}{\mathrm{d}\,\dot{\lambda}} = \mathbf{F}_{\mathrm{p/prev}}^c \cdot \sum_{i=1}^{3} s_i \exp(\dot{\lambda} s_i)\, \mathbf{n}_i \mathbf{n}_i, \tag{2.75}$$

with

$$\frac{\mathrm{d}\,\mathbf{P}^c}{\mathrm{d}\,\mathbf{F}_\mathrm{e}} = 2\mu_1 (\mathbf{II})^{c_{23}} + 2\mu_2 \mathbf{F}_\mathrm{e}^{-1} \mathbf{F}_\mathrm{e}^{-c} + 2\left(\mu_1 - \mu_2 \ln(J_\mathrm{e})\right) \left(\mathbf{F}_\mathrm{e}^{-1} \mathbf{F}_\mathrm{e}^{-c}\right)^{c_{23}}. \tag{2.76}$$

Important to realize is that the return mapping problem may be so non-linear that Newton's method, which is used to solve Eq. (2.70), does not converge. Here, this is monitored by comparing the previous residual of the return mapping algorithm with the new residual. If the new residual is larger than the previous residual, cut-backs are initiated. For the first cut-back, this entails that the return mapping algorithm is attempted for the average between the new deformation gradient tensor $\mathbf{F}^{it}$ and the converged one of the previous time increment. If successful, the return mapping algorithm is repeated for the new deformation gradient tensor $\mathbf{F}^{it}$. If unsuccessful, another cut-back is made.

Jacobian

Next, attention is paid to the Jacobian in Eq. (2.63). The Jacobian constitutes of two contributions; one due to contact and one due to the constitutive behaviour, i.e. the internal forces:

$$\underline{\underline{\mathbf{K}}} = \frac{\mathrm{d}\,\underline{\mathbf{f}}}{\mathrm{d}\,\underline{\mathbf{u}}} = \frac{\mathrm{d}\underline{\mathbf{f}}_{\mathrm{mat}}}{\mathrm{d}\underline{\mathbf{u}}} - \frac{\mathrm{d}\underline{\mathbf{f}}_{\mathrm{con}}}{\mathrm{d}\underline{\mathbf{u}}}. \tag{2.77}$$

First, the part of the Jacobian due to contact is considered. It can be expressed as follows:

$$\frac{\mathrm{d}\underline{\mathbf{f}}_{\mathrm{con}}}{\mathrm{d}\underline{\mathbf{u}}} = \sum_{i \in C} -k_i \mathrm{H}(-g_i) \frac{\mathrm{d}\,g_i}{\mathrm{d}\,\underline{\mathbf{u}}} \left(\underline{\mathbf{n}^*_{\mathrm{lead}/i}}\right)^T + k_i \Big\langle -g_i \Big\rangle \frac{\mathrm{d}\,\underline{\underline{\mathbf{n}^*_{\mathrm{lead}/i}}}}{\mathrm{d}\,\underline{\mathbf{u}}}, \tag{2.78}$$

where H(•) denotes the Heaviside function, g_i is abbreviated but must be considered as in Eq. (2.61) and the two derivatives with respect to $\underline{\mathbf{u}}$ need to be discussed in more detail.

The first derivative will now be considered. For this purpose, the variation of g_i, i.e. δg_i, can be written as follows:

$$\delta g_i = \delta \mathbf{x}_i(\mathbf{u}_i) \cdot \mathbf{n}^*_{\mathrm{lead}/i} - \delta \mathbf{X}^*_{\mathrm{lead}/i} \cdot \mathbf{n}^*_{\mathrm{lead}/i} + \Big(\mathbf{x}_i(\mathbf{u}_i) - \mathbf{X}^*_{\mathrm{lead}/i}\Big) \cdot \delta \mathbf{n}^*_{\mathrm{lead}/i}, \tag{2.79}$$

with the help of Eq. (2.8). This exercise makes clear that, because of the following orthogonalities:

$$\delta \mathbf{X}^*_{\mathrm{lead}/i} \quad \perp \quad \mathbf{n}^*_{\mathrm{lead}/i}, \tag{2.80}$$

$$\mathbf{x}_i - \mathbf{X}^*_{\mathrm{lead}/i} \quad \perp \quad \delta \mathbf{n}^*_{\mathrm{lead}/i}, \tag{2.81}$$

the first partial derivative in Eq. (2.78) can be reduced to:

$$\frac{\mathrm{d}\,g_i}{\mathrm{d}\,\underline{\mathbf{u}}} = \frac{\mathrm{d}\,\mathbf{x}_i(\mathbf{u}_i)}{\mathrm{d}\,\underline{\mathbf{u}}} \cdot \mathbf{n}^*_{\mathrm{lead}/i} = \underline{\mathbf{I}}_i \cdot \mathbf{n}^*_{\mathrm{lead}/i} = \underline{\mathbf{n}^*_{\mathrm{lead}/i}}, \tag{2.82}$$

where $\underline{\mathbf{I}}_i$ denotes a zero column with the exception of component i, which is equal to the identity tensor. Thus, Eq. (2.78) can now be expressed as follows:

$$\frac{\mathrm{d}\underline{\mathbf{f}}_{\mathrm{con}}}{\mathrm{d}\underline{\mathbf{u}}} = \sum_{i \in C} -k_i \mathrm{H}(-g_i) \underline{\underline{\mathbf{n}^*_{\mathrm{lead}/i} \mathbf{n}^*_{\mathrm{lead}/i}}} + k_i \Big\langle -g_i \Big\rangle \frac{\mathrm{d}\,\underline{\underline{\mathbf{n}^*_{\mathrm{lead}/i}}}}{\mathrm{d}\,\underline{\mathbf{u}}}, \tag{2.83}$$

where each component of the matrix $\underline{\underline{\mathbf{n}^*_{\mathrm{lead}/i} \mathbf{n}^*_{\mathrm{lead}/i}}}$ is zero, except for the one associated with row i, column i.

Next, the remaining partial derivative in Eq. (2.83) is considered. It can be expressed as follows:

$$\frac{\mathrm{d}\,\underline{\mathbf{n}}^*_{\mathrm{lead}/i}}{\mathrm{d}\,\underline{\mathbf{u}}} = \frac{\mathrm{d}\,\underline{\mathbf{n}}^*_{\mathrm{lead}/i}}{\mathrm{d}\,\underline{\xi}_i}\frac{\mathrm{d}\,\underline{\xi}^*_i}{\mathrm{d}\,\mathbf{u}_i} \cdot \left(\frac{\mathrm{d}\,\mathbf{u}_i}{\mathrm{d}\,\underline{\mathbf{u}}}\right)^T , \tag{2.84}$$

$$= \frac{\mathrm{d}\,\underline{\mathbf{n}}^*_{\mathrm{lead}/i}}{\mathrm{d}\,\underline{\xi}_i}\frac{\mathrm{d}\,\underline{\xi}^*_i}{\mathrm{d}\,\mathbf{u}_i} \cdot (\underline{\mathbf{I}}_i)^T , \tag{2.85}$$

with

$$\frac{\mathrm{d}\underline{\xi}^*_i}{\mathrm{d}\,\mathbf{u}_i} = -\left(\left.\frac{\partial^2\omega}{\partial(\underline{\xi}_i)^2}\right|_{\underline{\xi}^*_i}\right)^{-1} \left.\frac{\partial^2\omega}{\partial\mathbf{u}_i\,\partial\underline{\xi}_i}\right|_{\underline{\xi}^*_i} , \tag{2.86}$$

which is obtained by applying the implicit function theorem to the objective function in Eq. (2.10), where the objective function of Eq. (2.10) is denoted by ω, i.e.:

$$\omega(\underline{\xi}_i \,|\, \mathbf{u}_i) = \|\mathbf{x}_i(\mathbf{u}_i) - \mathbf{X}_{\mathrm{lead}/i}(\underline{\xi}_i)\|. \tag{2.87}$$

Next, the part of the Jacobian (Eq. (2.77)) associated with the internal forces is considered. It can be expressed as follows:

$$\frac{\mathrm{d}\underline{\mathbf{f}}_{\mathrm{mat}}}{\mathrm{d}\underline{\mathbf{u}}} = \sum_{i=1}^{8n_{\mathrm{el}}} \eta_i \nabla_0 \underline{\mathcal{N}}_0 \cdot {}^4\mathbf{K}_{\mathrm{mat}/i} \cdot \nabla_0 \underline{\mathcal{N}}_0^T , \tag{2.88}$$

where ${}^4\mathbf{K}_{\mathrm{mat}/i}$ denotes the consistent tangent (stiffness) tensor associated with the integration point i. It depends on the displacements, as well as on the current plastic variables and the previous plastic multiplier of the integration point i.

In case the material of the integration point i does not exceed the yield criterion for the current estimate of the displacement, i.e. $\underline{\mathbf{u}}^{it}$, the consistent tangent tensor reads:

$${}^4\mathbf{K}_{\mathrm{mat}} = \mathbf{F}^{-1}_{\mathrm{p/prev}} \cdot \frac{\mathrm{d}\,\mathbf{P}^c}{\mathrm{d}\,\mathbf{F}_{\mathrm{e}}} \cdot \mathbf{F}^{-c}_{\mathrm{p/prev}}. \tag{2.89}$$

where subscript i is ignored for ease of notation. It may be noted that in this case, the elastic deformation gradient tensor in the expressions of the fourth-order stiffness tensor $\frac{\mathrm{d}\,\mathbf{P}^c}{\mathrm{d}\,\mathbf{F}_{\mathrm{e}}}$ in Eq. (2.76) reads:

$$\mathbf{F}_{\mathrm{e}} = \mathbf{F}^{-1}_{\mathrm{p/prev}} \cdot \mathbf{F}^{it}. \tag{2.90}$$

However, if the material at the integration point i exceeds the yield criterion for the current estimate of displacements $\underline{\mathbf{u}}^{it}$, the consistent tangent tensor reads:

$$
\begin{aligned}
{}^4\mathbf{K}_{\text{mat}} = \mathbf{F}_{\text{p}}^{-1} \cdot \frac{\mathrm{d}\,\mathbf{P}^c}{\mathrm{d}\,\mathbf{F}_{\text{e}}} \cdot \mathbf{F}_{\text{p}}^{-c} + \\
\left(\mathbf{F}_{\text{p}}^{-1} \cdot \frac{\mathrm{d}\,\mathbf{P}^c}{\mathrm{d}\,\mathbf{F}_{\text{e}}} : \frac{\mathrm{d}\,\mathbf{F}_{\text{e}}^c}{\mathrm{d}\,\mathbf{F}_{\text{p}}} - \left(\mathbf{F}_{\text{p}}^{-1}\mathbf{P} \cdot \mathbf{F}_{\text{p}}^{-c}\right)^{c23} \right) : \frac{\mathrm{d}\,\mathbf{F}_{\text{p}}^c}{\mathrm{d}\,\mathbf{F}},
\end{aligned} \tag{2.91}
$$

with

$$
\frac{\mathrm{d}\,\mathbf{F}_{\text{p}}^c}{\mathrm{d}\,\mathbf{F}} = \frac{\mathrm{d}\,\mathbf{F}_{\text{p}}^c}{\mathrm{d}\,\dot{\lambda}} \frac{\mathrm{d}\,\dot{\lambda}}{\mathrm{d}\,\mathbf{F}}, \tag{2.92}
$$

where, thanks to the implicit function theorem applied to the yield criterion:

$$
\frac{\mathrm{d}\,\dot{\lambda}}{\mathrm{d}\,\mathbf{F}} = -\left(\frac{\mathrm{d}\,y}{\mathrm{d}\,\dot{\lambda}}\right)^{-1} \frac{\mathrm{d}\,y}{\mathrm{d}\,\mathbf{F}}, \tag{2.93}
$$

with:

$$
\frac{\mathrm{d}\,y}{\mathrm{d}\,\mathbf{F}} = \frac{\mathrm{d}\,y}{\mathrm{d}\,\mathbf{M}} : \frac{\mathrm{d}\,\mathbf{M}^c}{\mathrm{d}\,\mathbf{F}_{\text{e}}} : \frac{\mathrm{d}\,\mathbf{F}_{\text{e}}^c}{\mathrm{d}\,\mathbf{F}}, \tag{2.94}
$$

$$
= \frac{\mathrm{d}\,y}{\mathrm{d}\,\mathbf{M}} : \frac{\mathrm{d}\,\mathbf{M}^c}{\mathrm{d}\,\mathbf{F}_{\text{e}}} : \left(\mathbf{I}\mathbf{F}_{\text{p}}^{-c}\right)^{c13}. \tag{2.95}
$$

In this case of yielding, the elastic deformation gradient tensor in all of the said second-order and fourth-order tensors reads:

$$
\mathbf{F}_{\text{e}} = \mathbf{F}_{\text{p}}^{-1} \cdot \mathbf{F}^{it}. \tag{2.96}
$$

2.2.3 Contact variants

The way in which contact is included in the method described so far is truly unilateral. This means that the contact constraints are treated as true inequality constraints. Hence, each follower-vertex may penetrate the leader-surface in one iteration but not in the next. Consequently, the application of the contact force and stiffness may also change from one iteration to the next - as indicated by the Macaulay brackets and the Heaviside function in Eqs. (2.61) and (2.83).

The issue with this unilateral way of applying the penalty constraints is that Newton's method may fail to converge. This may occur, for instance, if the penalty stiffness is too large, in which case the contact status of follower-vertices keeps changing from one iteration to the next. On the other hand, if the penalty stiffness is too small, the amount of penetration may be so large that the simulation loses its physical meaning. A sensitivity study of the stiffness in the standard unilateral case as well as the variants described in this section, can be found in Appendix A.

Adaptive penalty stiffness

To avoid this problem, contact simulations using the penalty method may adaptively change the penalty stiffness during the course of a simulation. Normally, a relatively small penalty stiffness k^0 is used at the start of the simulation. If the amount of penetration after convergence of a time increment exceeds some threshold, g_{max}, the penalty stiffness is increased and the time increment is repeated. On the other hand, if the amount of penetration after convergence of a time increment is smaller than some threshold, g_{min}, the penalty stiffness is reduced and the time increment is repeated.

In this thesis, the following updating rule is used to adapt the penalty stiffness of follower-vertex i:

$$k_i = \begin{cases} k_i & \text{if} \quad g_{\mathrm{min}} \leq \langle -g_i^* \rangle \leq g_{\mathrm{max}}, \\ (1+\kappa)\frac{\langle -g_i^* \rangle}{g_{\mathrm{max}}} k_i & \text{if} \quad \langle -g_i^* \rangle > g_{\mathrm{max}}, \\ \max\left((1-\kappa)\frac{\langle -g_i^* \rangle}{g_{\mathrm{min}}} k_i, k^0\right) & \text{if} \quad \langle -g_i^* \rangle < g_{\mathrm{min}}, \end{cases} \tag{2.97}$$

where $\kappa > 0$ denotes a user-defined value, and the following abbreviation is used:

$$g_i^* = g_i\left(\mathbf{u}_i^*, \underline{\xi}_i^*(\mathbf{u}_i^*)\right). \tag{2.98}$$

Updating the penalty stiffness during the course of a simulation is also not free of trouble. The reason is that a too large value of κ may cause the penetration that is obtained after the convergence of the repeated time increment to skip the domain $[g_{\mathrm{min}}, g_{\mathrm{max}}]$ entirely. Consequently, increments may be repeated often before a viable domain is reached, if it is reached at all. Another potential consequence of selecting a too-large value of κ is that Newton's method does not converge at all because the new penalty stiffness is simply too large. Alternatively, selecting a small value for κ means that the same time increment must be repeated often before the penetration reaches the viable domain.

Active set method

As an alternative to considering the contact constraints as inequality constraints in combination with adjustable penalty stiffnesses, the contact constraints may also be considered as equality constraints in combination with a relatively large, constant penalty stiffness. This entails that before Newton's method is applied, one has to decide which follower-vertices are in contact and which are not. During the course of the iterative process of Newton's method, the follower-vertices selected to be in contact are forced to the leader-surface, regardless of whether or not they penetrate the leader-surface. In such an active set approach, one has to assess whether or not the follower-vertices assumed to be in contact were indeed the correct ones. This assessment is made after Newton's method has converged. If a mistake was made, the increment must be repeated with a different set of follower-vertices in contact, i.e. the active set. This can be easily performed by considering the sign of the signed distance function g_i of each follower-vertex. Fig. 2.2 illustrates this process.

In the active set method, the expressions for the contact forces and the associated part of the Jacobian slightly change. Instead of Eq. (2.61), the expression for the contact forces

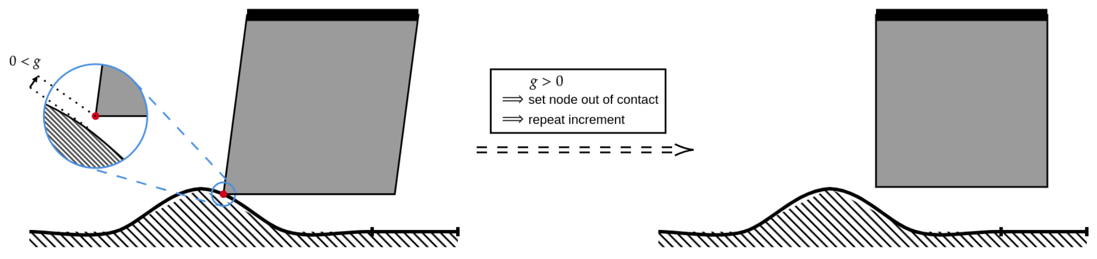


Figure 2.2: The active set method. Left: the red vertex is assumed to be in contact at the start of the current time increment and treated with an equality constraint during the iterative process. After convergence is reached, the sign of signed distance g indicates that the red vertex should not have been in contact. Consequently, the increment is repeated with the status of the red vertex changed from 'active' to 'inactive', yielding the image on the right.

reads:

$$\underline{\mathbf{f}}_{\text{con}}(\underline{\mathbf{u}}, \underline{\underline{\xi}}^*) = -k \sum_{i \in \text{act}} g_i \underline{\mathbf{n}^*_{\text{lead}/i}}, \tag{2.99}$$

where C_{act} denotes the active set of follower-vertices (i.e. $C_{\text{act}} \subseteq C$, cf. Eqs. (2.61) and (2.83)). Instead of Eq. (2.83), the part of the Jacobian due to contact now reads:

$$\frac{\mathrm{d}\underline{\mathbf{f}}_{\text{con}}}{\mathrm{d}\underline{\mathbf{u}}} = -k \sum_{i \in C_{\text{act}}} \underline{\underline{\mathbf{n}^*_{\text{lead}/i}\mathbf{n}^*_{\text{lead}/i}}} + g_i \frac{\mathrm{d}\,\underline{\mathbf{n}}^*_{\text{lead}/i}}{\mathrm{d}\,\underline{\mathbf{u}}}, \tag{2.100}$$

In addition, it can be observed that each penalty stiffness is the same in the active set approach of this thesis (cf. k versus k_i in Eqs. (2.61) and (2.83)).

2.3 Minimisation

The current section demonstrates that the mechanical model of the previous section can be formulated as a true minimization problem, and hence, true minimization solvers can be exploited to solve it. Subsequently, the current section discusses the two minimization solvers that are utilized.

Of the three contact formulations introduced for Newton's method - 1) unilateral constraints with constant penalty stiffness, 2) unilateral contact constraints with adaptive penalty stiffness, and 3) active set approach with constant penalty stiffness - only the first one is considered for the minimization approaches. The reason is that the second and third variants were considered for Newton's method to treat its relatively fragile robustness, while the discussed minimization approaches are substantially more robust.

2.3.1 Objective function

The current subsection provides the objective function of the aforementioned mechanical problem and demonstrates that the column with force vectors in Eq. (2.59) is, in fact, the gradient of this objective function.

According to true incremental variational elastoplasticity (see e.g. [90, 91, 92, 52]), the objective function of the aforementioned mechanical problem, discretized in space and time, could be written as follows:

$$\underline{\mathbf{u}}^*, \underline{\underline{\zeta}}^* = \operatorname*{argmin}_{\underline{\mathbf{u}}, \underline{\underline{\zeta}}} \quad \Pi_{\text{mat}}(\underline{\mathbf{u}}, \underline{\underline{\zeta}} \,|\, \underline{\underline{\zeta}}_{\text{prev}}) - \Pi_{\text{con}}(\underline{\mathbf{u}}, \underline{\underline{\xi}}^*(\underline{\mathbf{u}})) - \underline{\mathbf{f}}_{\text{ext}}^T \cdot \underline{\mathbf{u}}, \tag{2.101}$$

with

$$\underline{\xi}_i^*(\mathbf{u}_i) = \operatorname*{argmin}_{\underline{\xi}_i} \quad \omega(\underline{\xi}_i \,|\, \mathbf{u}_i) \qquad \forall \quad i \in C, \tag{2.102}$$

where Π_{mat} denotes the contribution to the objective function due to the material behaviour and Π_{con} denotes the contribution to the objective function due to contact. It may be noted that Neumann and Dirichlet boundary conditions are again not distinguished to ease the notation.

However, the objective function of Eq. (2.101) is not required for standard simulations that are driven by displacement. Instead, the minimization problem exploited here is formulated as follows:

$$\underline{\mathbf{u}}^* = \operatorname*{argmin}_{\underline{\mathbf{u}}} \quad \Pi_{\text{mat/u}}(\underline{\mathbf{u}} \,|\, \underline{\underline{\zeta}}_{\text{prev}}, \underline{\underline{\zeta}}^*) - \Pi_{\text{con}}(\underline{\mathbf{u}}, \underline{\underline{\xi}}^*(\underline{\mathbf{u}})) - \underline{\mathbf{f}}_{\text{ext}}^T \cdot \underline{\mathbf{u}}, \tag{2.103}$$

such that

$$\underline{\xi}_i^*(\mathbf{u}_i) = \operatorname*{argmin}_{\underline{\xi}_i} \quad \omega(\underline{\xi}_i \,|\, \mathbf{u}_i) \qquad \forall \quad i \in C, \tag{2.104}$$

$$y\left(\underline{\zeta}_i^* \,\middle|\, \underline{\zeta}_{\text{prev}/i}, \underline{\mathbf{u}}\right) = 0 \qquad \forall \quad i \in \{1, 2, ..., 8n_{\text{el}}\}, \tag{2.105}$$

where Eq. (2.105) denotes the return mapping problem. $\Pi_{\text{mat/u}}$ denotes the contribution due to the constitutive behaviour and is formulated in such a way that it requires the exact plastic variables. This is in contrast to Π_{mat} in Eq. (2.101), which also considers the plastic variables as primary variables of the optimization problem. Thus, Eq. (2.103) requires that every time displacement column $\underline{\mathbf{u}}$ changes, the plastic variables are recomputed exactly - just as in the conventional Newton's method.

The contribution due to the constitutive behaviour in Eq. (2.103) is for instance shown in [51] to read:

$$\Pi_{\mathrm{mat/u}} = \sum_{i=1}^{8n_{\mathrm{el}}} \eta_i \left(W(\mathbf{F}_{\mathrm{e}/i}) + M_{\mathrm{y0}}\dot{\lambda}_i + \frac{h}{m+1}(\lambda_{\mathrm{p/prev}/i} + \dot{\lambda}_i)^{m+1} \right), \tag{2.106}$$

where the elastic deformation gradient tensor $\mathbf{F}_{\mathrm{e}}$ must be taken according to Eq. (2.90) if plastic yielding does not occur (in which case $\dot{\lambda}_i = 0$) and according to Eq. (2.96) in case of plastic yielding.

As a consequence of Eq. (2.106), the constitutive behaviour's contribution to the gradient reads:

$$\begin{aligned}\frac{\mathrm{d}\,\Pi_{\mathrm{mat/u}}}{\mathrm{d}\,\underline{\mathbf{u}}} &= \sum_{i=1}^{8n_{\mathrm{el}}} \eta_i \frac{\mathrm{d}\,W}{\mathrm{d}\,\mathbf{F}_{\mathrm{e}/i}} : \frac{\mathrm{d}\,\mathbf{F}^c_{\mathrm{e}/i}}{\mathrm{d}\,\mathbf{F}_i} : \frac{\mathrm{d}\,\mathbf{F}^c_i}{\mathrm{d}\,\underline{\mathbf{u}}}, & (2.107)\\ &= \sum_{i=1}^{8n_{\mathrm{el}}} \eta_i \nabla_0 \underline{\mathcal{N}}_0 \cdot \left(\bar{\mathbf{F}}^{-1}_{\mathrm{p}/i} \cdot \mathbf{P}^c \left(\underline{\mathbf{u}}, \underline{\zeta}^*_i \right) \right), & (2.108)\end{aligned}$$

with

$$\bar{\mathbf{F}}_{\mathrm{p}/i} = \begin{cases} \mathbf{F}_{\mathrm{p/prev}/i} & \text{in case of purely elastic deformation,} \\ \mathbf{F}^*_{\mathrm{p}/i} & \text{in case of plastic yielding.} \end{cases} \tag{2.109}$$

Hence, the contribution of the constitutive behaviour to the gradient of Eq. (2.108) is the same as the internal forces of Eq. (2.60).

In Eq. (2.103), the contribution due to contact in case of inequality constraints reads:

$$\Pi_{\mathrm{con}} = -\frac{1}{2} \sum_{i \in C} k_i \left\langle -g_i \right\rangle^2, \tag{2.110}$$

where g_i is abbreviated but must be considered as in Eq. (2.61). In the case of the active set method, in which equality constraints are utilized, the contribution due to contact reads:

$$\Pi_{\mathrm{con}} = -\frac{1}{2} k \sum_{i \in C_{\mathrm{act}}} g_i^2. \tag{2.111}$$

With the help of Eq. (2.82), the contact contribution to the gradient in the case of inequality constraints is as follows:

$$\frac{\mathrm{d}\,\Pi_{\mathrm{con}}}{\mathrm{d}\,\underline{\mathbf{u}}} = \sum_{i \in C} k_i \langle -g_i \rangle \underline{\mathbf{n}^*_{\mathrm{lead}/i}}, \tag{2.112}$$

which is the same as the contact forces of Eq. (2.61). Again, with the help of Eq. (2.82), the contact contribution to the gradient in case of equality constraints as used in the active set method reads:

$$\frac{\mathrm{d}\,\Pi_{\text{con}}}{\mathrm{d}\,\underline{\mathbf{u}}} = -k \sum_{i \in C_{\text{act}}} g_i \, \underline{\mathbf{n}^*_{\text{lead}/i}}, \tag{2.113}$$

which is the same as the contact forces of Eq. (2.99).

Now, the gradient of the objective function of Eq. (2.103) is shown to be equal to the discretized force vectors of the previous section, Newton's method of the previous section can indeed be considered as a potential algorithm to solve the minimization problem of Eq. (2.103).

In the field of computational mechanics, the application of Newton's method typically hinges on the first-order Taylor expansion of the column with force vectors, as presented in Eq. (2.63). In the field of optimization, however, the application of Newton's method typically starts with a second-order Taylor expansion of the objective function:

$$\bar{\Pi}^{it}(\underline{\mathbf{h}}) = \Pi^{it} + \underline{\mathbf{h}}^T \cdot \underline{\mathbf{f}}^{it} + \frac{1}{2}\underline{\mathbf{h}}^T \cdot \underline{\underline{\mathbf{K}}}^{it} \cdot \underline{\mathbf{h}}, \tag{2.114}$$

with

$$\Pi^{it} = \Pi_{\text{mat/u}}\left(\underline{\mathbf{u}}^{it}, \underline{\underline{\zeta}}_{\text{prev}}, \underline{\underline{\zeta}}^{*it}\right) - \Pi_{\text{con}}\left(\underline{\mathbf{u}}^{it}, \underline{\underline{\zeta}}^{*it}(\underline{\mathbf{u}}^{it})\right) - \underline{\mathbf{f}}^T_{\text{ext}} \cdot \underline{\mathbf{u}}^{it} \tag{2.115}$$

$$\underline{\mathbf{f}}^{it} = \underline{\mathbf{f}}\big|_{\underline{\mathbf{u}}^{it}, \underline{\underline{\zeta}}^{*it}, \underline{\underline{\xi}}^{*it}} \tag{2.116}$$

$$\underline{\underline{\mathbf{K}}}^{it} = \underline{\underline{\mathbf{K}}}\big|_{\underline{\mathbf{u}}^{it}, \underline{\underline{\zeta}}^{*it}, \underline{\underline{\xi}}^{*it}}. \tag{2.117}$$

Subsequently, the assumption is made that the gradient of the approximation in Eq. (2.114) equals zero at the minimizer of the approximation in Eq. (2.114), which yields:

$$\frac{\mathrm{d}\,\bar{\Pi}^{it}}{\mathrm{d}\,\underline{\mathbf{h}}} = \underline{\mathbf{0}}, \tag{2.118}$$

$$\underline{\mathbf{f}}^{it} + \underline{\underline{\mathbf{K}}}^{it} \cdot \underline{\mathbf{h}} = \underline{\mathbf{0}}. \tag{2.119}$$

Although this expression clearly equals the same system of linear vector equations of Eq. (2.63), it is worth presenting Newton's method in the format of Eq. (2.114) because the two alternate minimization algorithms in this chapter are based on it.

2.3.2 BFGS method

The first true minimization algorithm of this chapter is the quasi-Newton method [93] (also known as [94]) with the Broyden-Fletcher-Goldfarb-Shanno (BFGS) update for the inverted Hessian [35, 36, 37, 38, 39]. Just like Newton's method, quasi-Newton methods are based on the second-order Taylor expansion of the objection function, but with one notable difference:

$$\bar{\Pi}^{it}(\underline{\mathbf{h}}) = \Pi^{it} + \underline{\mathbf{h}}^T \cdot \underline{\mathbf{f}}^{it} + \frac{1}{2}\underline{\mathbf{h}}^T \cdot \bar{\underline{\underline{\mathbf{K}}}}^{it} \cdot \underline{\mathbf{h}}, \tag{2.120}$$

which is that $\bar{\underline{\underline{\mathbf{K}}}}^{it}$ does not denote the exact Hessian, but only an approximation (hence, the tilde on top of the symbol). The approximation of the Hessian is based on results of previous iterations. Hence, the Hessian approximation incorporates more information about the curvature as the iterative process progresses.

In most quasi-Newton methods, furthermore, it is not the Hessian that is approximated, but the inverse of the Hessian. Consequently, no system of linear equations needs to be solved to determine search direction $\underline{\mathbf{h}}$ as in Newton's method (cf. Eq. (2.63)), but only a matrix-column multiplication must be performed:

$$\underline{\mathbf{h}} = -\bar{\underline{\underline{\mathbf{L}}}}^{it} \cdot \underline{\mathbf{f}}^{it}, \tag{2.121}$$

where $\bar{\underline{\underline{\mathbf{L}}}}^{it}$ denotes the approximation of the inverted Hessian of the it^{th} iteration. This accelerates the computation of the search direction in quasi-Newton methods, relative to that of Newton's method.

This thesis uses the well-known BFGS update to approximate the inverted Hessian [35, 36, 37, 38, 39]. In the mathematical notation of this chapter, the BFGS update can be expressed as follows:

$$\begin{aligned}\bar{\underline{\underline{\mathbf{L}}}}^{it} = \bar{\underline{\underline{\mathbf{L}}}}^{it-1} + \frac{(\Delta\underline{\mathbf{u}}^T \cdot \Delta\underline{\mathbf{f}} + \Delta\underline{\mathbf{f}}^T \cdot \bar{\underline{\underline{\mathbf{L}}}}^{it-1} \cdot \Delta\underline{\mathbf{f}})}{(\Delta\underline{\mathbf{u}}^T \cdot \Delta\underline{\mathbf{f}})^2} \Delta\underline{\mathbf{u}}\Delta\underline{\mathbf{u}}^T \\ - \frac{\bar{\underline{\underline{\mathbf{L}}}}^{it-1} \cdot \Delta\underline{\mathbf{f}} \cdot \Delta\underline{\mathbf{u}}^T + \Delta\underline{\mathbf{u}} \cdot \Delta\underline{\mathbf{f}}^T \cdot \bar{\underline{\underline{\mathbf{L}}}}^{it-1}}{\Delta\underline{\mathbf{u}}^T \cdot \Delta\underline{\mathbf{f}}},\end{aligned} \tag{2.122}$$

with

$$\Delta\underline{\mathbf{u}} = \underline{\mathbf{u}}^{it} - \underline{\mathbf{u}}^{it-1}, \tag{2.123}$$

$$\Delta\underline{\mathbf{f}} = \underline{\mathbf{f}}\big|_{\underline{\mathbf{u}}^{it},,\underline{\underline{\zeta}}^{*it},\underline{\underline{\xi}}^{*it}} - \underline{\mathbf{f}}\big|_{\underline{\mathbf{u}}^{it-1},\underline{\underline{\zeta}}^{*(it-1)},\underline{\underline{\xi}}^{*(it-1)}}. \tag{2.124}$$

The discussion of the mathematical foundation of the approximation is considered too lengthy for this thesis, but more information can be found in [39]. Eqs. (2.123) and (2.124) clearly show that the estimates of the minimizer and associated gradient evaluations of all previous iterations are incorporated into the current approximation of the inverted Hessian.

Wolfe conditions

Not only the BFGS method, but all quasi-Newton methods come with two conditions on the approximation of Eq. (2.120). First, the gradient of the approximation at the current estimate (i.e. $\underline{\mathbf{h}} = \underline{\mathbf{0}}$) must equal the gradient of the exact objective function. Second, the gradient of the approximation at the previous estimate (i.e. $\underline{\mathbf{h}} = -\underline{\mathbf{h}}^{it-1}$) must equal the gradient of the exact objective function. Thus, these statements read:

$$\left.\frac{\mathrm{d}\,\bar{\Pi}^{it}}{\mathrm{d}\,\underline{\mathbf{h}}}\right|_{\underline{\mathbf{h}}=\underline{\mathbf{0}}} = \underline{\mathbf{f}}^{it}, \qquad \left.\frac{\mathrm{d}\,\bar{\Pi}^{it}}{\mathrm{d}\,\underline{\mathbf{h}}}\right|_{\underline{\mathbf{h}}=-\underline{\mathbf{h}}^{it-1}} = \underline{\mathbf{f}}^{it-1}. \tag{2.125}$$

Working out the left-hand sides of these two equations and combining the results yields the so-called secant equation:

$$\bar{\underline{\underline{\mathbf{K}}}}^{it} \cdot \underline{\mathbf{h}}^{it-1} = \underline{\mathbf{f}}^{it} - \underline{\mathbf{f}}^{it-1}. \tag{2.126}$$

By increasing the iteration number by one, the next approximation of the inverted Hessian will only be invertible if:

$$\underline{\mathbf{h}}^{T} \cdot \bar{\underline{\underline{\mathbf{K}}}}^{it+1} \cdot \underline{\mathbf{h}} > 0, \tag{2.127}$$

with $\underline{\mathbf{h}} = \underline{\mathbf{h}}^{it}$. If this statement must be true, the right-hand-side of Eq. (2.126) premultiplied by search direction $\underline{\mathbf{h}}$ must also be larger than zero:

$$\underline{\mathbf{h}}^{T} \cdot \underline{\mathbf{f}}^{it+1} - \underline{\mathbf{h}}^{T} \cdot \underline{\mathbf{f}}^{it} > 0 \tag{2.128}$$

$$\underline{\mathbf{h}}^{T} \cdot \underline{\mathbf{f}}^{it+1} > \underline{\mathbf{h}}^{T} \cdot \underline{\mathbf{f}}^{it}, \tag{2.129}$$

where again the iteration number in Eq. (2.126) is increased by one.

Thus, to ensure that the Hessian approximation of the next iteration, i.e. $it + 1$, (and hence, the approximation of the inverted Hessian) is invertible, not any stepsize in search direction $\underline{\mathbf{h}}$ can be taken, but only one for which Eq. (2.129) holds. For this reason, the column with displacement vectors is not simply updated with $\underline{\mathbf{h}}$ as in Newton's method, but as follows:

$$\underline{\mathbf{u}}^{it+1} = \underline{\mathbf{u}}^{it} + \alpha\underline{\mathbf{h}}, \tag{2.130}$$

where α denotes a suitable stepsize.

In practise however, Eq. (2.129) is slightly weakened. This entails that stepsize α must be abide the so-called curvature condition:

$$\underline{\mathbf{h}}^{T} \cdot \underline{\mathbf{f}}^{it+\alpha h} > \chi_2\underline{\mathbf{h}}^{T} \cdot \underline{\mathbf{f}}^{it}, \tag{curvature}$$

where $0 < \chi_2 < 1$ and

$$\underline{\mathbf{f}}^{it+\alpha h} = \underline{\mathbf{f}}\big|_{\underline{\mathbf{u}}^{it}+\alpha\underline{\mathbf{h}},\underline{\underline{\xi}}^{*it+\alpha h},\underline{\underline{\zeta}}^{*it+\alpha h}} \tag{2.131}$$

with

$$\underline{\xi}_i^*(\mathbf{u}_i + \alpha\mathbf{h}_i) = \underset{\underline{\xi}_i}{\operatorname{argmin}} \quad \omega(\underline{\xi}_i \,|\, \mathbf{u}_i + \alpha\mathbf{h}_i) \qquad \forall \quad i \in C, \tag{2.132}$$

$$y\left(\underline{\zeta}_i^{*it+\alpha h} \,\Big|\, \underline{\zeta}_{\text{prev}/i}, \underline{\mathbf{u}}^{it} + \alpha\underline{\mathbf{h}}\right) = 0 \qquad \forall \quad i \in \{1, 2, ..., 8n_{\text{el}}\}. \tag{2.133}$$

Although the curvature condition ensures that the approximation of the inverted Hessian of the next iteration is invertible, it does not guarantee that the objective function decreases at the next estimate of the minimizer, i.e. at $\underline{\mathbf{u}}^{it} + \alpha\underline{\mathbf{h}}$. The Armijo rule is often also imposed to guarantee that the objective function is reduced at the next estimate:

$$\Pi^{it+\alpha h} < \Pi^{it} + \chi_1\alpha\underline{\mathbf{h}}^T \cdot \underline{\mathbf{f}}\big|_{\underline{\mathbf{u}}^{it}}, \tag{Armijo}$$

with Π^{it} provided in Eq. (2.115) and

$$\begin{aligned}\Pi^{it+\alpha h} = \Pi_{\text{mat/u}}\left(\underline{\mathbf{u}}^{it} + \alpha\underline{\mathbf{h}} \,\Big|\, \underline{\underline{\zeta}}_{\text{prev}}, \underline{\underline{\zeta}}^{*it+\alpha h}\right)& \\ - \Pi_{\text{con}}\left(\underline{\mathbf{u}}^{it} + \alpha\underline{\mathbf{h}}, \underline{\underline{\xi}}^{*it+\alpha h}(\underline{\mathbf{u}}^{it} + \alpha\underline{\mathbf{h}})\right) - \underline{\mathbf{f}}_{\text{ext}}^T \cdot \left(\underline{\mathbf{u}}^{it} + \alpha\underline{\mathbf{h}}\right),&\end{aligned} \tag{2.134}$$

Furthermore, $0 < \chi_1 < \chi_2 < 1$. If $\chi_1 = 0$, the Armijo rule accepts any stepsize α for which the objective function is smaller than the current one. This can yield tiny reductions of the objective function. For this reason, χ_1 is typically set to be larger than zero.

The Armijo rule and the curvature condition together are known as the Wolfe conditions. Fig. 2.3 depicts the domains in which the Armijo rule and the curvature condition are satisfied for an arbitrary objective function.

Line search

A line search algorithm that computes a stepsize that abides both Wolfe conditions must consider both the objective function (Eq. (Armijo)), as well as the gradient (Eq. (curvature)) at a proposed stepsize. The problem is that the objective function constitutes a sum over all quadrature points (see Eq. (2.106)) and in case of a large number of FEs, the objective function is polluted by noise due to the machine precision.

On the other hand, the derivative of the objective function in direction $\underline{\mathbf{h}}$ with respect to stepsize α:

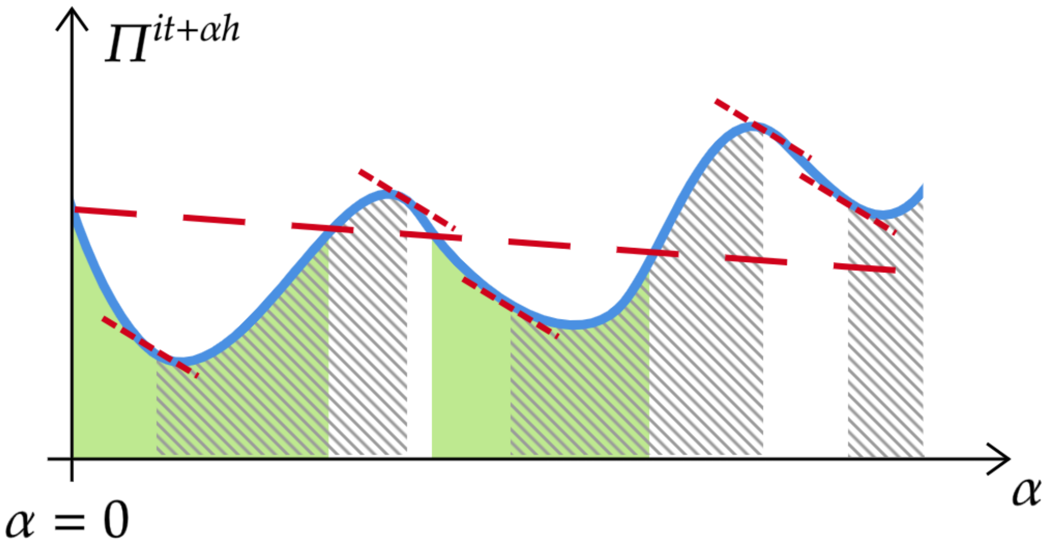


Figure 2.3: Sketch of objective function $\Pi^{it+\alpha h}$ (blue) in search direction $\underline{\mathbf{h}}$ as a function of stepsize α. The big red dashed line depicts the Armijo rule. Green domains depict domains for which the Armijo rule holds. Shaded domains depict domains for which the curvature condition holds. The bounds of the shaded domains (thin red dashed lines) are governed by the derivative of the objective function in direction $\underline{\mathbf{h}}$ with respect to α. Domains in which the green and shaded domains overlap, abide both Wolfe conditions. Hence, a stepsize in these overlapping domains would be accepted.

$$\frac{\mathrm{d}\,\Pi^{it+\alpha h}}{\mathrm{d}\,\alpha} = \underline{\mathbf{h}}^T \cdot \underline{\mathbf{f}}^{it+\alpha h}, \tag{2.135}$$

is substantially less polluted by noise due to machine precision. Consequently, the line search in this thesis only requires gradient evaluations, but no objective function evaluations.

The line search algorithm of this thesis, which only exploits the derivative of Eq. (2.135), is based on two observations. First, it can be observed in Fig. 2.3 that the first local minimum in the search direction typically abides both Wolfe conditions. Second, Fig. 2.4 shows that only one minimum occurs for the line searches encountered in the mechanical problems of this thesis (at $\frac{\mathrm{d}\,\Pi^{it+\alpha h}}{\mathrm{d}\,\alpha} = 0$). Thus, the line search of this thesis aims to find a stepsize α such that $\frac{\mathrm{d}\,\Pi^{it+\alpha h}}{\mathrm{d}\,\alpha} \approx 0$, because at this stepsize both Wolfe conditions are satisifed. The line search accomplishes this by only sampling $\frac{\mathrm{d}\,\Pi^{it+\alpha h}}{\mathrm{d}\,\alpha}$.

Fig. 2.4 shows that quite a variety of responses may occur. For instance, a linear approximation of response I would suffice to determine $\frac{\mathrm{d}\,\Pi^{it+\alpha h}}{\mathrm{d}\,\alpha} \approx 0$ (for which only one additional stepsize $0 < \alpha^2$ needs to be sampled, because $\alpha^1 = 0$ is already sampled). For response II on the other hand, a quadratic approximation would probably suffice (for which two additional stepsizes $0 < \alpha^2 < \alpha^3$ need to be sampled). A quadratic approximation would also work for response III, provided that the approximation is made in the convex domain, i.e. $0 < \alpha^1 < \alpha^2 < \alpha^3$. Finally, response IV is hard to approximate with any type

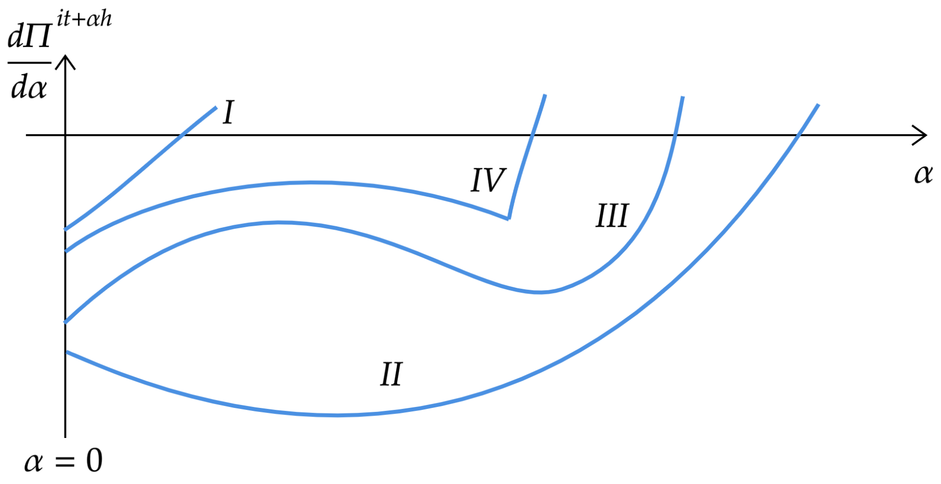


Figure 2.4: Sketch of four types of responses $\frac{\mathrm{d}\,\Pi^{it+\alpha h}}{\mathrm{d}\,\alpha}$ (blue) in search direction $\underline{\mathbf{h}}$ as a function of stepsize α that were observed in the simulations with inequality constraints. The different types of responses are indicated by Roman numerals.

of polynomial expression.

A pseudo-code of the line search algorithm of this thesis is presented in Alg. 1. In essence, it is based on a quadratic approximation (line 13) so that responses I, II and III can be treated relatively rapidly. To ensure that the quadratic approximation is only made in the convex part of responses of type III, lines 4 to 12 first increase three stepsizes α^1, α^2 and α^3 until they are located in the convex domain. If the quadratic approximation fails to yield a stepsize for which $\left|\frac{\mathrm{d}\,\Pi^{it+\alpha h}}{\mathrm{d}\,\alpha}\right| < tol_{\mathrm{ls}}$ in $maxiter_{\mathrm{ls/q}}$ iterations, a secant method (i.e. using a linear approximation) is attempted (lines 14-16). If the secant line search also fails because the number of iterations exceeds $maxiter_{\mathrm{ls/s}}$, the algorithm switches to bisectioning (lines 17-19).

A final issue worth mentioning is that the line search algorithm may suggest too exotic deformations to evaluate, i.e. characterized by $J_{\mathrm{e}} < 0$ in an integration point. In these cases, the gradient $\underline{\mathbf{f}}^{it+\alpha h}$ cannot be computed and the line search algorithm must reduce stepsize α. This particularity is not presented in Alg. 1.

Limited-storage variant

Besides the BFGS method, its limited-storage alternative is also studied (i.e. the L-BFGS method) [40, 41, 39]. The L-BFGS approach reduces the storage requirements of the BFGS method that are caused by the fact that the inverted Hessian approximation in Eq. (2.122) is a full matrix - which can, for instance, be recognized by the term $\Delta\underline{\mathbf{u}}\Delta\underline{\mathbf{u}}^T$.

To explain the workings of the L-BFGS method, the BFGS method's construction of the inverted Hessian approximation is first expanded. To this purpose, it can be recognized that the BFGS' inverted Hessian approximation of Eq. (2.122) is effectively a function of initially assumed inverted Hessian approximation $\underline{\underline{\bar{\mathbf{L}}}}^0$ and all previously recorded iterative changes of the estimated minimizers and of the associated gradients, i.e.:

Algorithm 1 Line search

```
    INPUT: $\underline{\mathbf{u}}^{it}$, $\underline{\mathbf{f}}^{it}$, $\underline{\mathbf{h}}$, $\delta\alpha$, $maxiter_{\mathrm{ls/q}}$, $maxiter_{\mathrm{ls/s}}$
 1: $\alpha^1 = 0$, $\alpha^2 = \delta\alpha$, $\alpha^3 = 2\delta\alpha$
 2: $f^0 = f^1 = \underline{\mathbf{h}}^T \cdot \underline{\mathbf{f}}^{it}$, $f^2 = \underline{\mathbf{h}}^T \cdot \underline{\mathbf{f}}^{it+\alpha^2 h}$, $f^3 = \underline{\mathbf{h}}^T \cdot \underline{\mathbf{f}}^{it+\alpha^3 h}$
 3: if $f^2 > 0$ then
 4:     $\alpha^+ = \alpha^2$, $f^+ = f^2$
 5: else if $f^3 > 0$ then
 6:     $\alpha^+ = \alpha^3$, $f^+ = f^3$
 7: else
 8:     while $f^3 < 0$ do
 9:         Increase $\alpha^1$, $\alpha^2$ and $\alpha^3$ and recompute $f^3$
10:     end while
11:     $\alpha^+ = \alpha^3$, $f^+ = f^3$
12: end if
13: Quadratic line search until $|f| < \min(10^{-15}, 10^{-7}|f^0|)$ or $iter_{\mathrm{ls/s}} > maxiter_{\mathrm{ls/q}}$. IN-
    PUT: $\alpha^1, \alpha^2, \alpha^3, f^1, f^2, f^3$. OUTPUT: $f$, $\alpha$
14: if $|f| > \min(10^{-15}, 10^{-7}|f^0|)$ then
15:     Secant line search until $|f| < \min(10^{-15}, 10^{-7}|f^0|)$ or $iter_{\mathrm{ls/s}} >$
        $maxiter_{\mathrm{ls/s}}$. INPUT: $\alpha^1, \alpha^+, f^1, f^+$. OUTPUT: $f$, $\alpha$
16: end if
17: if $|f| > \min(10^{-12}, 10^{-6}|f^0|)$ then
18:     Bisection until $|f| < \min(10^{-15}, 10^{-7}|f^0|)$. INPUT: $\alpha^1, \alpha^+, f^1, f^+$.
        OUTPUT: $f$, $\alpha$
19: end if
20: $\underline{\mathbf{u}}^{it+1} = \underline{\mathbf{u}}^{it} + \alpha\underline{\mathbf{h}}$, $\underline{\mathbf{f}}^{it+1} = \underline{\mathbf{f}}^{it+\alpha h}$
    OUTPUT: $\underline{\mathbf{u}}^{it+1}$, $\underline{\mathbf{f}}^{it+1}$
```

Algorithm 2 L-BFGS method's search direction computation

```
    INPUT: $\underline{\underline{\bar{\mathbf{L}}}}^0$, $n_{\mathrm{li}}$, $\underline{\mathbf{f}}^{it}$, $\underline{\underline{\Delta\mathbf{u}}} = [\Delta\underline{\mathbf{u}}^{it-n_{\mathrm{li}}}, \Delta\underline{\mathbf{u}}^{it-n_{\mathrm{li}}+1}, ..., \Delta\underline{\mathbf{u}}^{it-1}]$,
           $\underline{\underline{\Delta\mathbf{f}}} = [\Delta\underline{\mathbf{f}}^{it-n_{\mathrm{li}}}, \Delta\underline{\mathbf{f}}^{it-n_{\mathrm{li}}+1}, ..., \Delta\underline{\mathbf{f}}^{it-1}]$
 1: $\underline{\Theta} = \mathrm{zeros}(n_{\mathrm{li}}, 1)$
 2: $\underline{\mathbf{h}} = \underline{\mathbf{f}}^{it}$
 3: for $i = n_{\mathrm{li}} : -1 : \max(1, n_{\mathrm{li}} - it + 2)$ do
 4:     $\Delta\underline{\mathbf{u}} = \underline{\underline{\Delta\mathbf{u}}}(:, i)$, $\Delta\underline{\mathbf{f}} = \underline{\underline{\Delta\mathbf{f}}}(:, i)$
 5:     $\theta = \frac{\Delta\underline{\mathbf{u}}^T \cdot \underline{\mathbf{h}}}{\Delta\underline{\mathbf{u}}^T \cdot \Delta\underline{\mathbf{f}}}$, $\underline{\Theta}(i) = \theta$
 6:     $\underline{\mathbf{h}} = \underline{\mathbf{h}} - \theta\Delta\underline{\mathbf{f}}$
 7: end for
 8: $\underline{\mathbf{h}} = -\underline{\mathbf{h}}$
 9: $\underline{\mathbf{h}} = \underline{\underline{\bar{\mathbf{L}}}}^0 \cdot \underline{\mathbf{h}}$
10: for $i = \max(1, n_{\mathrm{li}} - it + 2) : 1 : n_{\mathrm{li}}$ do
11:     $\Delta\underline{\mathbf{u}} = \underline{\underline{\Delta\mathbf{u}}}(:, i)$, $\Delta\underline{\mathbf{f}} = \underline{\underline{\Delta\mathbf{f}}}(:, i)$, $\theta = \underline{\Theta}(i)$
12:     $\underline{\mathbf{h}} = \underline{\mathbf{h}} + \left(\theta - \frac{\Delta\underline{\mathbf{f}}^T \cdot \underline{\mathbf{h}}}{\Delta\underline{\mathbf{u}}^T \cdot \Delta\underline{\mathbf{f}}}\right) \Delta\underline{\mathbf{u}}$
13: end for
    OUTPUT: $\underline{\mathbf{h}}$
```

$$\underline{\underline{\bar{\mathbf{L}}}}^{it} = \mathcal{F}(\underline{\underline{\bar{\mathbf{L}}}}^{0}, \Delta\underline{\mathbf{u}}^{1}, \Delta\underline{\mathbf{f}}^{1}, \Delta\underline{\mathbf{u}}^{2}, \Delta\underline{\mathbf{f}}^{2}, ..., \Delta\underline{\mathbf{u}}^{it-1}, \Delta\underline{\mathbf{f}}^{it-1}), \tag{2.136}$$

where $\mathcal{F}(\bullet)$ denotes some function and the superscripts denote the iteration numbers.

In contrast to the BFGS method, the L-BFGS method does not use the iterative changes of all previous iterations, but only those of the n_{li} most recent iterations. Thus, in the general case that $it > n_{\text{li}}$, the L-BFGS method writes the inverted Hessian approximation as follows (cf. Eq. (2.136)):

$$\underline{\underline{\bar{\mathbf{L}}}}^{it} = \mathcal{F}(\underline{\underline{\bar{\mathbf{L}}}}^{0}, \Delta\underline{\mathbf{u}}^{it-n_{\text{li}}}, \Delta\underline{\mathbf{f}}^{it-n_{\text{li}}}, \Delta\underline{\mathbf{u}}^{it-n_{\text{li}}+1}, \Delta\underline{\mathbf{f}}^{it-n_{\text{li}}+1}, ..., \Delta\underline{\mathbf{u}}^{it-1}, \Delta\underline{\mathbf{f}}^{it-1}). \tag{2.137}$$

The crucial difference with the BFGS approach is that the L-BFGS method does not explicitly compute, nor store the inverted Hessian approximation. Instead, it directly computes search direction $\underline{\mathbf{h}}$ according to the matrix-column product of Eq. (2.121) in terms of the intially assumed inverted Hessian approximation and the iterative changes of the estimated minimizer and of the associated gradient of the last n_{li} iterations. This can be performed relatively fast because the results of not all previous iterations are considered, but only those of the last n_{li} iterations. Consequently, only $2n_{\text{it}}n_{\text{var}}$ scalars need to be stored for the L-BFGS method, whereas n_{var}^2 must be stored for the BFGS method (where n_{var} denotes the number of unknowns in $\underline{\mathbf{u}}$).

Computing search direction $\underline{\mathbf{h}}$ according to Eq. (2.121) directly in terms of the initially assumed inverted Hessian approximation and the iterative changes of the minimizer and of the associated gradient of the last n_{li} iterations is achieved here by means of Alg. 2 [95]. The implementation of Alg. 2 is fairly simple to verify, because the predictions of the BFGS method and the L-BFGS method should be the same for the first n_{li} iterations.

2.3.3 Trust region method

The second true minimization algorithm investigated in this chapter is the trust region (TR) method [42, 43, 44, 39]. Whereas minimization methods such as the non-linear conjugate gradient method [96, 97] and quasi-Newton methods first compute a search direction and then calculate the stepsize, TR methods work the other way around. Hence, TR methods first need a (maximal) stepsize, and only then will they predict the correction to the current estimate of the minimizer.

The correction to the current estimate is computed by minimizing a quadratic approximation of the objective function, whilst safeguarding that the norm of the correction does not exceed the maximal stepsize. The maximal stepsize quantifies the size of the region in which the quadratic approximation can be trusted to be a suitable description of the exact objective function.

Depending on the obtained reduction of the objective function at the newly proposed estimate, the maximal stepsize for the next iteration is updated. The pink dash-dotted circles in Fig. 2.5 illustrate how the size of the trust region changes from one iteration to the next. The reduction obtained also determines whether the correction should or should not be used to update the current estimate of the minimizer.

To better explain TR methods, the computation of the correction to the current estimate is first considered. This computation is called the TR subproblem. Subsequently, the updating of the maximal stepsize for the next iteration is considered as well as whether or not the correction will be used to update the current estimate.

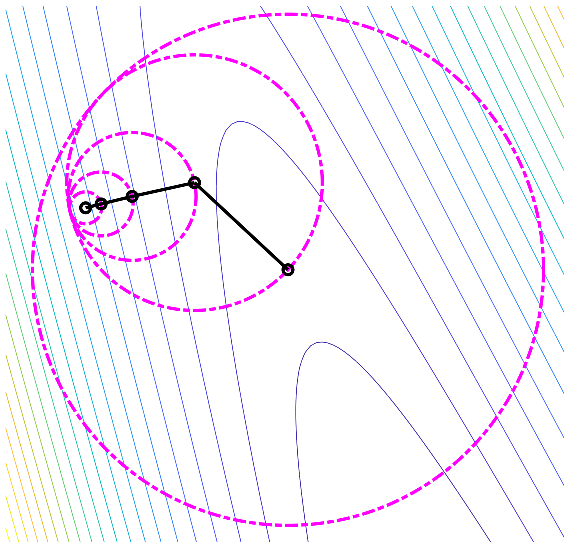


Figure 2.5: TR methods consider a trust region that is updated from one iteration to the next. The trust regions of different iterations are indicated by the pink dash-dotted circles. The black dots represent the estimate of the minimizer for different iterations. The colored lines depict isolines of an arbitrary objective function.

Trust region subproblem

The computation of correction $\underline{\mathbf{h}}^*$ to the current estimate of the minimizer, $\underline{\mathbf{u}}^{it}$, is based on the second-order Taylor expansion of the objective function, just like in Newton's method and quasi-Newton methods (cf. Eqs. (2.114) and (2.120)). Hence, TR methods write the computation of correction $\underline{\mathbf{h}}^*$ for given maximal stepsize R_{TR} as the following optimization problem:

$$\underline{\mathbf{h}}^* = \underset{\underline{\mathbf{h}}}{\operatorname{argmin}} \quad \Pi^{it} + \underline{\mathbf{h}}^T \cdot \underline{\mathbf{f}}^{it} + \frac{1}{2}\underline{\mathbf{h}}^T \cdot \underline{\underline{\mathbf{K}}}^{it} \cdot \underline{\mathbf{h}}, \tag{2.138}$$

$$\text{such that} \qquad \|\underline{\mathbf{h}}\| \leq R_{\mathrm{TR}}, \tag{2.139}$$

where

$$\|\underline{\mathbf{h}}\| = \sqrt{\underline{\mathbf{h}}^T \cdot \underline{\mathbf{h}}}. \tag{2.140}$$

It may be clear from Eq. (2.138) that the exact Hessian is used in this thesis. However, Hessian approximations that are updated during the iteration process as in quasi-Newton methods may also be used [39].

Although the optimization problem of Eq. (2.138) may seem similar to that of Newton's method of Eq. (2.114), it is rather different and not just because of the constraint in

Eq. (2.139). The reason is that Newton's method assumes that the minimizer is characterized by a zero gradient (Eq. (2.118)). However, this is true for any stationary point. Furthermore, Newton's method fails if a singular Hessian is encountered.

Often, the conjugate gradient variant of Steihaug and Toint [45, 46, 39] is used to solve the TR subproblem of Eqs. (2.138) and (2.139). The Steihaug-Toint conjugate gradient method is rather similar to the standard linear conjugate gradient method (to solve systems of linear equations as in Eq. (2.63)) [98]. The reason is that it first calculates a descent direction of the TR subproblem in a conjugate fashion (lines 1 and 18 to 25 in Alg. 3) and then calculates the stepsize in the search direction. Calculating the stepsize can most of the time be performed in a closed-form manner (line 10 in Alg. 3), similarly as in the standard linear conjugate gradient method to solve systems of linear equations [98], because the TR subproblem's objective function behaves quadratically in the search direction.

Nevertheless, the Steihaug-Toint conjugate gradient method varies from the standard linear method in three ways. First, the Steihaug-Toint variant requires to start at initial guess $\underline{\mathbf{h}} = \underline{\mathbf{0}}$ (line 1 of Alg. 3). This guarantees that $\|\underline{\mathbf{h}}\|$ increases for each iteration (see Fig. 2.6) and hence, it is easy to monitor if $\|\underline{\mathbf{h}}\|$ exceeds TR radius R_{TR}.

Second, if the minimizer of the TR subproblem's objective function (Eq. (2.138)) lies outside the trust region (case B in Fig. 2.6), the positive root of 1D quadratic function must be calculated to determine for which $\underline{\mathbf{h}}^*$ the current step crosses the TR boundary (lines 11 until 17 of Alg. 3). This location is then assumed to be the minimizer of the TR subproblem - according to Steihaug [46] and implemented here, but not according to Toint [45].

Third, the TR subproblem's objective function in the search direction may not have a minimum but a maximum. In this case, the stepsize in the current search direction must be taken as large as possible, until it crosses the TR boundary (case C in Fig. 2.6). Hence, the positive root of the same 1D quadratic function must be calculated to determine the associated stepsize (lines 3 to 9 of Alg. 3).

As may be noted in Alg. 3, the preconditioned variant of the Steihaug-Toint conjugate gradient method is also used here (Alg. 3 yields the unpreconditioned algorithm if $\underline{\underline{\mathbf{M}}}^{it} = \underline{\underline{\mathbf{I}}}$). Preconditioning reduces the number of conjugate gradient iterations. However, it often also affects the number of global TR iterations (denoted by the superscript *it* in the mathematical notation of this subsection). The reason for this is that the conjugate gradient method takes other directions if it is preconditioned. Consequently, if the TR boundary is crossed or a direction of negative curvature is encountered, the assumed minimizer (green stars in Fig. 2.6) will be located elsewhere compared to the unpreconditioned conjugate gradient method.

Preconditioners of the following format are often considered:

$$\underline{\underline{\mathbf{M}}}^{it} = \underline{\underline{\mathbf{\Lambda}}} \cdot \underline{\underline{\mathbf{\Lambda}}}^{cT} \approx \underline{\underline{\mathbf{K}}}^{it}. \tag{2.141}$$

where $\underline{\underline{\mathbf{\Lambda}}}$ denotes a lower triangular matrix of tensors and its transposed $\underline{\underline{\mathbf{\Lambda}}}^{cT}$ its upper triangular counterpart. Consequently, the original TR subproblem of Eqs. (2.138) and (2.139) is then rewritten with the help of:

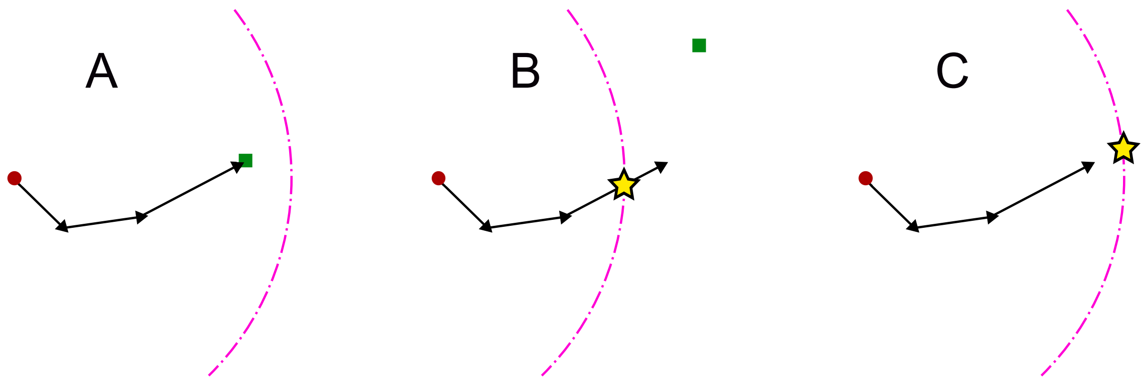


Figure 2.6: Artistic impression of the behaviour of the Steihaug-Toint conjugate gradient method for the three cases that can be encountered. Case A: the TR subproblem's minimizer is located within the TR boundary, case B: the TR subproblem's minimizer is located outside the TR boundary, case C: a direction with negative curvature is encountered during the conjugate gradient process. Red dots: starting points ($\underline{\mathbf{h}} = \underline{\mathbf{0}}$), pink dashed-dotted circles: TR boundaries, black arrows: updates $\underline{\mathbf{p}}$ for $\underline{\mathbf{h}}$, green squares: minimizers of Eq. (2.138), yellow stars: assumed minimizer of the TR subproblem if $\underline{\mathbf{p}}$ crosses the TR boundary (case B), or if $\underline{\mathbf{p}}$ is a direction of negative curvature (case C). This is only an artistic impression because, for a 2D problem as sketched here, the conjugate gradient method would converge in a maximum of two iterations.

$$\underline{\mathbf{h}} = \underline{\underline{\mathbf{\Lambda}}}^{-cT} \cdot \underline{\bar{\mathbf{h}}}, \tag{2.142}$$

as:

$$\underline{\bar{\mathbf{h}}}^* = \underset{\underline{\bar{\mathbf{h}}}}{\text{argmin}} \quad \Pi^{it} + \underline{\bar{\mathbf{h}}}^T \cdot \underline{\bar{\mathbf{f}}}^{it} + \frac{1}{2}\underline{\bar{\mathbf{h}}}^T \cdot \underline{\underline{\bar{\mathbf{K}}}}^{it} \cdot \underline{\bar{\mathbf{h}}}, \tag{2.143}$$

$$\text{such that} \qquad \|\underline{\bar{\mathbf{h}}}\| \leq R_{\text{TR}}, \tag{2.144}$$

where $\underline{\bar{\mathbf{h}}}$ denotes the preconditioned correction and

$$\underline{\bar{\mathbf{f}}}^{it} = \underline{\underline{\mathbf{\Lambda}}}^{-1} \cdot \underline{\mathbf{f}}^{it}, \tag{2.145}$$

$$\underline{\underline{\bar{\mathbf{K}}}}^{it} = \underline{\underline{\mathbf{\Lambda}}}^{-1} \cdot \underline{\underline{\mathbf{K}}}^{it} \cdot \underline{\underline{\mathbf{\Lambda}}}^{-cT}. \tag{2.146}$$

It may be observed in Alg. 3 that only preconditioner $\underline{\underline{\mathbf{M}}}^{it}$ is required and not necessarily the factorization of Eq. (2.141). Nevertheless, this thesis considers the incomplete Cholesky factorization of the Hessian, whilst [31] considers the full Cholesky factorization. No fill-in is used nor a drop tolerance. Thus, $\underline{\underline{\mathbf{\Lambda}}}$ and $\underline{\underline{\mathbf{\Lambda}}}^T$ in Eq. (2.141) denote the lower and upper triangular forms of the Hessian, respectively.

In case of incomplete Cholesky preconditioning, the preconditioner $\underline{\underline{\mathbf{M}}}^{it}$ is not guaranteed to be invertible. Therefore, the implementation of this thesis monitors if the preconditioner is singular, in which case it alters the Hessian's diagonal and recomputes the incomplete Cholesky factorization. This procedure is repeated as often as necessary.

Similar to [31] (and for instance standard available in MATLAB), if the preconditioner is singular, this thesis alters the diagonal of the Hessian by inflating its diagonal components according to:

$$\operatorname{diag}\left(\underline{\underline{\mathbf{K}}}^{it}\right) = \left(1 + 10^{i-8}\right) \operatorname{diag}\left(\underline{\underline{\mathbf{K}}}^{it}\right), \tag{2.147}$$

where i refers to the i^{th} time that the Hessian's diagonal of the current iteration is inflated and diag($\bullet$) refers to the diagonal of the second-order tensors on the diagonal of the matrix. As [31] mentions, this procedure will eventually provide an invertible triangular form if the diagonal components of the Hessian are positive.

Where this thesis differs from [31], however, is that this thesis' examples push the solver to such limits that negative diagonal components are occasionally encountered. In such scenarios, the negative diagonal components are simply made positive before they are potentially inflated. Also different from [31] is that this thesis recomputes the preconditioner for each iteration, whereas [31] only does so depending on the preconditioner's performance for the previous TR iteration (quantified by the number of conjugate gradient iterations of the previous global TR iteration).

Updating in trust region methods

Once the TR subproblem of Eqs. (2.143) and (2.144) is solved, $\underline{\mathbf{h}}^*$ is known. The next questions the TR algorithm must answer is whether correction $\underline{\mathbf{h}}^*$ is good enough to be used as an update of the current estimate $\underline{\mathbf{u}}^{it}$, and whether or not the second-order Taylor approximation of Eq. (2.138) has reflected the exact objective function sufficiently accurately in the TR (characterized by the radius of TR R_{TR}).

For both decisions, TR methods consider the ratio between the reduction of the exact objective function generated by the correction $\underline{\mathbf{h}}^*$ and the reduction that is expected based on the approximation of Eq. (2.138). The ratio between the two, ρ, can thus be expressed as follows:

$$\rho = \frac{\Pi^{it} - \Pi^{it+h^*}}{\bar{\Pi}^{it}(\underline{\mathbf{0}}) - \bar{\Pi}^{it}(\underline{\mathbf{h}}^*)}, \tag{2.148}$$

where Π^{it} denotes the exact objective function at $\underline{\mathbf{u}}^{it}$ as provided in Eq. (2.115) and Π^{it+h^*} denotes the exact objective function at $\underline{\mathbf{u}}^{it} + \underline{\mathbf{h}}^*$:

$$\begin{aligned}\Pi^{it+h^*} = \Pi_{\text{mat/u}} &\left(\underline{\mathbf{u}}^{it} + \underline{\mathbf{h}}^*, \underline{\underline{\zeta}}_{\text{prev}}, \underline{\underline{\zeta}}^{*it+h^*}\right) \\ &- \Pi_{\text{con}}\left(\underline{\mathbf{u}}^{it} + \underline{\mathbf{h}}^*, \underline{\underline{\zeta}}^{*it+h^*}(\underline{\mathbf{u}}^{it} + \underline{\mathbf{h}}^*)\right) - \underline{\mathbf{f}}_{\text{ext}}^T \cdot \left(\underline{\mathbf{u}}^{it} + \underline{\mathbf{h}}^*\right).\end{aligned} \tag{2.149}$$

Furthermore, $\bar{\Pi}^{it}$ denotes the objective function approximation used in the TR subproblem of Eq. (2.138).

Algorithm 3 Preconditioned Steihaug-Toint conjugate gradient method.

```
INPUT: $R_{\text{TR}}$, $\underline{\mathbf{f}}^{it}$, $\underline{\underline{\mathbf{K}}}^{it}$, $\underline{\underline{\mathbf{M}}}^{it}$
1: $\underline{\mathbf{r}} = -\underline{\mathbf{f}}^{it}$, Solve $\underline{\underline{\mathbf{M}}}^{it} \cdot \underline{\mathbf{p}} = \underline{\mathbf{r}}$ for $\underline{\mathbf{p}}$, $\underline{\mathbf{q}} = \underline{\mathbf{p}}$, $\underline{\mathbf{h}}^* = \underline{\mathbf{0}}$
2: while do
3:     if $\underline{\mathbf{p}}^T \cdot \underline{\underline{\mathbf{K}}}^{it} \cdot \underline{\mathbf{p}} \leq 0$ then
4:         Raise *flag* that TR boundary is reached
5:         $a = \underline{\mathbf{p}}^T \cdot \underline{\underline{\mathbf{M}}}^{it} \cdot \underline{\mathbf{p}}$, $b = 2\underline{\mathbf{p}}^T \cdot \underline{\underline{\mathbf{M}}}^{it} \cdot \underline{\mathbf{h}}^*$, $c = \underline{\mathbf{h}}^{*T} \cdot \underline{\underline{\mathbf{M}}}^{it} \cdot \underline{\mathbf{h}}^* - R_{\text{TR}}^2$
6:         $\alpha = \frac{-b+\sqrt{b^2-4ac}}{2a}$
7:         $\underline{\mathbf{h}}^* = \underline{\mathbf{h}}^* + \alpha\, \underline{\mathbf{p}}$
8:         BREAK
9:     end if
10:    $\alpha = \frac{\underline{\mathbf{r}}^T \cdot \underline{\mathbf{q}}}{\underline{\mathbf{p}}^T \cdot \underline{\underline{\mathbf{K}}}^{it} \cdot \underline{\mathbf{p}}}$
11:    if $(\underline{\mathbf{h}}^* + \alpha\underline{\mathbf{p}})^T \cdot \underline{\underline{\mathbf{M}}}^{it} \cdot (\underline{\mathbf{h}}^* + \alpha\underline{\mathbf{p}}) > R_{\text{RT}}^2$ then
12:        Raise *flag* that TR boundary is reached
13:        $a = \underline{\mathbf{p}}^T \cdot \underline{\underline{\mathbf{M}}}^{it} \cdot \underline{\mathbf{p}}$, $b = 2\underline{\mathbf{p}}^T \cdot \underline{\underline{\mathbf{M}}}^{it} \cdot \underline{\mathbf{h}}^*$, $c = \underline{\mathbf{h}}^{*T} \cdot \underline{\underline{\mathbf{M}}}^{it} \cdot \underline{\mathbf{h}}^* - R_{\text{TR}}^2$
14:        $\alpha = \frac{-b+\sqrt{b^2-4ac}}{2a}$
15:        $\underline{\mathbf{h}}^* = \underline{\mathbf{h}}^* + \alpha\, \underline{\mathbf{p}}$
16:        BREAK
17:    end if
18:    $\underline{\mathbf{h}}^* = \underline{\mathbf{h}}^* + \alpha\, \underline{\mathbf{p}}$
19:    $\phi = \underline{\mathbf{r}}^T \cdot \underline{\mathbf{p}}$
20:    $\underline{\mathbf{r}} = \underline{\mathbf{r}} - \alpha\, \underline{\underline{\mathbf{K}}}^{it} \cdot \underline{\mathbf{p}}$
21:    if $\|\underline{\mathbf{r}}\| < \max\left(10^{-15}, 10^{-5}\|\underline{\mathbf{f}}^{it}\|\right)$ then
22:        BREAK
23:    end if
24:    Solve $\underline{\underline{\mathbf{M}}}^{it} \cdot \underline{\mathbf{p}} = \underline{\mathbf{r}}$ for $\underline{\mathbf{p}}$
25:    $\underline{\mathbf{p}} = \underline{\mathbf{q}} + \frac{\underline{\mathbf{r}}^T \cdot \underline{\mathbf{q}}}{\phi}\, \underline{\mathbf{p}}$
26: end while
OUTPUT: $\underline{\mathbf{h}}^*$, *flag* (indicating whether or not TR boundary is reached)
```

Algorithm 4 Updating in the TR method of this thesis.

INPUT: $\underline{\mathbf{h}}^*$, R_{TR}, $R_{\mathrm{TR/max}}$, Π^{it}, $\underline{\mathbf{f}}^{it}$, $\underline{\underline{\mathbf{K}}}^{it}$, *flag* (indicating whether or not TR boundary was reached)

```
 1: Evaluate $\underline{\mathbf{f}}^{it+h^*}$ for $\underline{\mathbf{u}}^{it} + \underline{\mathbf{h}}^*$
 2: if $\|\underline{\mathbf{f}}^{it+h^*}\| = \mathrm{NaN}$ then
 3:     $\rho = 0$
 4: else
 5:     $\rho = \frac{\underline{\mathbf{h}}^{*T} \cdot (\underline{\mathbf{f}}^{it} + \underline{\mathbf{f}}^{it+h^*})}{2\underline{\mathbf{h}}^{*T} \cdot \underline{\mathbf{f}}^{it} + \underline{\mathbf{h}}^{*T} \cdot \underline{\underline{\mathbf{K}}}^{it} \cdot \underline{\mathbf{h}}^*}$
 6: end if
 7: if $\rho < 0.25$ then
 8:     $\underline{\mathbf{u}}^{it+1} = \underline{\mathbf{u}}^{it}$
 9:     $R_{\mathrm{TR}} = 0.25 R_{\mathrm{TR}}$
10: else
11:     $\underline{\mathbf{u}}^{it+1} = \underline{\mathbf{u}}^{it} + \underline{\mathbf{h}}^*$
12:     if $\rho > 0.75$ and 'TR boundary was reached' then
13:         $R_{\mathrm{TR}} = \min(2R_{\mathrm{TR}}, R_{\mathrm{TR/max}})$
14:     end if
15: end if
```

OUTPUT: $\underline{\mathbf{u}}^{it+1}$, R_{TR}

As the objective function is substantially more easily polluted by noise due to machine precision than the gradient, the expression of Eq. (2.148) cannot be used. For this purpose, this thesis borrows the formulation for ρ of [31], which assumes that the gradient in direction $\underline{\mathbf{h}}^*$ scales similarly to the objective function in direction $\underline{\mathbf{h}}^*$. Thus, the following expression is used here:

$$\rho = \frac{\underline{\mathbf{h}}^{*T} \cdot (\underline{\mathbf{f}}^{it} + \underline{\mathbf{f}}^{it+h^*})}{2\underline{\mathbf{h}}^{*T} \cdot \underline{\mathbf{f}}^{it} + \underline{\mathbf{h}}^{*T} \cdot \underline{\underline{\mathbf{K}}}^{it} \cdot \underline{\mathbf{h}}^*}, \tag{2.150}$$

where the denominator is replaced with the help of Eq. (2.138).

Alg. 4 clearly demonstrates that the ratio ρ is used to both decide whether or not estimate $\underline{\mathbf{u}}^{it}$ must be updated with correction $\underline{\mathbf{h}}^*$, and if the TR radius R_{TR} must be decreased for the next iteration, kept the same or increased. $R_{\mathrm{TR/max}}$ in Alg. 4 denotes the maximal TR radius as defined by the user. The full TR algorithm (not shown) also requires the user to define an initial value for TR radius R_{TR}.

2.4 Gregory patches

One point that has not yet been discussed is the leader-surface description. The current section sheds details on this. In other words, the current section discusses the formulation of $\mathbf{X}_{\mathrm{lead}}$ and $\mathbf{n}_{\mathrm{lead}}$ in Eqs. (2.11) and (2.12), as a function of the two parametric surface coordinates in $\underline{\xi}$.

This thesis considers FE vertices on the deformable body's surface as the follower-vertices and the surface of the rigid body as the leader-surface. The leader-surface is generated using standard bilinear FE discretizations. However, the FE discretization is not directly used as the description of the leader-surface. Instead, the surface is transformed using Gregory patches [99] and the resulting surface is considered as the leader-surface.

The advantage of using Gregory patches is illustrated graphically in Fig. 2.7. The top image shows that two particularities occur if an FE discretization were used as the leader-surface. The first particularity occurs if a follower-vertex is located in one of the two domains bounded by blue arrows. In such a case, the projection of the follower-vertex falls on the associated vertex of the leader-surface, regardless of where the follower-vertex is located in the domain that is bounded by the blue arrows in Fig. 2.7.

The second and more crucial particularity occurs if a follower-vertex penetrates the leader-surface near the top of the mountain in the middle of the image. The problem occurs if the follower-vertex moves from the left of the red dashed line to the right of the red dashed line. This is a problem because the direction of the follower-vertex's force vector changes abruptly, i.e. the horizontal force component changes sign. This problem occurs no matter how close the follower-vertex is located near the top of the mountain, because the red dashed line is attached to the leader-surface.

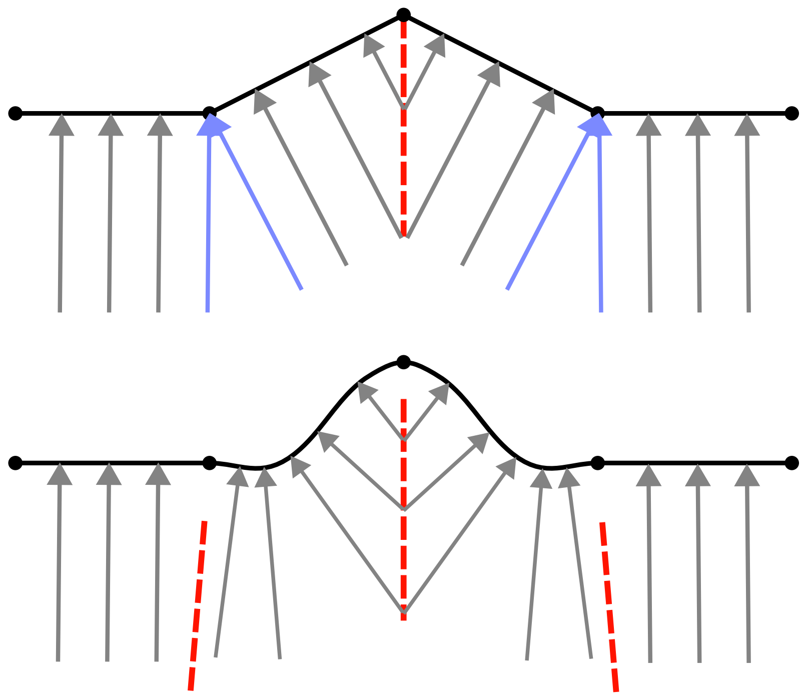


Figure 2.7: Two sketches of the same leader-surface (black lines). The top one is described by standard linear FEs (top) and and the bottom one by Gregory patches (bottom). The arrows indicate to which leader-surface location a follower-vertex is projected. If a follower-vertex crosses a red dashed line, its associated force vector changes drastically. Because the red dashed lines are not attached to the vertices of the leader-surface for the Gregory patches (bottom), this problem is less pronounced than if an FE discretization is used for the leader-surface description.

This problem does not occur, or at least to a lesser extent, for Gregory patches; see the bottom image of Fig. 2.7. The reason is that the smoothness obtained by the use of Gregory patches shifts this discrete transition away from the leader-surface. Thus, the discrete change may still occur, but if the follower-vertex moves closer to the leader-surface it will eventually disappear. The bottom image of Fig. 2.7 presents this by means of the fact that the red dashed lines are not connected to the leader-surface.

The remainder of this section provides a concise description of Gregory patches. Since this thesis only uses the rigid body as the leader-body, the discussion here is limited to the reference configuration. Gregory patches for contact surfaces of deformable bodies are described in more detail in [60].

Single Gregory patch

In the current work, each quadrilateral segment of the rigid leader-body is smoothed with a single Gregory patch. The inputs required to construct a Gregory patch are the 3D locations of the corner vertices of the associated quadrilateral segment, and those with which the four corner vertices share an edge (i.e. the red dots in the left image of Fig. 2.8).

The 3D location on a Gregory patch, $\mathbf{X}_{\text{lead}}$, can be written as a function of parametric surface coordinates $\underline{\xi}$ as follows:

$$\mathbf{X}_{\text{lead}}(\underline{\xi}) = \sum_{i=0}^{3}\sum_{j=0}^{3} B_i^3(\xi_1)B_j^3(\xi_2)\mathbf{X}_{ij}(\xi_1,\xi_2), \tag{2.151}$$

where $0 \leq \xi_1 \leq 1$ and $0 \leq \xi_2 \leq 1$, $B_i^3(\bullet)$ denotes the i^{th} term of the Bernstein polynomial of degree three, and $\mathbf{X}_{ij}$ denotes the location of a control point, with the exception of the following four cases:

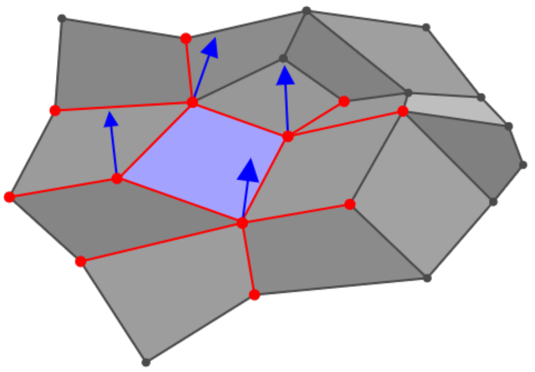
$$\begin{aligned}
\mathbf{X}_{11}(\underline{\xi}) &= (\xi_1\mathbf{X}_{110} + \xi_2\mathbf{X}_{111})/(\xi_1+\xi_2),\\
\mathbf{X}_{21}(\underline{\xi}) &= ((1-\xi_1)\mathbf{X}_{210} + \xi_2\mathbf{X}_{211})/(1-\xi_1+\xi_2),\\
\mathbf{X}_{12}(\underline{\xi}) &= (\xi_1\mathbf{X}_{120} + (1-\xi_2)\mathbf{X}_{121})/(\xi_1+1-\xi_2),\\
\mathbf{X}_{22}(\underline{\xi}) &= ((1-\xi_1)\mathbf{X}_{220} + (1-\xi_2)\mathbf{X}_{221})/(2-\xi_1-\xi_2),
\end{aligned} \tag{2.152}$$

which are themselves clearly functions of eight other control points.

Thus, each Gregory patch (Eq. (2.151)) requires a total of 20 control points (right in Fig. 2.8). These 20 control points can be classified as follows:

$$\begin{aligned}
\text{corners}&\left\{\mathbf{X}_{00},\mathbf{X}_{03},\mathbf{X}_{30},\mathbf{X}_{33},\right.\\
\text{edges}&\begin{cases}\mathbf{X}_{10},\mathbf{X}_{20},\mathbf{X}_{01},\mathbf{X}_{02}\\ \mathbf{X}_{13},\mathbf{X}_{23},\mathbf{X}_{31},\mathbf{X}_{32}\end{cases}\\
\text{interior}&\begin{cases}\mathbf{X}_{110},\mathbf{X}_{111},\mathbf{X}_{210},\mathbf{X}_{211},\\ \mathbf{X}_{120},\mathbf{X}_{121},\mathbf{X}_{220},\mathbf{X}_{221}.\end{cases}
\end{aligned}$$

The locations of the four corner points correspond to the locations of the corner vertices



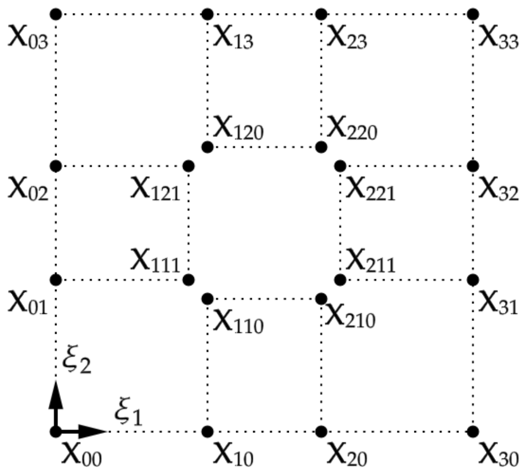


Figure 2.8: Left: A Gregory patch for the highlighted quadrilateral surface segment requires the average normal unit vectors in the four associated corner points (blue arrows). This is achieved by considering all the corner points with which the four corner points share an edge. Right: The locations of the 20 control points of a Gregory patch, together with surface coordinates ξ_1 and ξ_2.

of the associated quadrilateral segment. The remainder of this section discusses the computation of the locations of the edge control points and the interior control points. Since the average normal vectors in the four vertices of the associated quadrilateral segment are required, their construction will first be discussed.

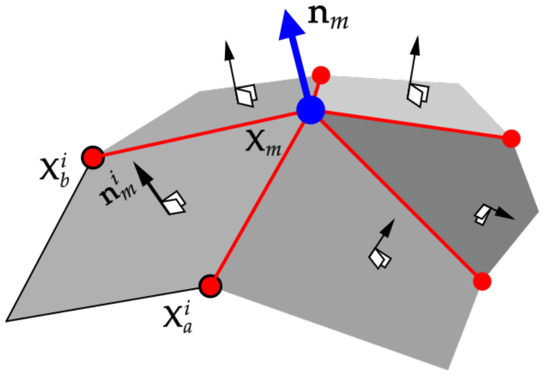


Figure 2.9: Average normal vector $\mathbf{n}_m$ of point m is constructed as the average of the normal vectors of its facets.

Average normal vectors

For the mth vertex ($m \in \{0, 1, 2, 3\}$), the average normal vector reads:

$$\mathbf{n}_m = \frac{\sum_{i=1}^{n_\mathrm{f}} \mathbf{n}_m^i}{\| \sum_{i=1}^{n_\mathrm{f}} \mathbf{n}_m^i \|}, \tag{2.153}$$

where $\mathbf{n}_m^i$ denotes the (normalized) normal vector of the ith facet connected to vertex m.

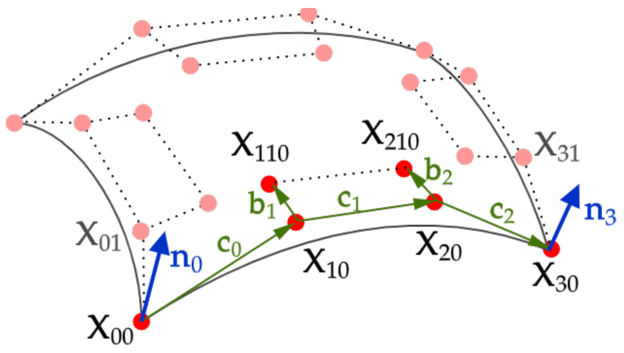


Figure 2.10: Construction of the edge control points and the interior control points associated with the edge between corner control points $\mathbf{X}_{00}$ and $\mathbf{X}_{30}$.

n_f denotes the number of facets connected to vertex m. Fig. 2.9 presents this graphically.

The normal vector of the i^{th} facet can be expressed as follows:

$$\mathbf{n}_m^i = \frac{(\mathbf{X}_b^i - \mathbf{X}_m) \times (\mathbf{X}_a^i - \mathbf{X}_m)}{\|(\mathbf{X}_b^i - \mathbf{X}_m) \times (\mathbf{X}_a^i - \mathbf{X}_m)\|}, \tag{2.154}$$

where $\mathbf{X}_b^i$ and $\mathbf{X}_a^i$ denote the locations of the two additional corner points that together with $\mathbf{X}_m^i$ define facet i (see Fig. 2.9). $\times$ denotes the cross product.

Instead of discussing the construction of all edge control points and interior control points, the discussion is limited to those that are associated with a single edge. The reason is that the edge-wise construction must simply be repeated for all four edges to define all edge control points and interior control points of the Gregory patch. Below, this edge-wise construction is only detailed for the edge associated with corner control points $\mathbf{X}_{00}$ and $\mathbf{X}_{30}$, which is presented graphically in Fig. 2.10.

Edge and interior control points

The locations of the two edge control points associated with the edge between the corner control points $\mathbf{X}_{00}$ and $\mathbf{X}_{30}$ can be written as follows (see Fig. 2.10):

$$\mathbf{X}_{10} = \mathbf{X}_{00} + \mathbf{c}_0, \tag{2.155}$$
$$\mathbf{X}_{20} = \mathbf{X}_{30} - \mathbf{c}_2, \tag{2.156}$$

with

$$\mathbf{c}_0 = \frac{1}{3}\left((\mathbf{X}_{30} - \mathbf{X}_{00}) - (\mathbf{X}_{30} - \mathbf{X}_{00}) \cdot \mathbf{n}_0\, \mathbf{n}_0\right), \tag{2.157}$$
$$\mathbf{c}_2 = \frac{1}{3}\left((\mathbf{X}_{30} - \mathbf{X}_{00}) - (\mathbf{X}_{30} - \mathbf{y}_{00}) \cdot \mathbf{n}_3\, \mathbf{n}_3\right). \tag{2.158}$$

The locations of the two interior control points associated with the edge between the corner control points $\mathbf{X}_{00}$ and $\mathbf{X}_{30}$ can be written as follows (see Fig. 2.10):

$$\mathbf{X}_{110} = \mathbf{X}_{10} + \mathbf{b}_1, \tag{2.159}$$
$$\mathbf{X}_{210} = \mathbf{X}_{20} + \mathbf{b}_2, \tag{2.160}$$

with

$$\mathbf{b}_1 = \frac{1}{3}\left((d_0 + d_2)\mathbf{a}_0 + d_0\mathbf{a}_3 + 2d_1\mathbf{c}_1 + d_3\mathbf{c}_0\right), \tag{2.161}$$
$$\mathbf{b}_2 = \frac{1}{3}\left((d_0 + d_2)\mathbf{a}_3 + d_2\mathbf{a}_0 + 2d_3\mathbf{c}_1 + d_1\mathbf{c}_2\right), \tag{2.162}$$

where

$$\mathbf{a}_0 = \frac{\mathbf{n}_0 \times (\mathbf{X}_{30} - \mathbf{X}_{00})}{\|\mathbf{n}_0 \times (\mathbf{X}_{30} - \mathbf{X}_{00})\|}, \tag{2.163}$$
$$\mathbf{a}_3 = \frac{\mathbf{n}_3 \times (\mathbf{X}_{30} - \mathbf{X}_{00})}{\|\mathbf{n}_3 \times (\mathbf{X}_{30} - \mathbf{X}_{00})\|}, \tag{2.164}$$
$$\mathbf{c}_1 = \mathbf{X}_{20} - \mathbf{X}_{10}, \tag{2.165}$$
$$d_0 = \mathbf{a}_0 \cdot \mathbf{b}_0, \tag{2.166}$$
$$d_1 = \frac{\mathbf{c}_0 \cdot \mathbf{b}_0}{\mathbf{c}_0 \cdot \mathbf{c}_0}, \tag{2.167}$$
$$d_2 = \mathbf{a}_3 \cdot \mathbf{b}_3, \tag{2.168}$$
$$d_3 = \frac{\mathbf{c}_2 \cdot \mathbf{b}_3}{\mathbf{c}_2 \cdot \mathbf{c}_2}. \tag{2.169}$$

2.5 Results

This section presents a demonstration of the capabilities and limitations of the aforementioned optimization approaches. The first subsection focuses on 2D simulations and the second subsection focuses on 3D simulations. Because the 2D results are substantially more straightforward to visually interpret than the 3D results, the 2D results are discussed in substantially more detail. They, for instance, indicate substantially clearly whether snap-through or snap-back phenomena are encountered. First, however, the simulation setups are discussed in detail.

Simulation setups

The same mechanical parameters are used in all simulations. The elastic parameters are set to $E = 0.05$ and $\nu = 0.3$. The plastic parameters are set to $M_{\mathrm{y0}} = 0.01$, $h_{\mathrm{hard}} = 0.05$ and $m_{\mathrm{hard}} = 1$. Forcing the constitutive model to behave purely elastically is achieved by setting the initial yield stress M_{y0} substantially large.

Fig. 2.11 graphically presents the engineering stress-engineering strain behaviour of the two material models for a single material point subjected to uniaxial tension. In other words, the material point is free to contract in the directions in which it is not actively elongated. Fig. 2.11 clearly demonstrates that the stress-strain behaviour is non-linear during hardening, although a linear hardening law is applied. This is largely caused by the multiplicative decomposition of the deformation gradient tensor in the constitutive model and not merely by the fact that the engineering quantities of the stress and strain are presented.

Newton's method is the main solver in the simulations. If Newton's method does not converge in 15 iterations for the 2D simulations and 20 iterations for the 3D simulations, a cut-back is made that divides the time increment in half, and Newton's method is again attempted. This process is repeated until the adjusted time increment is only a factor of $2^{-10} = 1024^{-1}$ of the original time increment. If another cutback is required, the solver switches from Newton's method to one of the true minimization algorithms. Cutbacks are also initiated if Newton's method with the active set strategy to enforce the contact constraints encounters the same active set twice in the same time increment.

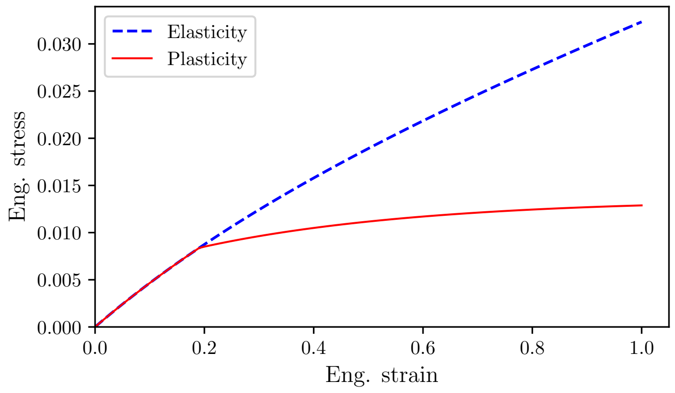


Figure 2.11: The engineering stress-engineering strain responses of the two material models for a single material point exposed to uniaxial tension.

After the true minimization algorithm has converged (or Newton's method, for that matter), the size of the time increment is increased, and hence, Newton's method is used for the next time increment. Increasing the size of the time increment is continued until the original time increment is reached. Each simulation is initially divided into 100 time increments. All solvers quantify the residual as the L^2-norm of the (not normalized) gradient, which equals the column with force vectors of Eq. (2.59). The tolerance in all solvers is set to 10^{-11}.

Cutbacks are also allowed in the return mapping algorithm - for each integration point separately. Time increments solved using Newton's method do not often require cutbacks of the return mapping algorithm, but they can be crucial if a minimization algorithm is used because enormous deformation changes can occur. The return mapping algorithm performs a cutback in three cases: i) if $J_e < 10^{-12}$ is encountered, ii) if a plastic deformation gradient tensor is not invertible, and iii) if the new residual of the return mapping algorithm is ten times larger than that of the previous return mapping iteration.

The residual in the return mapping algorithm is quantified by the magnitude of the yield function y and the tolerance is set to 10^{-15}. The total number of cutbacks that the return mapping algorithm is allowed to make is 25, which entails a maximum refinement of approximately 33 million. If still no convergence is reached beyond this limit, the column of force vector will be returned as Not-a-Number.

The true minimization algorithms come with a number of hyperparameters that are not yet provided. For the line search of the quasi-Newton methods, parameters $\delta\alpha$, $maxiter_{\text{ls/q}}$ and $maxiter_{\text{ls/s}}$ are set to 0.5, 20 and 10, respectively. The initial approximation of the inverted Hessian is set to $\underline{\underline{\hat{\mathbf{L}}}}^0 = \underline{\underline{\mathbf{I}}}$. The different numbers of previous iterations that are incorporated in the L-BFGS method (i.e. n_{li}) are reported in the results below. In addition, for the TR methods, the initial TR radius is set as $10^{-4}n_{\text{var}}$ (where n_{var} denotes the number of variables/degrees of freedom). No maximum for the TR radius is set, although a preliminary investigation that is not reported below has indicated that enforcing a maximum may accelerate the minimization, but a deceleration will be observed if the maximum is set too small. However, the predictions did not differ from those obtained without maximal TR radius.

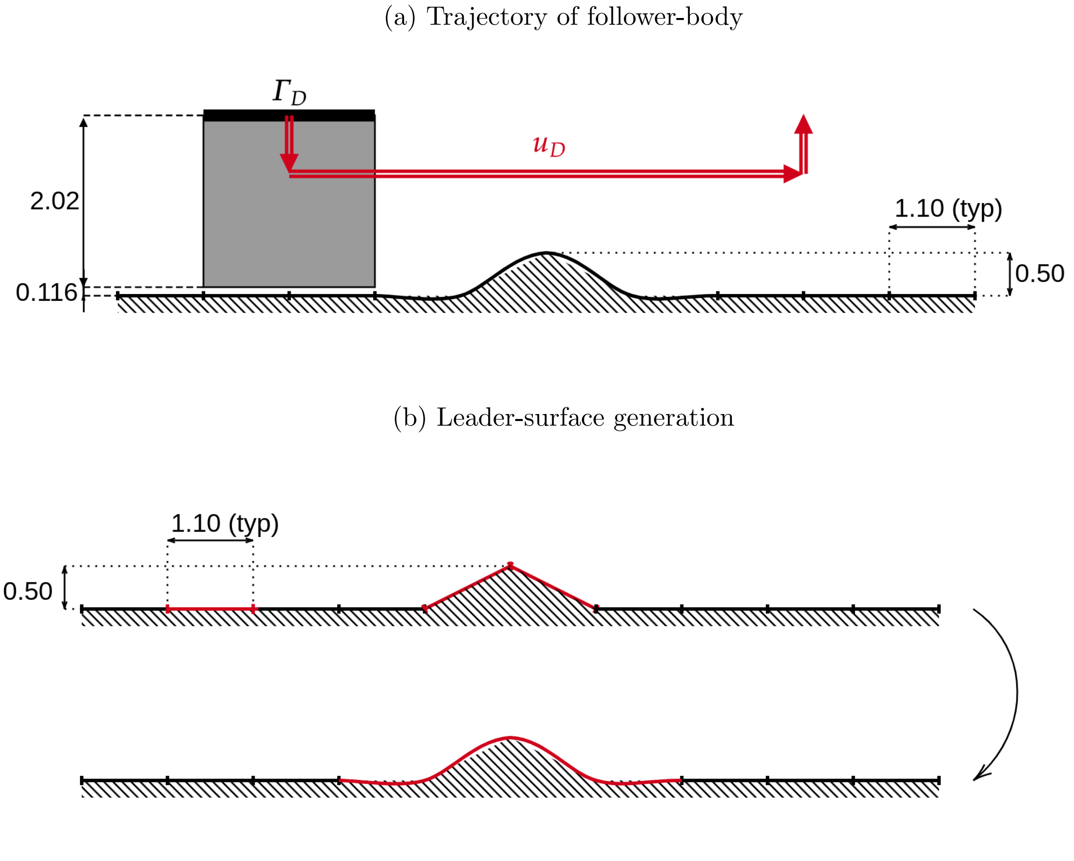
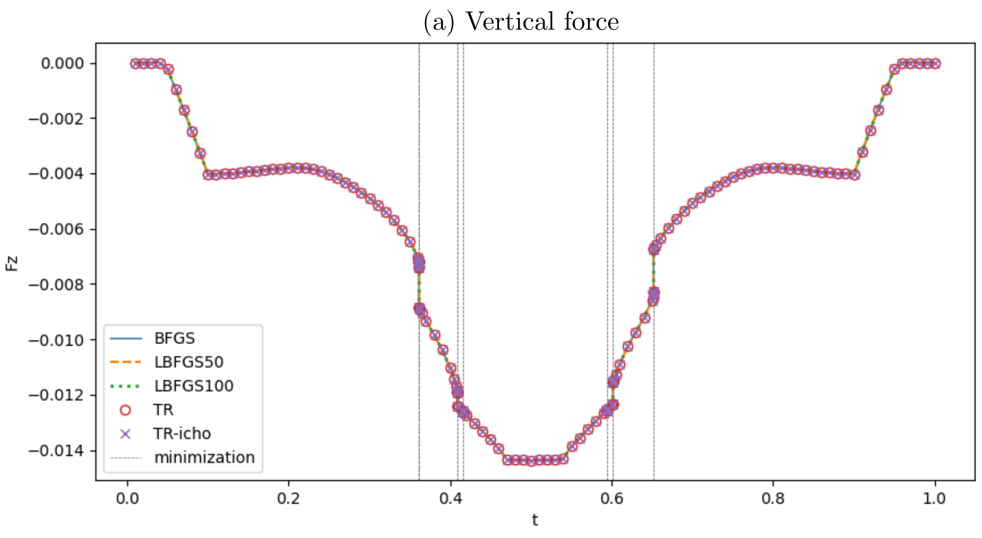

(a) Trajectory of follower-body
Γ_D
u_D
2.02
0.116
1.10 (typ)
0.50
(b) Leader-surface generation
1.10 (typ)
0.50

Figure 2.12: Sketches of the 2D simulation setup.

2.5.1 2D results

The 2D contact simulations focus on the sliding of a square deformable body under compression, as illustrated in the top image of Fig. 2.12. The displacements of the vertices on the top of the block are completely prescribed during the simulation. The block is first compressed against the leader-surface during the first 10 increments by lowering the top edge with a vertical displacement of 0.25. During the next 80 increments, the block moves to the right with a displacement of 6.00. Finally, the compression is released during the last 10 increments of the simulation.

The generation of the leader-surface is presented in the bottom image of Fig. 2.12. It is worth to realize that the leader surface has not only a hill in the center, but also two small valleys on both sides of the hill.



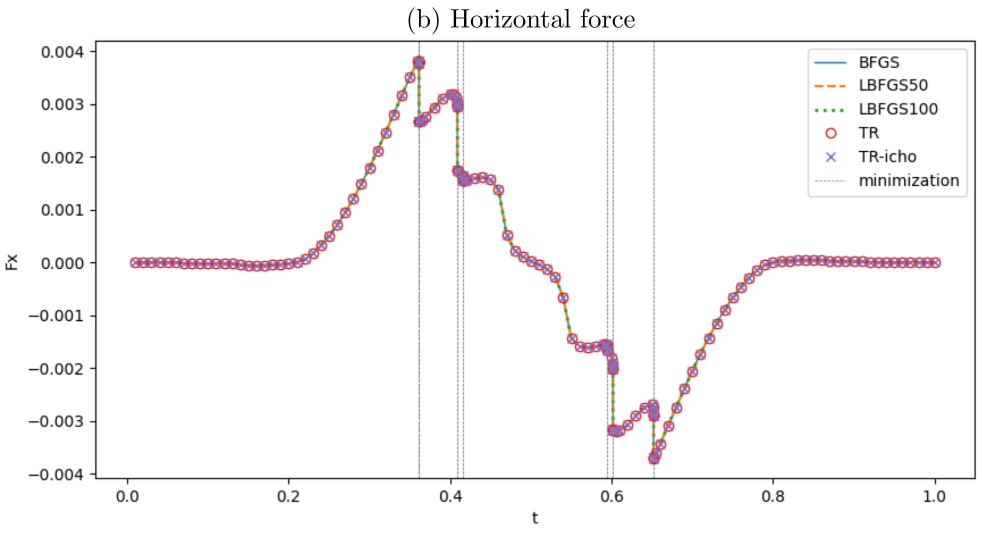


Figure 2.13: 2D hyperelasticity, 5×5 FEs. The force-time curves predicted by the different methods. The times at which a minimization solver is applied are indicated by the black vertical lines. Around a time of 0.36 two minimizations take place almost immediately after each other. They cannot be distinguished in the diagrams. The number behind 'LBFGS' in the legend refers to n_{li}.

Appendix A shows a simple sensitivity study for the stiffness parameter k and the contact variants described in Section 2.2.3. The three variants of contact enforcement are investigated for Newton's method for the very first 2D elastic simulation scenario. For the unilateral case with constant penalty stiffness and the active set method, which also comes with a constant penalty stiffness, penalty stiffnesses ranging from 10^2 to 10^6 are investigated. In the unilateral case with an adaptive penalty stiffness, the initial stiffness is set to one and a conservative value of $\kappa = 0.1$ in Eq. (2.97) is used to update the penalty stiffness. The values for maximally allowed penetration $g_{\max}$ range from 10^{-5} to 10^{-3}. The minimally allowed penetration is set such that $g_{\min} = 0.5 g_{\max}$.

2D elasticity, 5×5 FEs

The diagrams in Fig. 2.13 present the vertical and horizontal reaction forces predicted by the different methods in case the block deforms elastically and is discretized with 5×5 FEs (i.e. $n_{\text{var}} = 60$). All predictions are clearly the same. Some snapshots of the simulations are presented in Fig. 2.14.

The vertical black lines in Fig. 2.13 indicate when so many cut-backs have taken place (i.e. 10) that the solver switches from Newton's method to a true minimization method. Newton's method with the active set strategy is applied here, in combination with a (constant) penalty stiffness of 100. The reason that Newton's method with the active set method cannot get past the time of the first two minimizations is that the same active set is encountered no matter how often the time increment is refined.

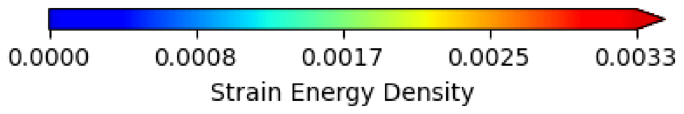
For Newton's method with the active set approach and (constant) penalty values ranging from 10^3 to 10^6, the simulation can, in fact, get past the first two times that a minimization algorithm is applied without switching the solver, but not past the time of the third minimization. In this case, the cause is that Newton's method simply does not converge for the time of the third minimization, no matter how many cut-backs are taken. In other words, the incremental objective function is not sufficiently convex. This can be visually inspected for minimization III in Fig. 2.15, since the left neighbour of the bottom right edges passes the top of the hill, whilst the curvature of the block's right edge changes drastically. Thus, minimizations III and VI in Fig. 2.15 correspond to snap-back phenomena that Newton's method cannot treat but must be treated for the simulation to continue.

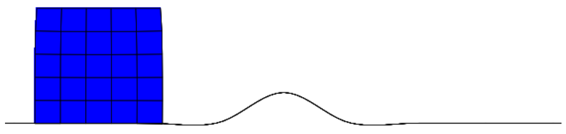
Because a penalty value of 10^3 or greater substantially reduces the speed of the true minimization algorithms and insignificantly improves the performance of Newton's method with the active set method (i.e. minimizations I and II occur at about 36% of the simulation and minimization III at 41%), Newton's method with the active set method with a penalty value of 100 is used as the main solver for the remaining simulations.

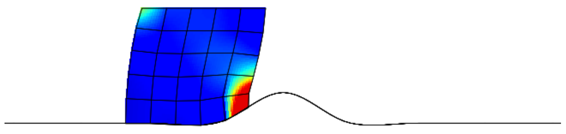
It is nevertheless worth mentioning that if Newton's method with the unilateral contact enforcement with a constant penalty stiffness of 100 or larger is used, the simulation dies before the time of the first two minimizations because Newton's method simply does not converge. On the other hand, if a constant penalty stiffness of 10 or smaller is used, the penetration in the leader-surface is so large when the block crosses the hill that the physical meaning of the simulation is lost.

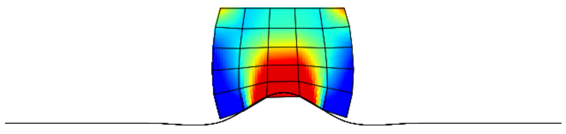
If, in addition, Newton's method with unilateral contact enforcement with the adaptive penalty approach is used, the simulation also dies at the time of the first two minimizations (i.e. at approximately 36% of the simulation), because Newton's method simply does not converge. This is irrespective of maximally allowed penetration $g_{\max}$, which was tested for values of 10^{-5}, 10^{-4} and 10^{-3} (whilst $g_{\min} = 0.5 g_{\max}$).

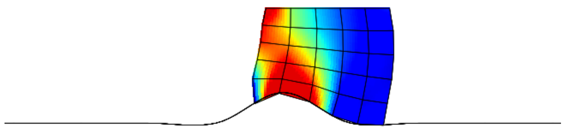
In conclusion, one can state that minimizations I, II, IV, V and VII in Figs. 2.13 and 2.15 could be avoided if Newton's method with the active set strategy and a constant penalty value of 10^3 or larger would be used. However, the increments at minimizations III and

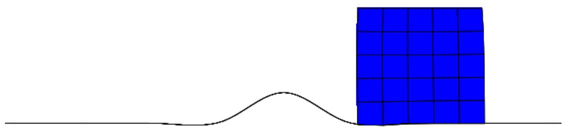



(a) Time=0.10


(b) Time=0.33


(c) Time=0.50


(d) Time=0.60


(e) Time=0.80

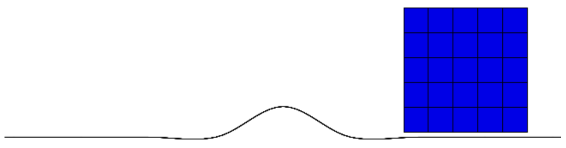
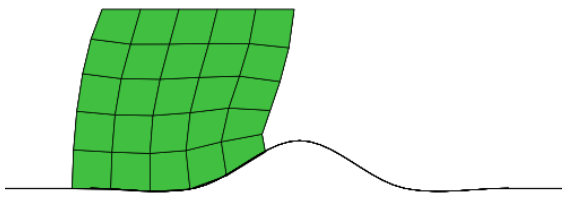

(f) Time=1.00

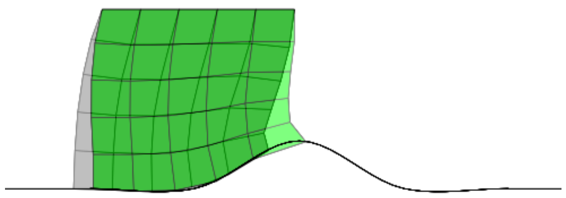
Figure 2.14: 2D hyperelasticity, 5×5 FEs. The deformations predicted by all methods for different pseudo-times.

VI still fail due to the lack of convexity of the incremental objective function. Newton's method with the active set method with a penalty value of 100 is selected as the main solver of the remaining simulations because larger penalty values substantially reduce the speed of the minimization solvers.

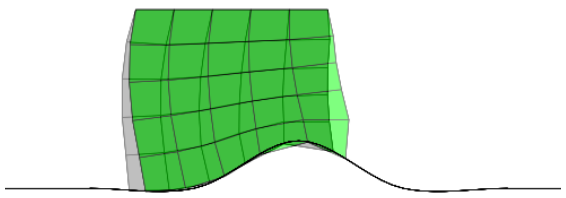
The only issue left to discuss for this example is the computational performance of the different minimization methods. The performances are presented in the first main row of Table 2.2. The L-BFGS method was clearly investigated for $n_{\text{li}} = 100$ and $n_{\text{li}} = 50$. The numbers of increments that were successfully and unsuccessfully treated with Newton's method are all the same, as are the total number of Newton iterations. All the simulations required to switch the solver to a true minimization solver for 7 increments. The total number of iterations that each minimization algorithm required is rather similar for all methods. However, the number of force evaluations is more interesting because it requires more time. The TR methods clearly outperform the quasi-Newton methods. From the two TR methods, the preconditioned is the clear winner, since its number of force evaluations is only 67% of that of the original TR method (while the number of CG iterations reduces with 27%). On the other hand, the construction of the preconditioner can certainly not be ignored.



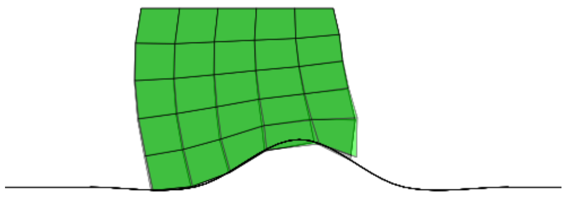
(a) Minimization I



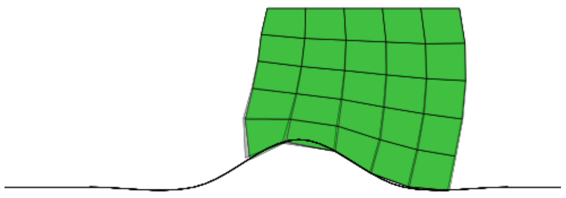
(b) Minimization II



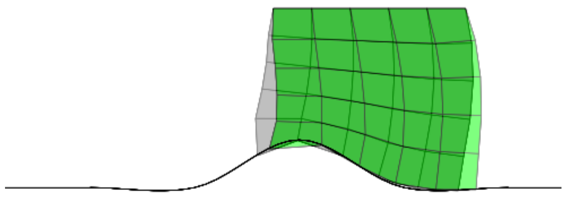
(c) Minimization III



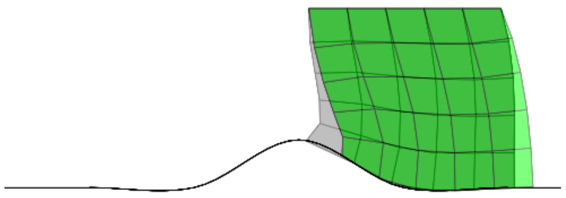
(d) Minimization IV



(e) Minimization V



(f) Minimization VI



(g) Minimization VII

Figure 2.15: 2D hyperelasticity, 5×5 FEs. The deformations before a minimization algorithm is applied (grey) and after a minimization algorithm has converged (green). Thus, the deformations correspond to the vertical black lines in Fig. 2.13. Minimizations I and II take place almost immediately after each other, around a time of 0.36.

Table 2.2: 2D hyperelasticity: Computational performances of the different methods for the different discretizations. Iteration counters include iterations of rejected increments.

Mesh	Method	Newt incr accept	Newt incr reject	Newt iter	Min incr	Min iter	Min force eval	CG iter
5 X 5	BFGS	167	118	1887	7	1137	6408	
	LBFGS100	167	118	1887	7	1240	6811	
	LBFGS50	167	118	1887	7	1582	8361	
	TR	167	118	1887	7	1496	1503	12581
	TR-icho	167	118	1887	7	999	1006	3421
10 X 10	BFGS	133	82	1055	3	1692	10136	
	LBFGS100	133	82	1055	3	1783	9236	
	LBFGS50	132	90	1119	3	2158	10642	
	TR	133	82	1055	3	8807	8810	29104
	TR-icho	133	82	1055	3	2395	2398	3393
15 X 15	BFGS	143	87	1342	4	2521	13237	
	LBFGS100	143	87	1342	4	2998	14309	
	LBFGS50	143	87	1342	4	4000	19534	
	TR	141	87	1302	4	35470	35474	81497
	TR-icho	141	87	1302	4	3856	3860	4511

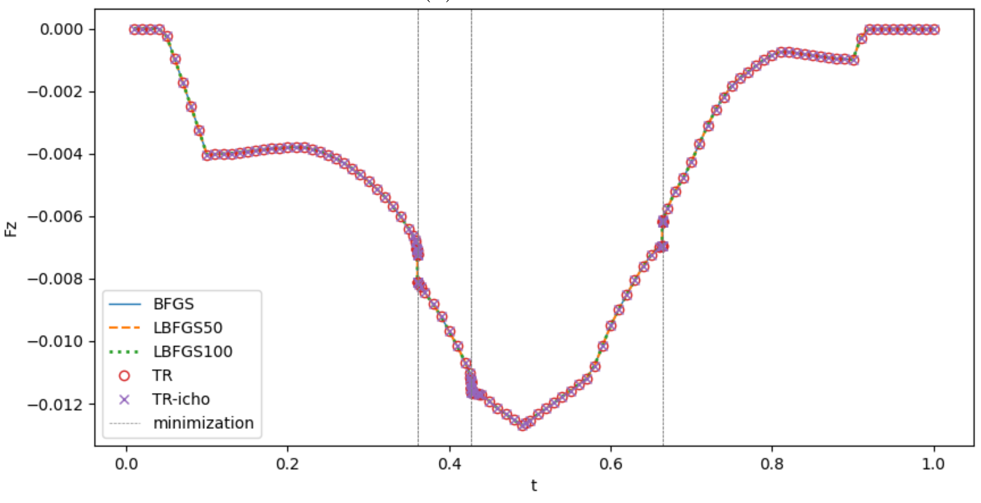
2D elastoplasticity, 5×5 FEs

The same simulations are then repeated, except for the elastoplastic constitutive model. The force-time curves are presented in the diagrams in Fig. 2.16. The diagrams clearly demonstrate that, again, all predictions of the different methods are the same. Furthermore, the force-time responses show a more asymmetric profile due to the history dependence of the elastoplastic constitutive model. This also becomes clear from the deformations that are presented in Fig. 2.17 for different pseudo-times.

Furthermore, the simulations only switch the solver four times, although only three vertical black lines can be distinguished in Fig. 2.16. The deformations before and after the minimization algorithms are applied are presented in Fig. 2.18. Again, it could be possible that some of the increments handled by the minimization solvers could be avoided if a larger penalty value were used for Newton's method with the active set method. However, minimization III looks very similar as minimization III in Fig. 2.15, which also did not converge for Newton's method with an active set approach and penalty values ranging from 10^3 to 10^6. It is furthermore fair to assume that snap-back behaviour occurs for the time increment of minimization IV in Fig. 2.18, which would not converge if Newton's method were applied.

The performance of the different solvers is presented in the first main row of Table 2.3. Of the quasi-Newton methods, the L-BFGS approach with the smallest number of previous iterations taken into account performs the worst. Again, the TR methods are the obvious winner in terms of the number of force vector evaluations - which is even more the bottleneck in elastoplastic simulations. Again, preconditioning also requires time in the preconditioned TR method.

(a) Vertical force



(b) Horizontal force

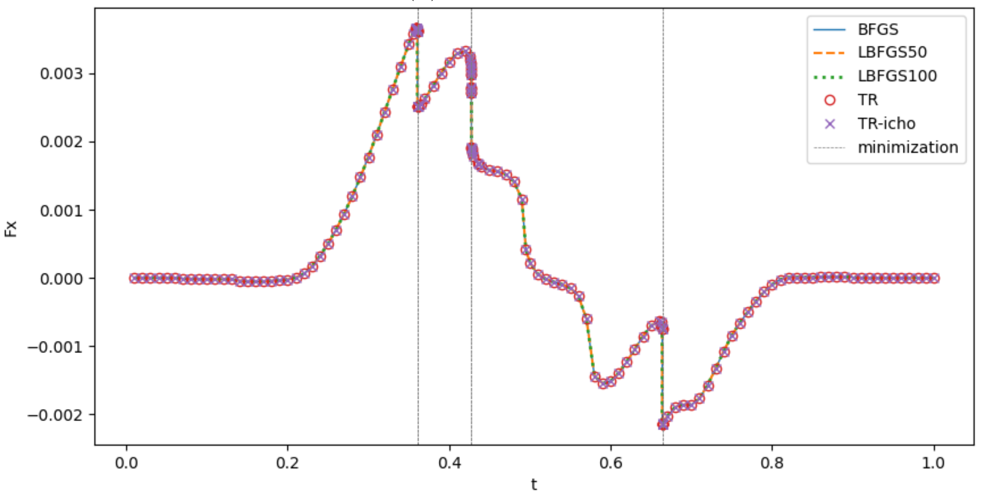


Figure 2.16: 2D hyperelastoplasticity, 5×5 FEs. The force-time curves predicted by the different methods. The times at which a minimization solver is applied are indicated by the black vertical lines. The number behind 'LBFGS' in the legend refers to n_{li}.

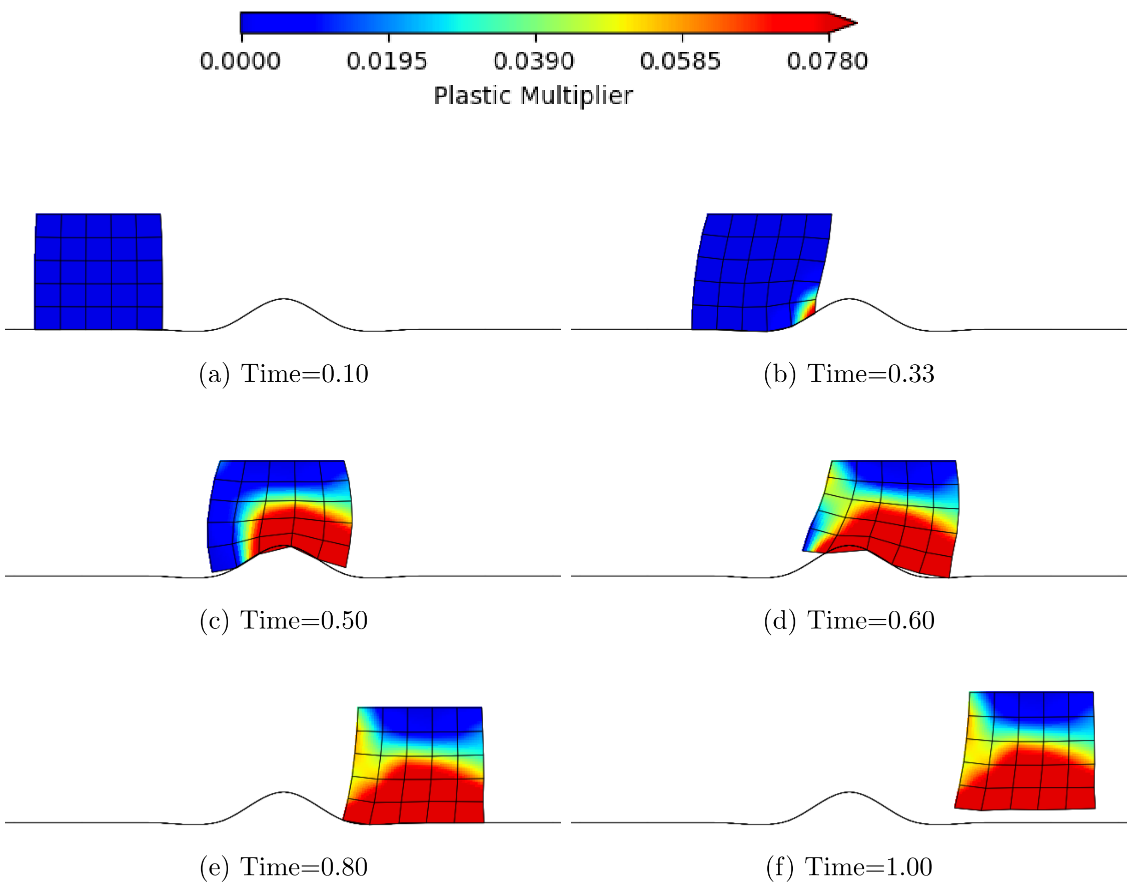


Figure 2.17: 2D hyperelastoplasticity, 5 × 5 FEs. The deformations predicted by all methods for different pseudo-times.

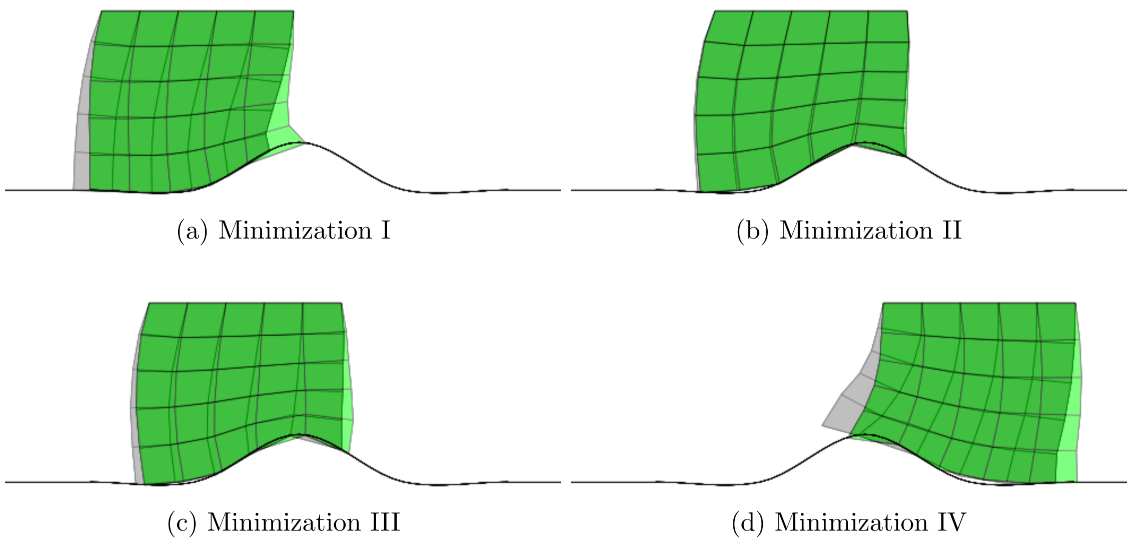


Figure 2.18: 2D hyperelastoplasticity, 5×5 FEs. The deformations before a minimization algorithm is applied (grey) and after a minimization algorithm has converged (green). Thus, the deformations correspond to the vertical black lines in Fig. 2.16.

Table 2.3: 2D hyperelastoplasticity: Computational performances of the different methods for the different discretizations. Red is used for simulations that failed on the cluster, but converged on a local computer. Iteration counters include iterations of rejected increments.

Mesh	Method	Newt incr accept	Newt incr reject	Newt iter	Min incr	Min iter	Min force eval	CG iter
5 X 5	BFGS	149	80	1657	4	686	3935	
	LBFGS100	149	80	1657	4	727	3941	
	LBFGS50	149	80	1657	4	926	5167	
	TR	149	80	1657	4	1128	1132	9108
	TR-icho	149	80	1657	4	661	665	1265
10 X 10	BFGS	122	74	1099	1	578	3472	
	LBFGS100	122	74	1099	1	724	3729	
	LBFGS50	122	74	1099	1	829	4339	
	TR	137	123	1365	3	8807	2964	18599
	TR-icho	137	123	1365	3	1703	1706	2628
15 X 15	BFGS	118	84	1195	1	893	5212	
	LBFGS100	118	84	1195	1	1108	5676	
	LBFGS50	118	84	1195	1	1585	8208	
	TR	118	84	1195	1	3104	3105	13779
	TR-icho	118	84	1302	4	1255	1256	1638

2D hyperelasticity, 10×10 FEs

Next, the focus returns the hyperelastic material behaviour, but the block is now discretized with 10×10 FEs (i.e. $n_{\text{var}} = 220$). The forces predicted by the methods during the course of the simulations are presented in Fig. 2.19. The predicted forces are seemingly the same for all the methods. The simulations switch three times to a minimization algorithm.

Some deformations predicted for different simulation times are presented in Fig. 2.20. They indicate that the block deforms substantially differently than for the 5×5 case. This is particularly pronounced for a pseudo-time of 0.50, since the block is 'stuck' behind the hill at this time instance. Based on the deformation at a pseudo-time of 0.50 and that at a pseudo-time of 0.60, snap-back must happen some time in between. The diagrams in Fig. 2.19 indicate that this occurs just before a pseudo-time of 0.60. The image named 'Minimization II' in Fig. 2.21 shows the snap-back behaviour in detail. The image named 'minimization III' in Fig. 2.21 also illustrates the snap-back that occurs for a pseudo-time of approximately 0.62.

Even though the force-time diagrams do not show much of a discrete force jump at the time of the first minimization, nor does the deformation before and after minimization I in Fig. 2.21, a discrete event does happen at this moment. The issue is that the block's contact moves from its bottom edge (see image b in Fig. 2.20) to its right edge (see minimization I in Fig. 2.21). During this rotation, the bottom left FE undergoes a drastic change of deformation. At some time during this process, this change is sufficiently abrupt to cause a switch of the solver.

Although the forces presented in the diagrams Fig. 2.19 are seemingly the same for all methods, the L-BFGS simulation with $n_{\text{li}} = 50$ behaves slightly differently than the other ones. This can well be observed for the deformations presented in Fig. 2.22. The large deformations in the bottom right corner are introduced during the second time the L-BFGS algorithm is applied. Nevertheless, no deformation occurs such that $J < 10^{-12}$, as this would not be accepted by the solver. A careful consideration has indicated that self-penetration occurs during the L-BFGS minimization. As contact between deformable bodies, such as self-contact, is not implemented, a further investigation can currently not be performed.

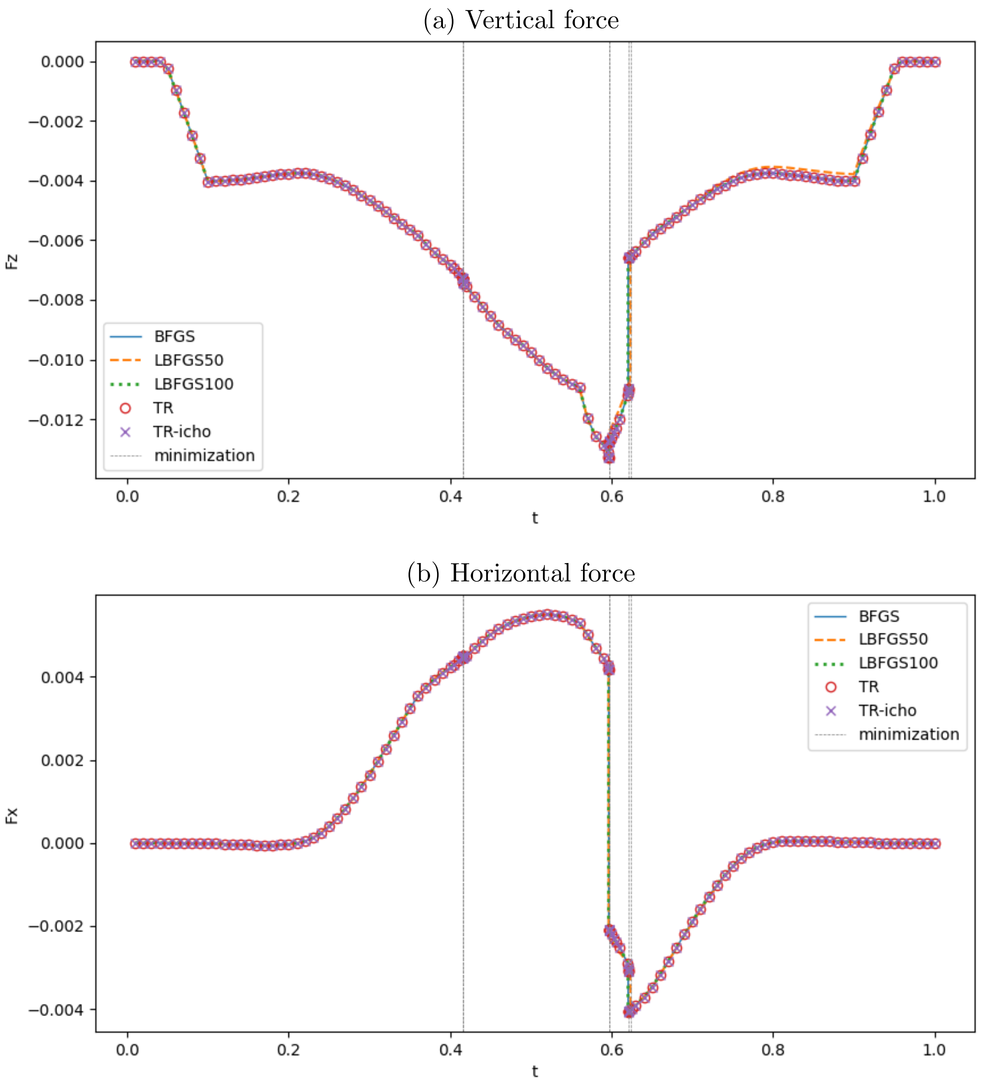


Figure 2.19: 2D hyperelasticity, 10 × 10 FEs. The force-time curves predicted by the different methods. The times at which a minimization solver is applied are indicated by the black vertical lines. The number behind ‘LBFGS’ in the legend refers to n_{li}.

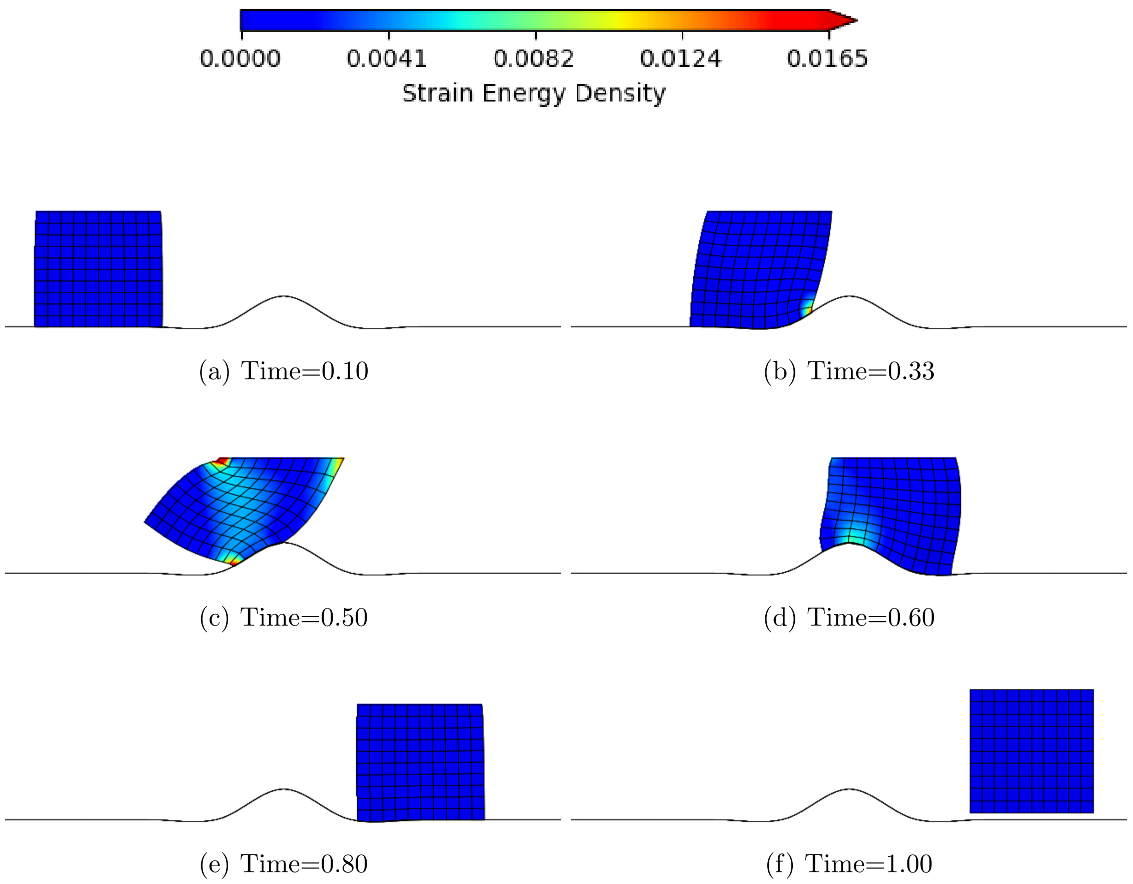


Figure 2.20: 2D hyperelasticity, 10×10 FEs. The deformations predicted by all methods, except for the L-BFGS method with $n_{\mathrm{li}} = 50$, for different pseudo-times.

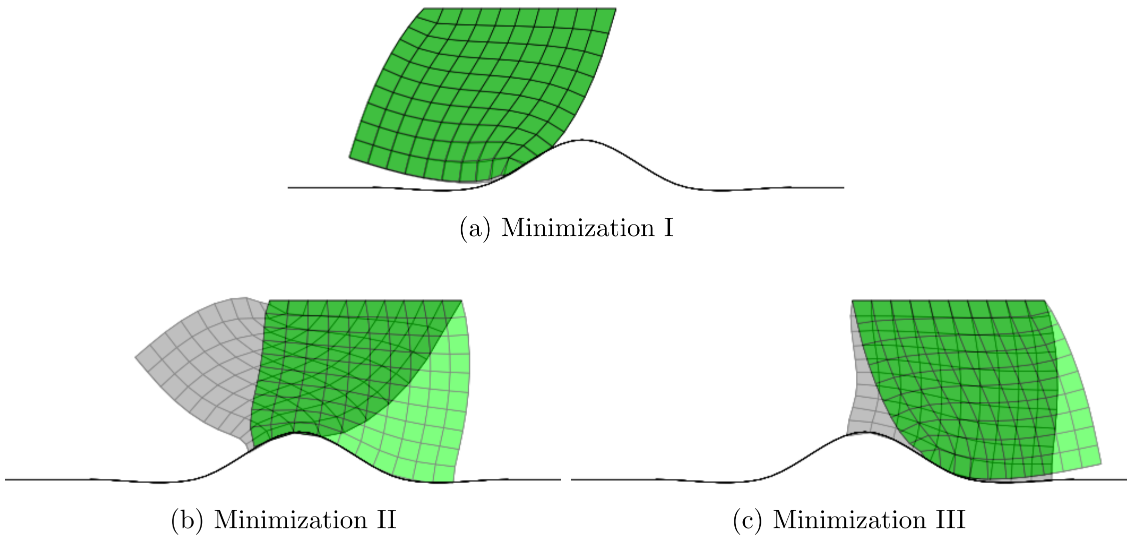
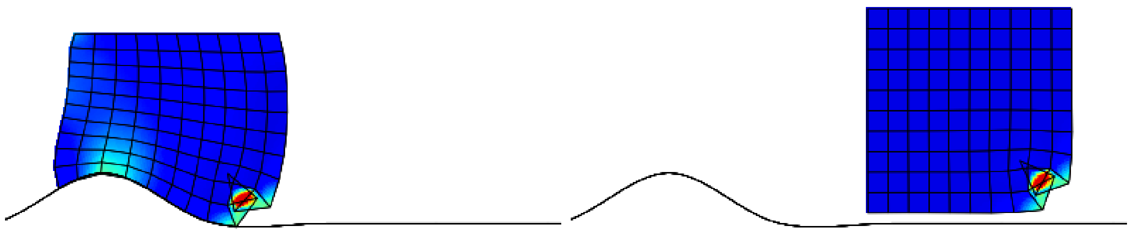


Figure 2.21: 2D hyperelastcity, 10×10 FEs. The deformations before a minimization algorithm is applied (grey) and after a minimization algorithm has converged (green). Thus, the deformations correspond to the vertical black lines in Fig. 2.19.



(a) Time=0.60 (b) Time=1.00

Figure 2.22: 2D hyperelastcity, 10×10 FEs. The deformations predicted by the L-BFGS approach with $n_{\text{li}} = 50$ for two pseudo-times.

The fact that the L-BFGS approach with $n_{\text{li}} = 50$ predicts different results can also be noticed in the second main row of Table 2.2 in which the performance of the different methods are presented for this example. The number of increments successfully and unsuccessfully solved with Newton's method is, for instance, different from those of the other methods, as is the total number of Newton iterations. Nevertheless, it can hardly be noticed in the simulation time, which is governed by the number of times that the column with force vectors must be evaluated.

In contrast to the previous simulations, in which both TR methods performed rather similarly and substantially faster than the quasi-Newton methods, only the preconditioned TR method is the clear winner in terms of the number of gradient/force evaluations for the current test case.

2D hyperelastoplasticity, 10×10 FEs

The next set of simulations is the same as the previous set, except that elastoplasticity is considered. The forces predicted by the different methods over the course of the simulations are presented in Fig. 2.23. The quasi-Newton simulations only switch solvers once. Both TR methods make different predictions than the quasi-Newton methods and switch solvers three times. Snapshots of the deformations predicted by the quasi-Newton methods for different pseudo-times are presented in Fig. 2.24. They seem physically correct at first sight.

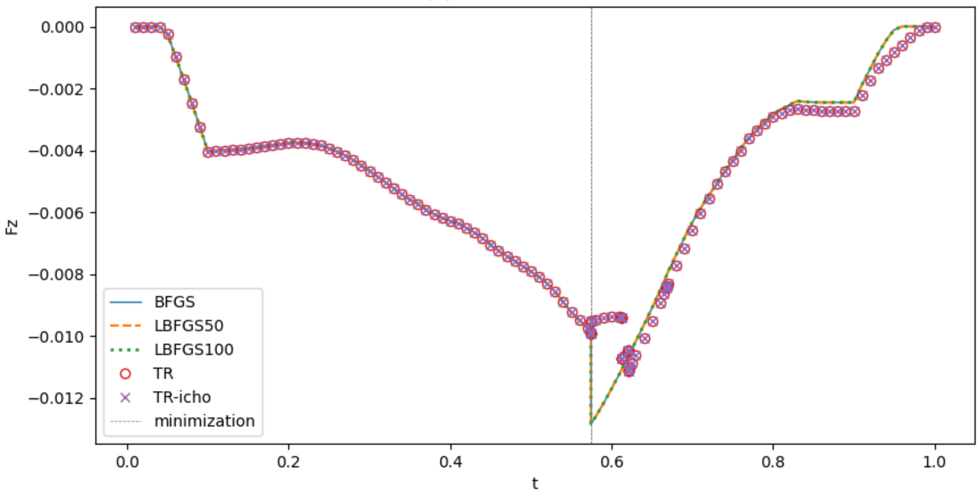
Two of the deformations predicted by the TR methods are presented in Fig. 2.25. The deformation at a pseudo-time of 0.60 (just after the first minimization is performed) is noticeably non-physical because some of the internal vertices penetrate the leader-surface. This occurs because the line segment between the follower-vertex at the bottom right corner and its left neighbour is substantially stretched by the leader-surface.

For this reason, an additional preconditioned TR simulation is performed in which not only edge vertices serve as follower-vertices, but also internal vertices. A snapshot of a non-converged deformation predicted by the preconditioned TR method is presented on the right in Fig. 2.26. The left image of Fig. 2.26 presents its counterpart for the conventional simulation in which only edge vertices serve as follower-vertices for the same global TR iteration. In fact, internal vertices do not penetrate the leader-surface (right image of Fig. 2.26). However, this change does not avoid the non-physical prediction made by the preconditioned TR method (not shown).

It is furthermore worth mentioning that all simulations of this thesis are performed on the cluster of the University of Luxembourg, but the two TR simulations for this test case fail on the cluster. However, they 'succeed' if performed on a local computer. This highlights the relative fragility of the two TR simulations of interest.

The computational performances of the different methods are presented in the second main row of Table 2.3 for this test case. Of the three quasi-Newton methods, the standard one requires the least minimization iterations and, more importantly, the least force/gradient evaluations. This was also observed for the 5×5 discretizations. Although the simulations with both TR methods require more increments to be handled by the minimization solvers, they require fewer force evaluations than all quasi-Newton methods. The preconditioned TR method is again the clear winner in terms of the number of force evaluations.

(a) Vertical force



(b) Horizontal force

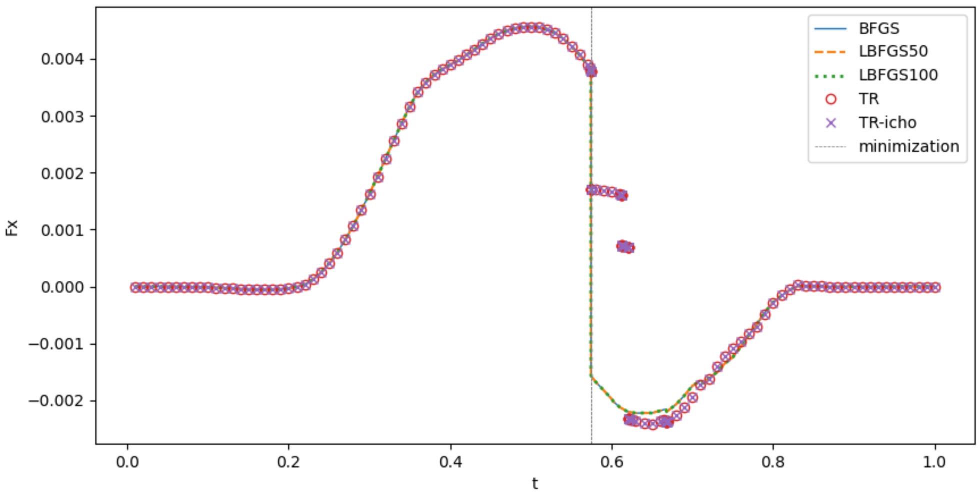
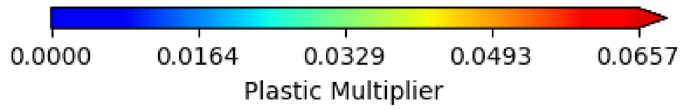
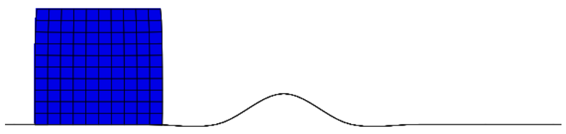
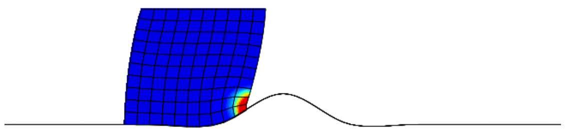
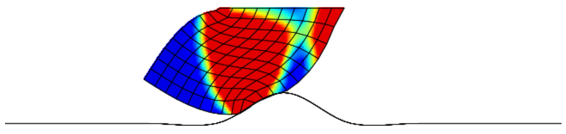


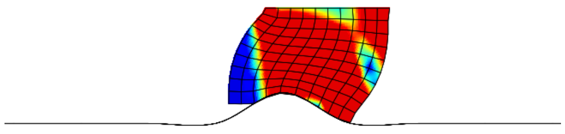
Figure 2.23: 2D hyperelastoplasticity, 10×10 FEs. The force-time curves predicted by the different methods. The time at which a quasi-Newton solver is applied is indicated by the black vertical lines. Simulations employing the TR solvers switch solvers three times, of which the first time correpond to the vertical black line. The other two times are not indicated. The number behind 'LBFGS' in the legend refers to n_{li}.

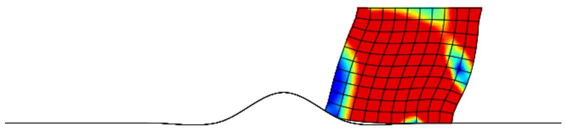



(a) Time=0.10


(b) Time=0.33


(c) Time=0.50


(d) Time=0.60


(e) Time=0.80

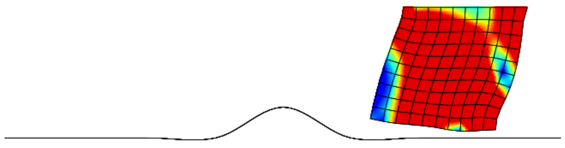
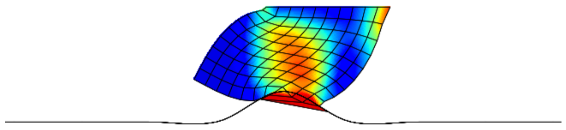

(f) Time=1.00

Figure 2.24: 2D hyperelastoplasticity, 10×10 FEs. The deformations predicted by quasi-Newton methods for different pseudo-times.


(a) Time=0.60

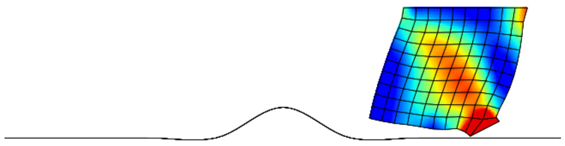
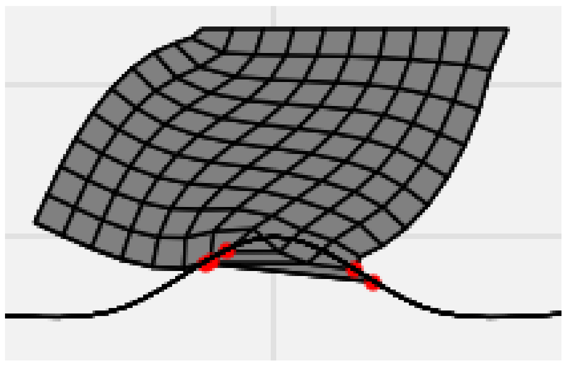

(b) Time=1.00

Figure 2.25: 2D hyperelastoplasticity, 10×10 FEs. Two deformations predicted by the TR methods.


(a) Conventional approach

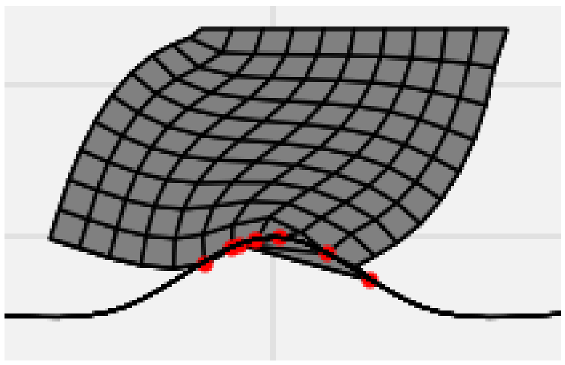
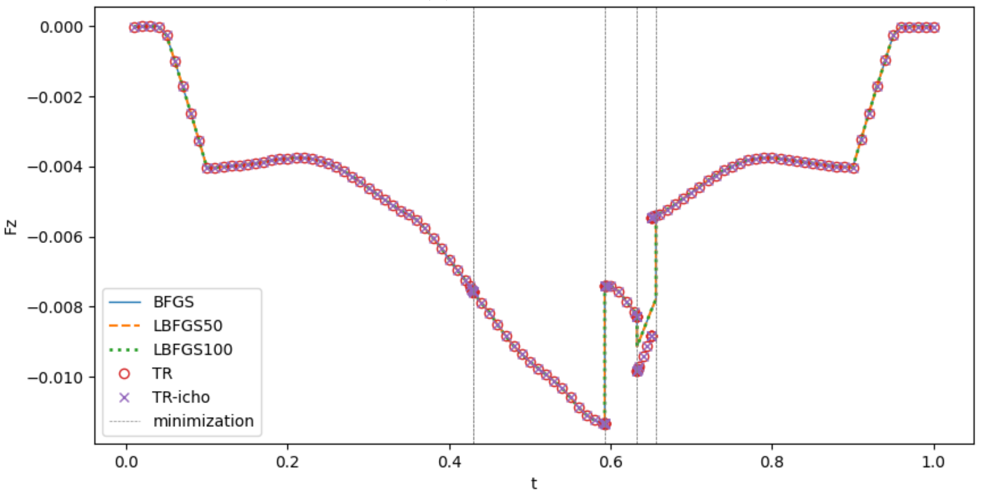
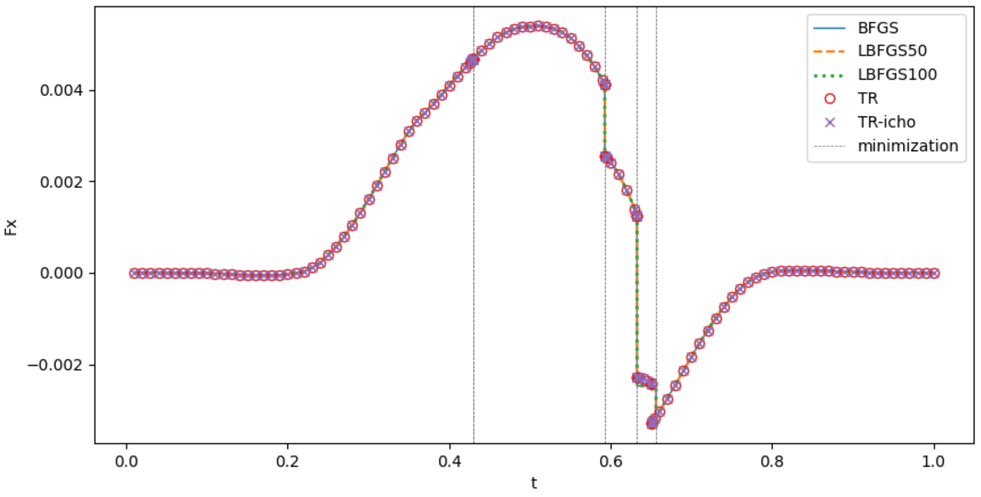

(b) Follower-vertices include inner vertices

Figure 2.26: 2D hyperelastoplasticity, 10×10 FEs. Two non-converged deformations predicted by the preconditioned TR method for the same global TR iteration. (a) The regular approach in which only surface-vertices serve as follower-vertices, (b) all vertices serve as follower-vertices. Red dots: vertices in contact.

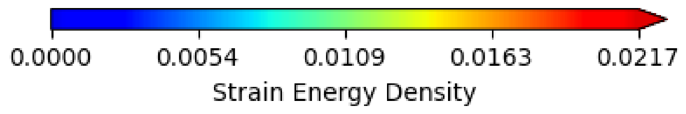
(a) Vertical force

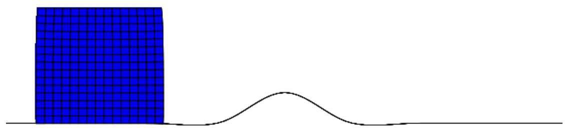


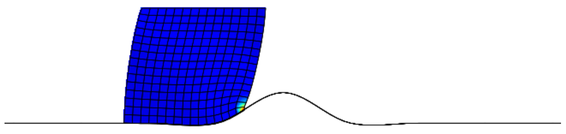


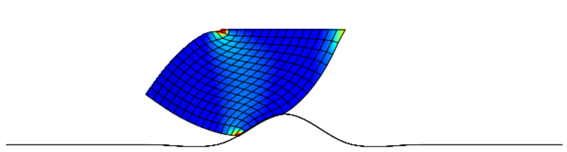
(b) Horizontal force

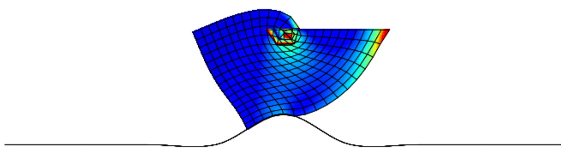
Figure 2.27: 2D hyperelasticity, 15 × 15 FEs. The force-time curves predicted by the different methods. The times at which a minimization solver is applied are indicated by the black vertical lines. The number behind 'LBFGS' in the legend refers to n_{li}.

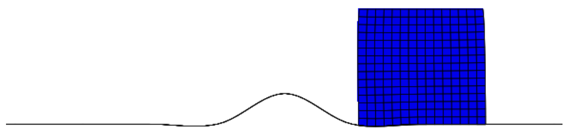



(a) Time=0.10


(b) Time=0.33


(c) Time=0.50


(d) Time=0.60


(e) Time=0.80

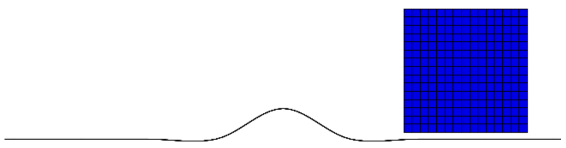
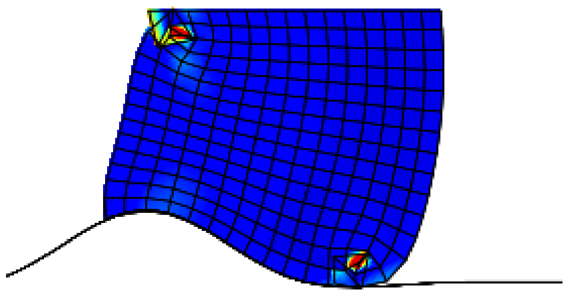

(f) Time=1.00

Figure 2.28: 2D hyperelasticity, 15 × 15 FEs. The deformations predicted by the quasi-Newton methods for different pseudo-times.


(a) Quasi-Newton methods

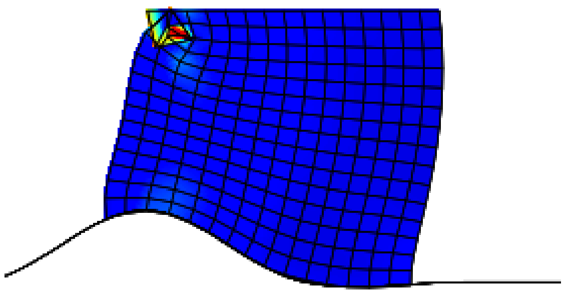

(b) TR methods

Figure 2.29: 2D hyperelasticity, 15 × 15 FEs. The deformations predicted by the quasi-Newton methods (a) and the TR methods (b) at a pseudo-time of 0.64.

2D hyperelasticity, 15×15 FEs

The next set of simulations considers elasticity and a discretization of 15×15 FEs ($n_{\text{var}} = 480$). The forces predicted by the different methods during the course of the simulations are presented in Fig. 2.27. They show that all simulations switch four times to a minimization solver during the course of a simulation. Again, the results predicted by the quasi-Newton methods differ from those predicted by the TR methods for a short time interval.

A visual inspection of the deformations predicted by the simulations using quasi-Newton solvers in Fig. 2.28 yields the conclusion that the quasi-Newton simulations have made rather acceptable predictions, except for the region of the top left corner (see image (d) for a pseudo-time of 0.60). However, it could very well have been the case that if a horizontally straight rigid body had been included that moves along with the top edge of the block, these large deformations would not have occurred. For this reason, the large deformations in the upper left corner are considered insufficiently interesting for further analysis.

More interesting are the deformations predicted for a pseudo-time of 0.64 in Fig. 2.29. They show that for this set of simulations, the TR methods (right image) predict better results than the quasi-Newton methods (left image). This is in contrast to the previous set of simulations performed for the elastoplastic block discretized by 10×10 FEs, since the results of the quasi-Newton methods were deemed physically more accurate.

The computational performances of the different methods are presented in the third main row of Table 2.2. They show that thanks to the better predictions of the TR methods, Newton's method had to work less for the associated simulations. It is also clearly visible that the BFGS method again required the least gradient evaluations of all quasi-Newton methods and the L-BFGS method with $n_{\text{li}} = 50$ the most. Furthermore, the preconditioned TR method is again the clear winner based on the number of force evaluations. The interesting thing is that the original TR method requires, by far, the most force evaluations.

2D hyperelastoplasticity, 15×15 FEs

The last set of 2D simulations is the same as the previous set except that elastoplasticity instead of elasticity is considered. The predicted force-time curves are presented in Fig. 2.30. The diagrams indicate that all methods make the same predictions and that they only switch once to their minimization algorithm. Some deformations predicted for different pseudo-times can be inspected in Fig. 2.31.

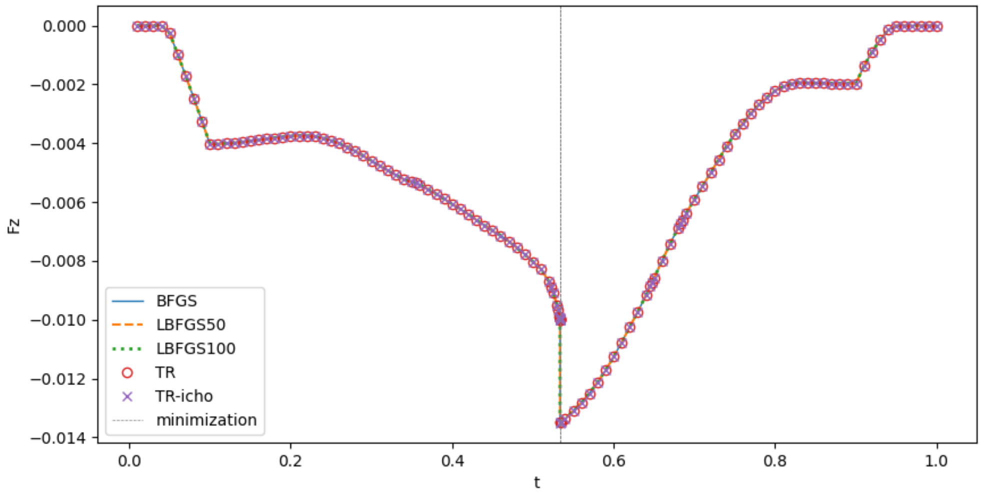
2.5.2 3D results

This subsection reports the capabilities and performances of the methods for 3D scenarios. Because the 3D results are substantially less trivial to interpret, the discussion and analysis of the 3D results are less in-depth than that of the 2D results.

The rigid leader-body in Fig. 2.32 is used in all scenarios. The figure indicates that its generation consists of three steps. Two types of contact simulation are considered as presented in Fig. 2.33. A cubic deformable follower-body is brought in contact with the rigid leader-body.

In the simulations of type I, the follower-body approaches the leader-body from the bottom and then moves in the positive y-direction. The displacement vectors of all vertices on the block's bottom edge are prescribed during the entire simulation, which is initially subdivided in 100 time increments. During the first 40% of the simulation, the block is displaced in the positive z-direction with a magnitude of 1.50. During the last 70% of the simulation, the block is displaced in positive y-direction with a magnitude of 5.00. Thus, between pseudo-times of 0.30 and 0.40, the block displaces in the y-direction and the z-direction.

(a) Vertical force



(b) Horizontal force

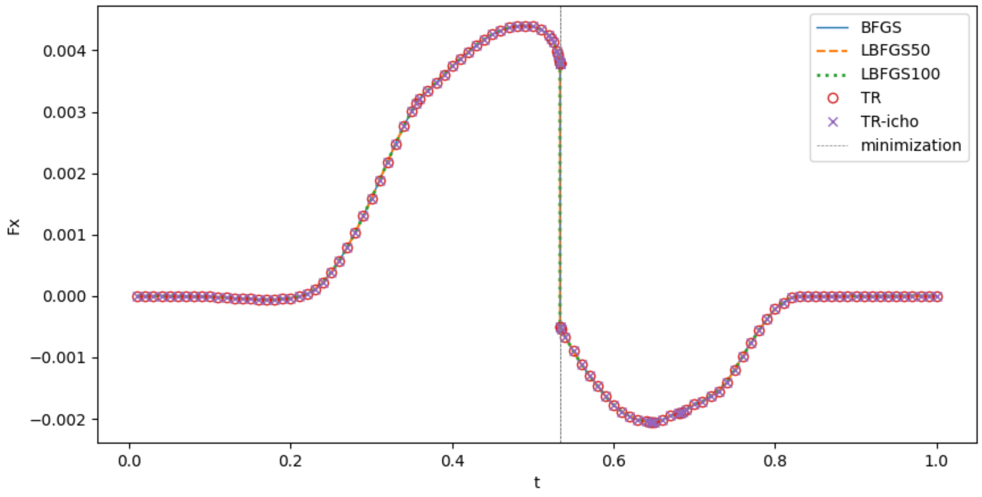
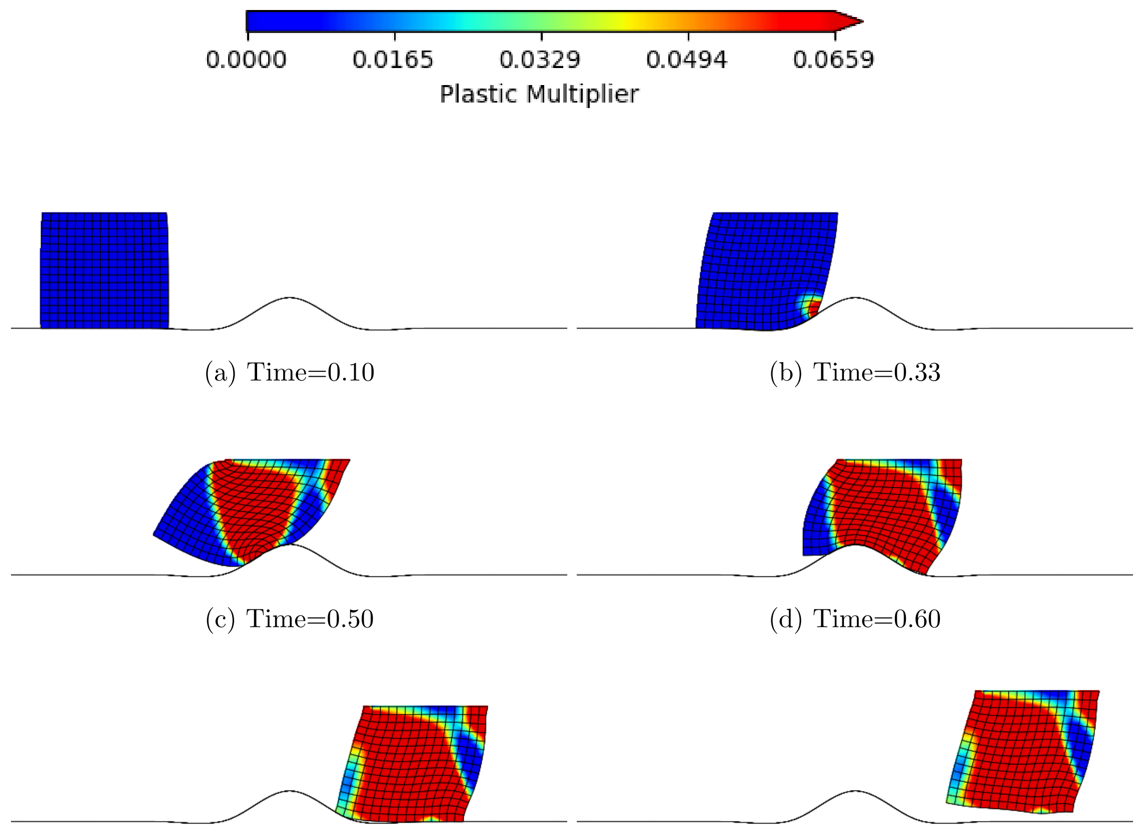


Figure 2.30: 2D hyperelastoplasticity, 15 × 15 FEs. The force-time curves predicted by the different methods. The times at which a minimization solver is applied are indicated by the black vertical lines. The number behind 'LBFGS' in the legend refers to n_{li}.



(a) Time=0.10

(b) Time=0.33

(c) Time=0.50

(d) Time=0.60

(e) Time=0.80

(f) Time=1.00

Figure 2.31: 2D hyperelastoplasticity, 15 × 15 FEs. The deformations predicted by all methods for different pseudo-times.

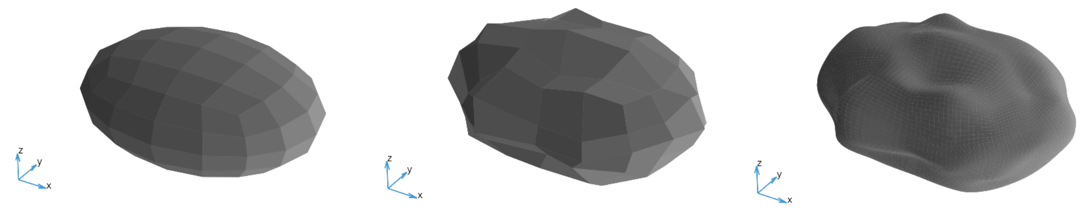


Figure 2.32: Leader-body. Left: ellipsoid with semi-axes of lengths 4.00, 3.00, and 2.00. Middle: ellipsoid with random perturbations of the vertices. Right: leader-surface smoothened using Gregory patches.

In the simulations of type II, the follower-body approaches the leader-body from the top and then moves in the positive x-direction. The displacement vectors of all vertices of the block's top edge are prescribed during the simulation, which is again initially subdivided in 100 time increments. During the first 40% of the simulation, the block is displaced in the negative z-direction with a magnitude of 1.50. During the last 70% of the simulation, the block is displaced in positive x-direction with a magnitude of 6.00. Thus, between pseudo-times of 0.30 and 0.40, the block displaces in the x-direction and the z-direction.

Originally, the intention was to investigate the capabilities of all methods for cubes with similar discretizations as the blocks in the 2D scenarios, i.e. 5 × 5 × 5, 10 × 10 × 10 and 15 × 15 × 15 FEs. However, the minimization methods failed for a substantial part of the simulations with discretizations consisting of 15 × 15 × 15 FEs. On the one hand, this is somewhat surprising because all minimization algorithms performed very similarly for the 2D simulations with the finest discretization. Nevertheless, an investigation in

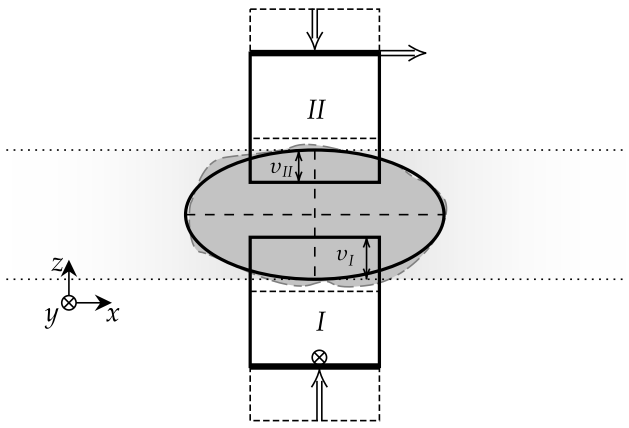


Figure 2.33: Sketches of the two types of 3D simulations. The cubic deformable follower-body of size $4.00 \times 4.00 \times 4.00$ is presented twice and indicated by Roman numerals I and II. The ellipsoidal leader-body is presented in grey. $v_I = 1.20$ and $v_{II} = 1.00$ denote the maximal overlaps in z-direction that occurs during the two types of simulations.

the manner in which the 3D simulations with the finest discretization failed indicates that the cause is simply that the force/gradient evaluations are too polluted by machine precision. This may indicate that more advanced methods are required if large numbers of FEs occur. These could, for instance, be domain decomposition methods [100, 101] or multigrid methods [102, 103].

It is also worth mentioning that the number of previous iterations incorporated in the L-BFGS method is furthermore increased to $n_{\text{li}} = 500$, because the 3D simulations contain more variables.

3D hyperelasticity, Type I, $5 \times 5 \times 5$ FEs

The first set of simulations discussed are of type I and discretize the block with $5 \times 5 \times 5$ FEs ($n_{\text{var}} = 540$). The forces predicted by the different methods for the elastic constitutive model are presented in Fig. 2.34. The simulations exploiting the quasi-Newton solvers require to switch solvers four times, whereas those using the TR methods switch solvers five times. For a short interval, which follows directly after the fourth minimization, the forces predicted by the TR methods are different from those predicted by the quasi-Newton methods. The interesting thing is that the TR methods predict an increase of the force in the x-direction, whereas the quasi-Newton methods predict a decrease of the same force. After a short time, the predictions of all simulations are the same again.

Some deformations predicted by the quasi-Newton methods for different time instances are presented in the left column of Fig. 2.36 and show substantial deformations.

The computational performances of the different methods are presented in the first main row of Table 2.4. Although the simulations exploiting the TR methods require more Newton increments and one additional minimization increment, they clearly beat the quasi-Newton methods in terms of the number of force evaluations, with the precondtioned TR method as the overall winner.

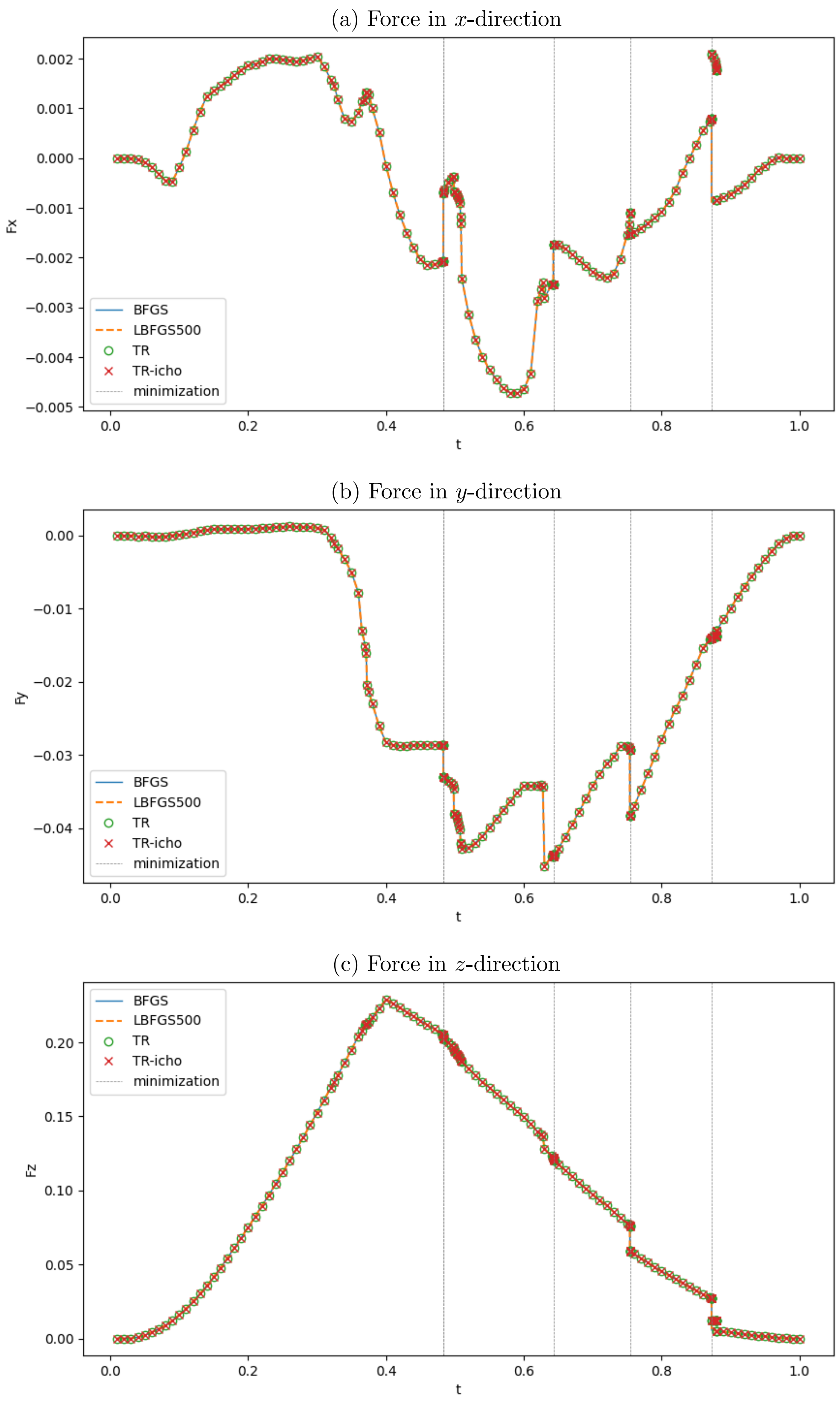
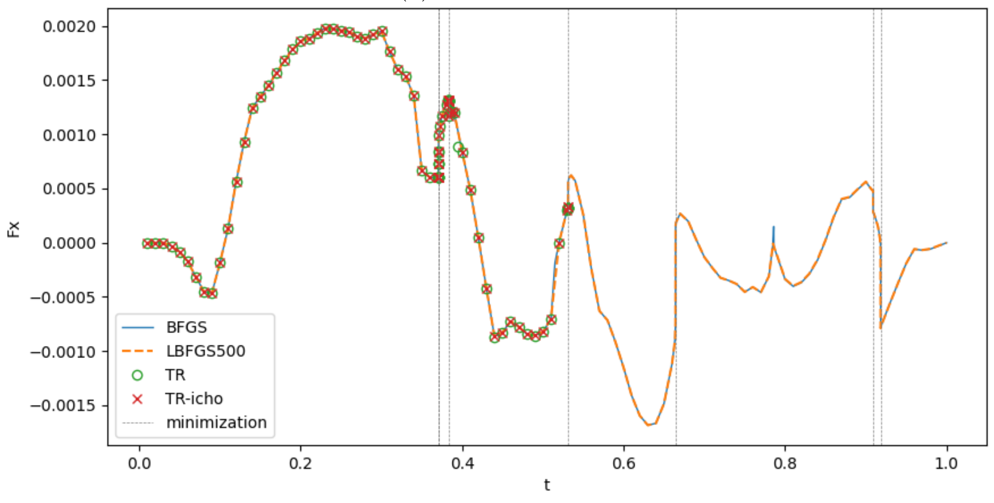
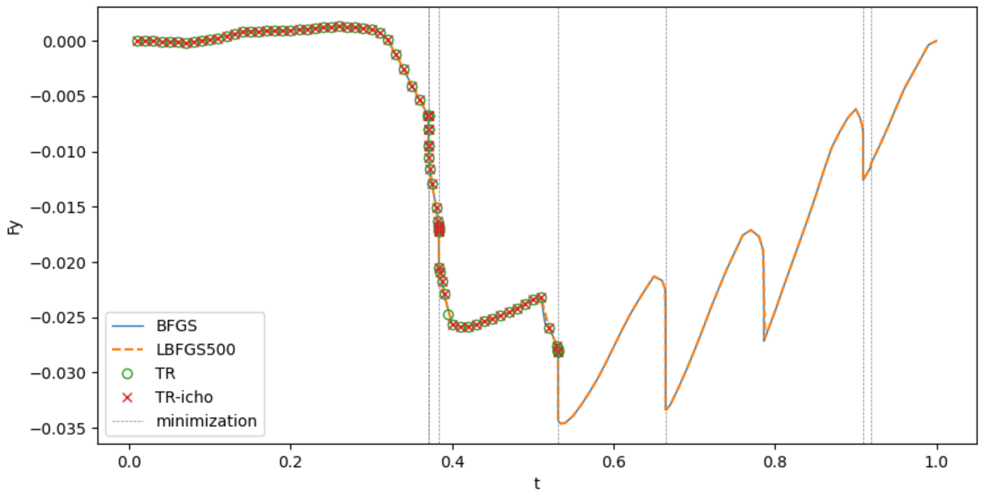


Figure 2.34: 3D hyperelasticity, simulation type I, $5 \times 5 \times 5$ FEs. The force-time curves predicted by the different methods. The times at which a minimization solver is applied in the quasi-Newton simulations are indicated by the black vertical lines. The number behind 'LBFGS' in the legend refers to n_{li}.

(a) Force in x-direction



(b) Force in y-direction



(c) Force in z-direction

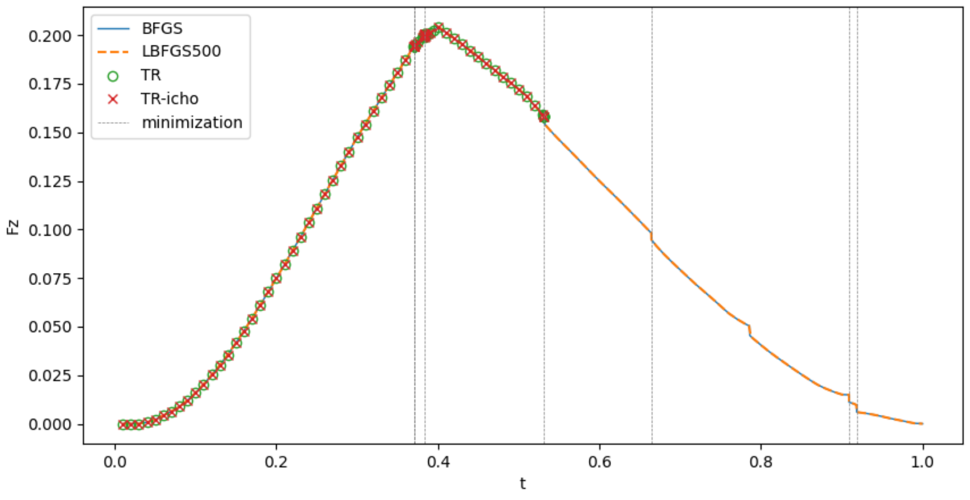


Figure 2.35: 3D hyperelastoplasticity, simulation type I, $5 \times 5 \times 5$ FEs. The force-time curves predicted by the different methods. The times at which a minimization solver is applied in the quasi-Newton simulations are indicated by the black vertical lines. The number behind 'LBFGS' in the legend refers to n_{li}.

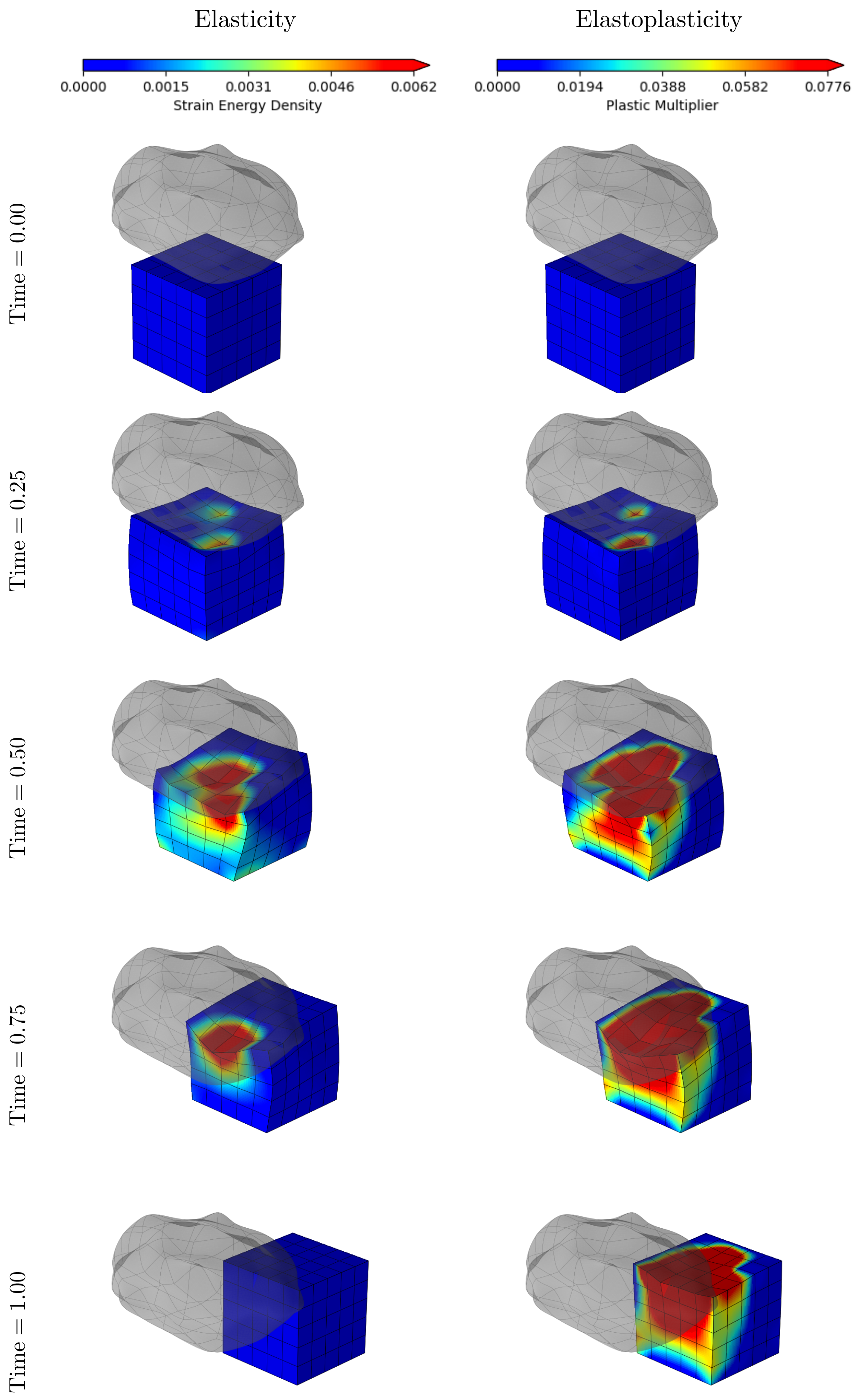


Figure 2.36: 3D simulation type I, 5 × 5 × 5 FEs. Some deformations predicted by the simulations that use the BFGS method.

Table 2.4: 3D simulation type I, $5\times5\times5$ FEs. Computational performances of the different methods. Iteration counters include iterations of rejected increments.

Method	Newt incr accept	Newt incr reject	Newt iter	Min incr	Min iter	Min force eval	CG iter
Hyperelasticity							
BFGS	158	159	2230	4	2340	13194	
LBFGS500	158	159	2231	4	2166	11846	
TR	167	169	2463	5	5428	5433	16716
TR-icho	167	169	2463	5	2368	2373	2932
Hyperelastoplasticity							
BFGS	174	158	2898	8	3218	19876	
LBFGS500	164	140	2660	7	2754	16381	

3D hyperelastoplasticity, Type I, $5 \times 5 \times 5$ **FEs**

Only the constitutive model is switched from the elastic model to the elastoplastic one for the next set of simulations. The forces predicted by the different methods are presented in Fig. 2.35 and some deformations in the right column of Fig. 2.36. The force-time diagrams show six instances in which minimization takes place, but more than one occurs in the first, third, and last instances. The simulations exploiting the TR methods did not converge for the minimization at the third instance. Furthermore, the simulation exploiting the standard TR method predicts a slightly different result for the second instance in which minimization is applied. This fact becomes clearer directly after the second pseudo-time that minimization is applied.

Moreover, the two quasi-Newton simulations also make somewhat different predictions. This is, for instance, visible in the force in the x-direction at a pseudo-time of approximately 0.78. It can also be observed in the second main row of Table 2.4 by the different numbers of successfully and unsuccessfully performed Newton increments, by the different numbers of Newton iterations and the different number of times the solver switched to the minimization algorithm. Exceptionally, the L-BFGS method required fewer force evaluations than the BFGS approach.

3D hyperelasticity, Type I, $10 \times 10 \times 10$ **FEs**

Next, the same simulations are carried out, but with the elastic material model and a discretization of $10 \times 10 \times 10$ FEs ($n_{\text{var}} = 3630$). The predicted forces over the course of the simulations are presented in the diagrams of Fig. 2.37, whilst some deformations predicted by the simulation utilizing the BFGS approach are presented in the left column of Fig. 2.39.

The force diagrams of Fig. 2.37 clearly indicate that towards the end of the simulation, the quasi-Newton methods and the TR methods predict different results for about 5% of the simulation. With the help of additional minimizations, the results predicted by the TR simulations match those predicted by the quasi-Newton methods again for a pseudo-time of approximately 0.82. This is obviously only possible because the simulation considers hyperelasticity and frictionless contact.

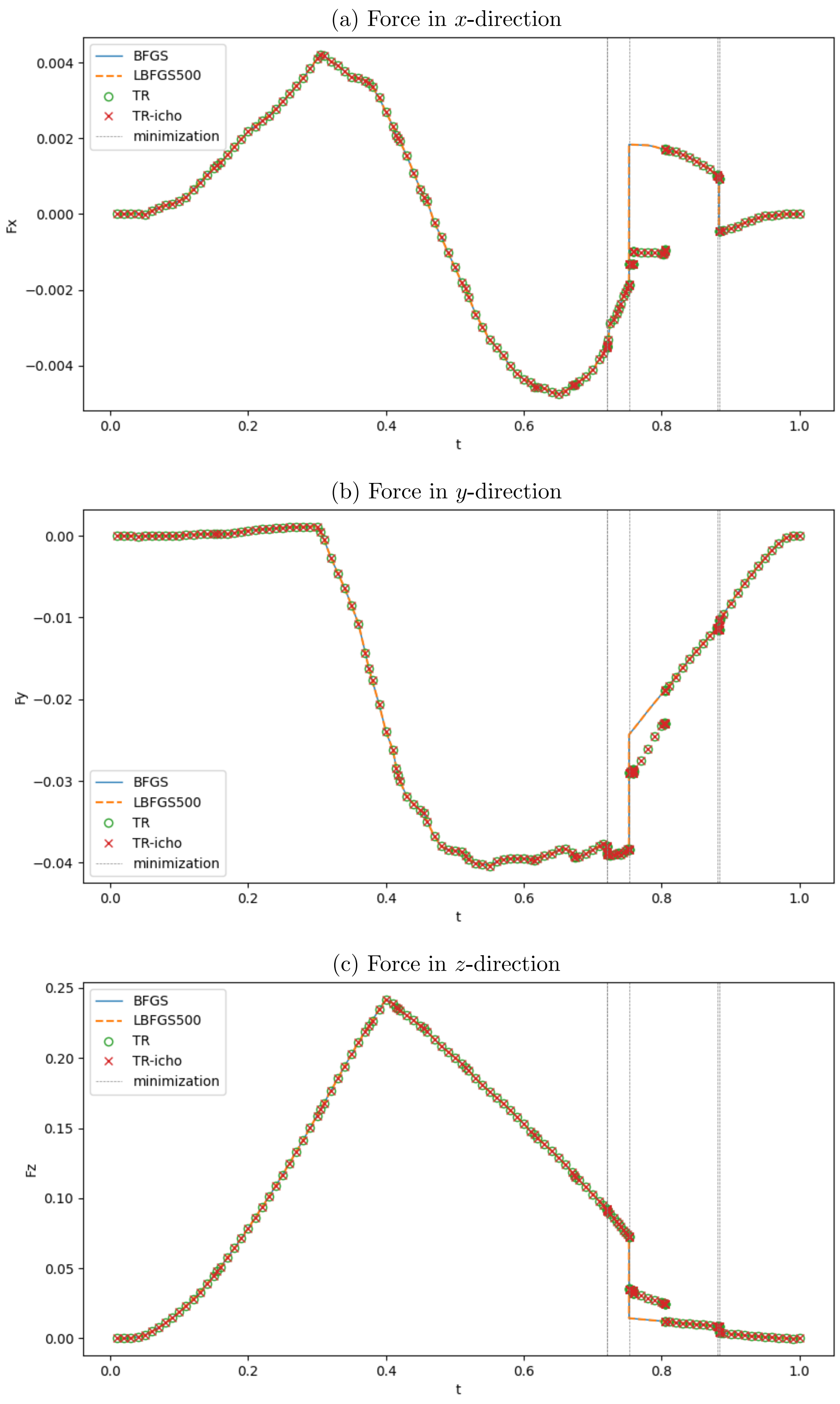
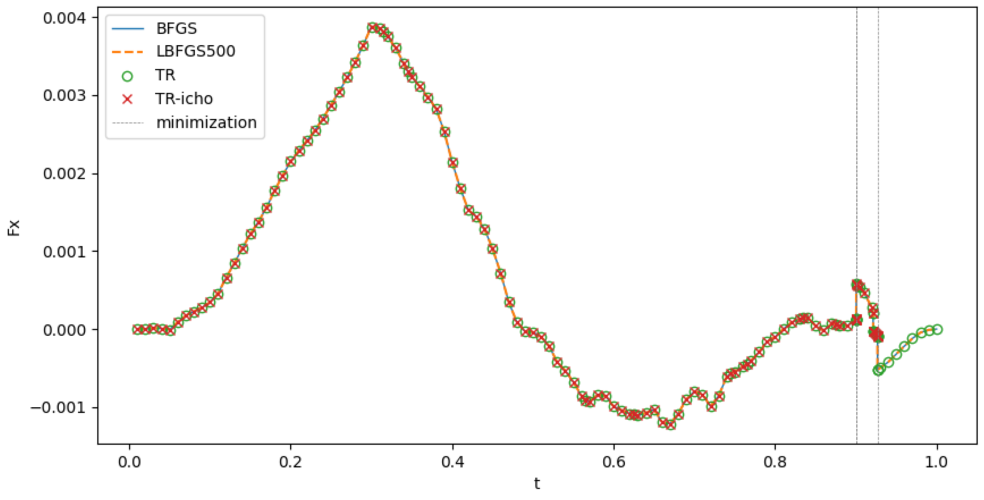
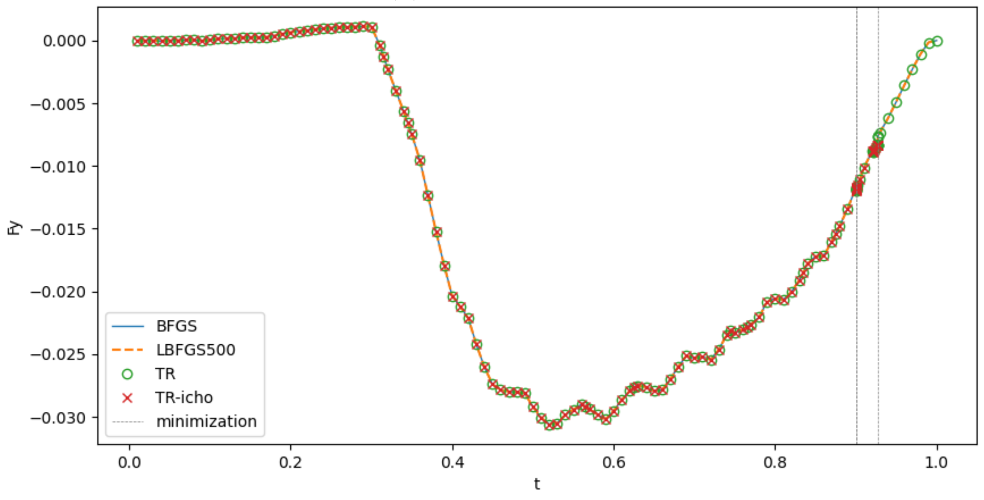


Figure 2.37: 3D hyperelasticity, simulation type I, $10 \times 10 \times 10$ FEs. The force-time curves predicted by the different methods. The times at which a minimization solver is applied in the quasi-Newton simulations are indicated by the black vertical lines. The number behind ‘LBFGS’ in the legend refers to n_{li}.

(a) Force in x-direction



(b) Force in y-direction



(c) Force in z-direction

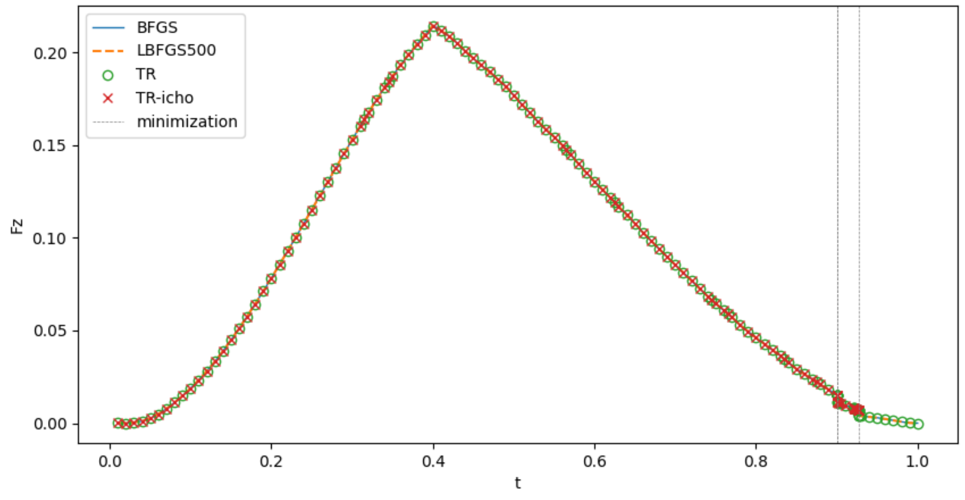


Figure 2.38: 3D hyperelastoplasticity, simulation type I, $10 \times 10 \times 10$ FEs. The force-time curves predicted by the different methods. The times at which a minimization solver is applied in the quasi-Newton simulations are indicated by the black vertical lines. The number behind ‘LBFGS’ in the legend refers to n_{li}.

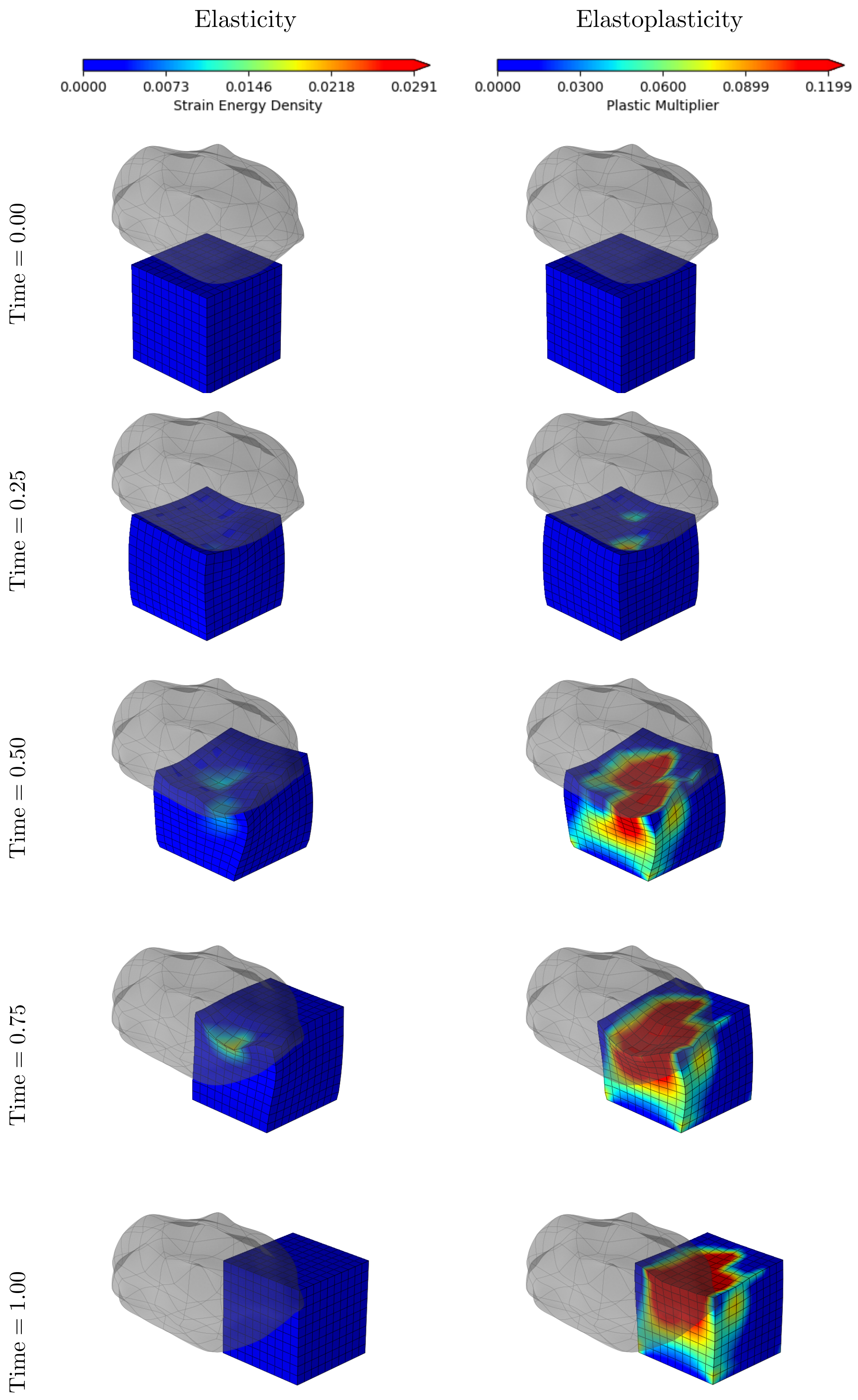


Figure 2.39: 3D simulation type I, $10 \times 10 \times 10$ FEs. Some deformations predicted by the simulations that use the BFGS method.

Table 2.5: 3D simulation type I, $10 \times 10 \times 10$ FEs. Computational performances of the different methods. Iteration counters include iterations of rejected increments.

Method	Newt incr accept	Newt incr reject	Newt iter	Min incr	Min iter	Min force eval	CG iter
Hyperelasticity							
BFGS	154	216	2910	6	5019	28784	
LBFGS500	154	215	2911	5	5033	27100	
TR	167	244	3498	8	5428	20465	81473
TR-icho	173	242	3449	7	6475	6482	9516
Hyperelastoplasticity							
BFGS	134	163	2337	2	1992	10854	
LBFGS500	134	163	2337	2	2008	10585	
TR	134	163	2338	2	2441	2443	19529

Although simulations using the TR methods require the optimization solver more often than those using quasi-Newton methods, as can be seen in the first main row of Table 2.5, they still require less force/gradient evaluations than the quasi-Newton simulations. However, it is also clear to see that the preconditioned TR simulation is the obvious winner in terms of the number of force/gradient evaluations.

3D hyperelastoplasticity, Type I, $10 \times 10 \times 10$ FEs

The only change in the next set of simulations compared to the previous one is that the constitutive model is adjusted from hyperelastic to hyperelastoplastic. The predicted forces are presented in the diagrams in Fig. 2.38. The diagrams show that all simulations required the switch of solvers twice and that the preconditioned TR method was unable to converge for the second instance in which it was applied.

All curves in Fig. 2.38 match each other furthermore. This entails that the deformations in the right column of Fig. 2.39 are the same for all methods (except the final one that could not be predicted by the preconditioned TR method).

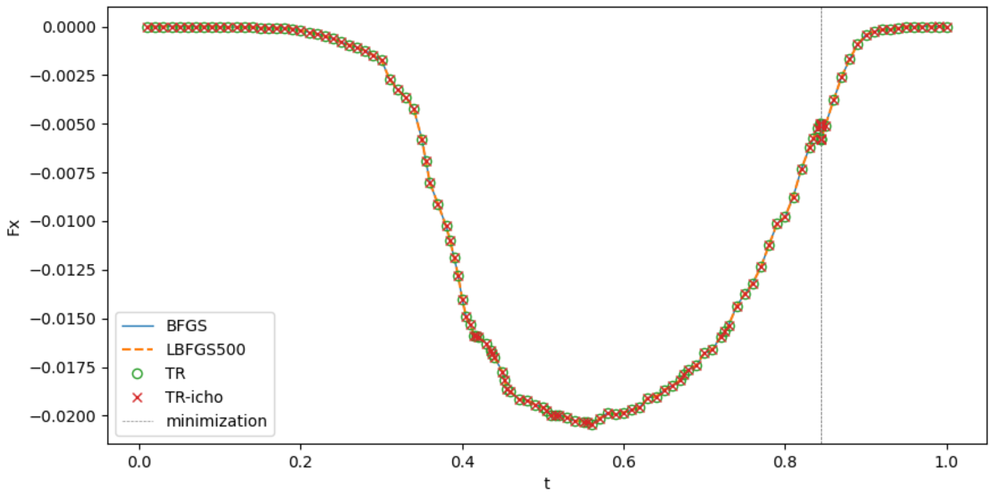
Moreover, the number of force evaluations of the different methods in the second main row of Table 2.5 shows that both quasi-Newton methods performed similarly and that the standard TR method requires approximately four times fewer gradient evaluations than the quasi-Newton methods.

3D hyperelasticity, Type II, $10 \times 10 \times 10$ FEs

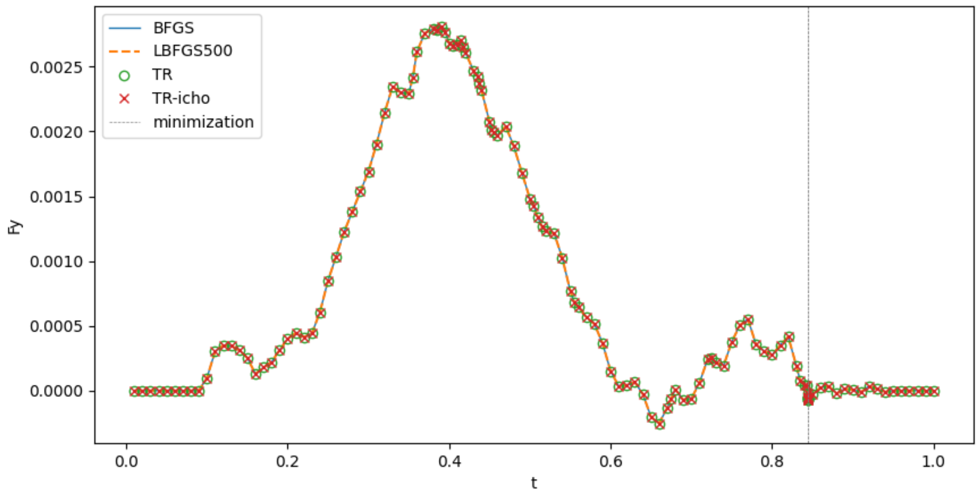
Next, simulations of type II are analyzed. In these simulations, the deformable cube approaches the rigid leader-body from above, and the amount of contact pressure is slightly lower. Because the elastic and elastoplastic simulations with a discretization of $5 \times 5 \times 5$ FEs did not require to switch solvers, they are not considered below.

First, hyperelasticity is considered. All simulations predicted the same results as can be seen in the force-time diagrams in Fig. 2.40. It also clear that the solver was switched only once. Some deformations predicted during the simulation process are presented in the left column of Fig. 2.42.

(a) Force in x-direction



(b) Force in y-direction



(c) Force in z-direction

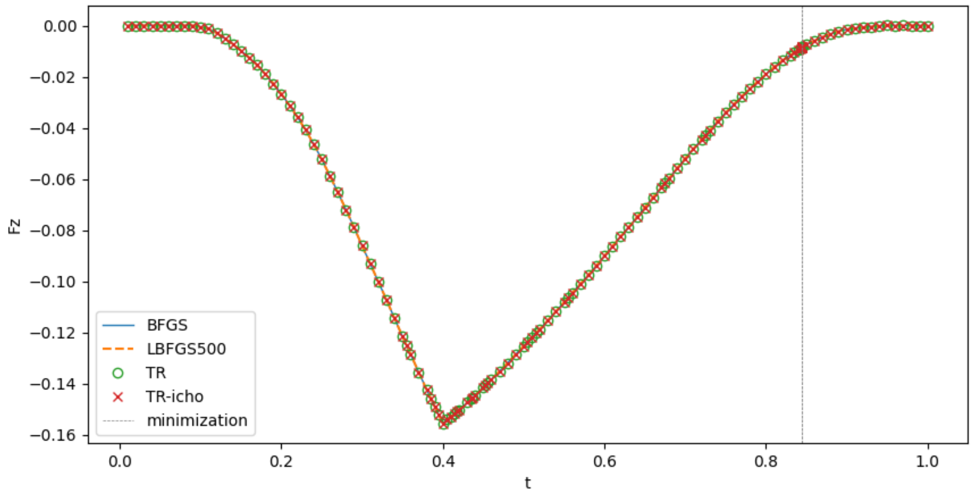
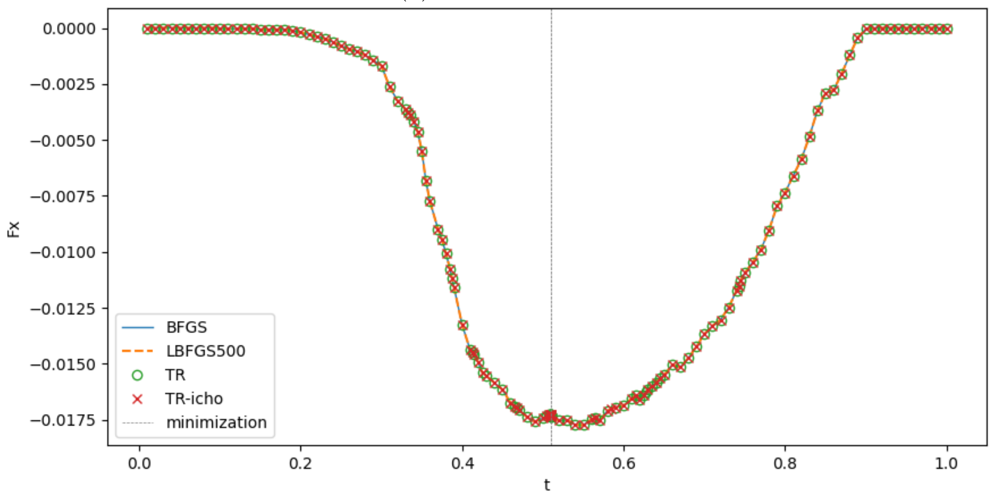
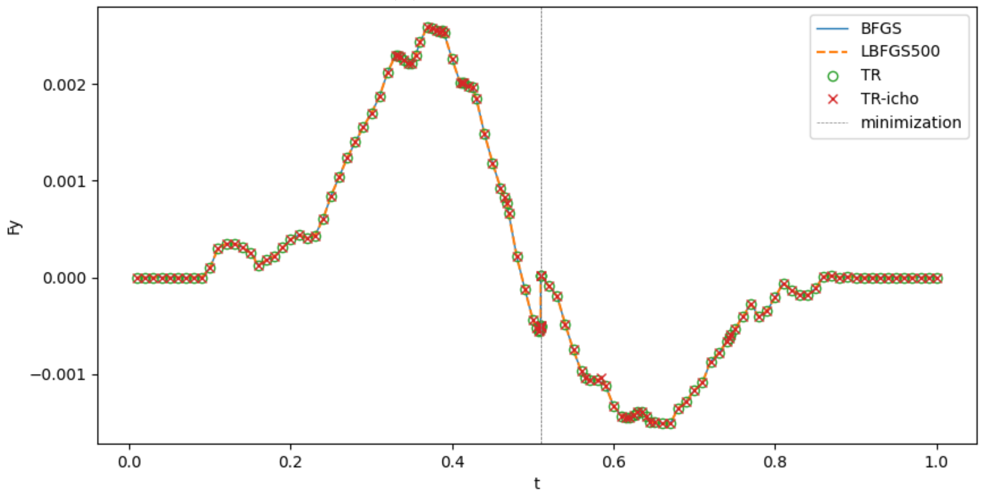


Figure 2.40: 3D hyperelasticity, simulation type II, $10 \times 10 \times 10$ FEs. The force-time curves predicted by the different methods. The times at which a minimization solver is applied in the quasi-Newton simulations are indicated by the black vertical lines. The number behind 'LBFGS' in the legend refers to n_{li}.

(a) Force in x-direction



(b) Force in y-direction



(c) Force in z-direction

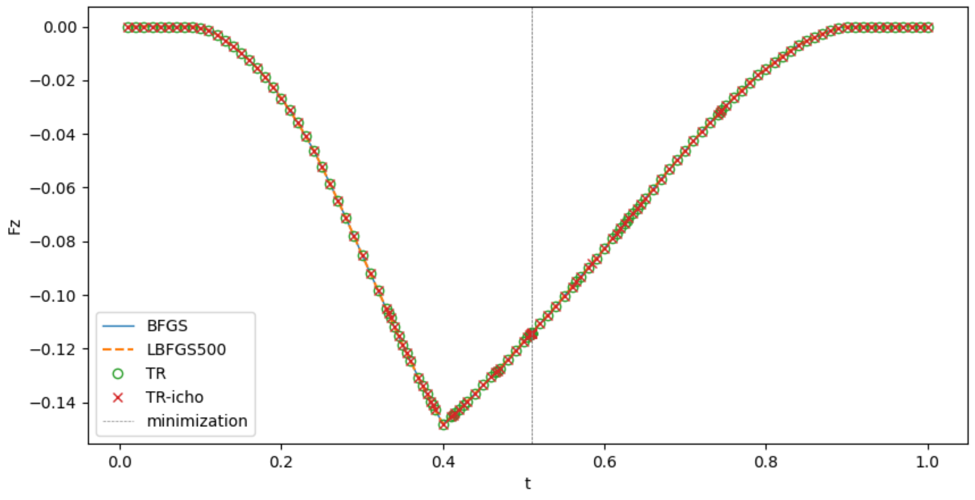


Figure 2.41: 3D hyperelastoplasticity, simulation type II, $10 \times 10 \times 10$ FEs. The force-time curves predicted by the different methods. The times at which a minimization solver is applied in the quasi-Newton simulations are indicated by the black vertical lines. The number behind 'LBFGS' in the legend refers to n_{li}.

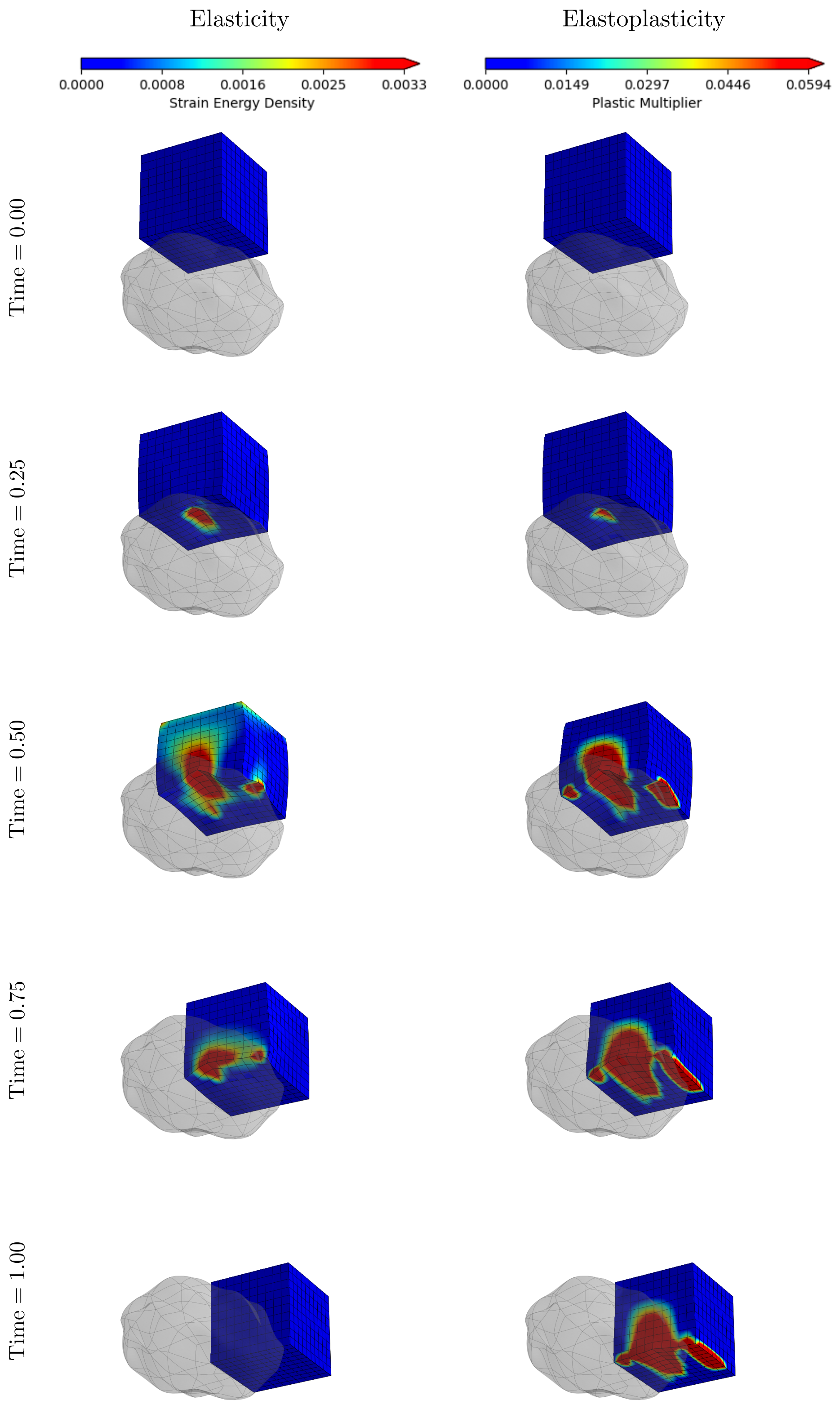


Figure 2.42: 3D simulation type II, 10 × 10 × 10 FEs. Some deformations predicted by the simulations that use the BFGS method.

Table 2.6: 3D simulation type II, $10 \times 10 \times 10$ FEs. Computational performances of the different methods. Iteration counters include iterations of rejected increments.

Method	Newt incr accept	Newt incr reject	Newt iter	Min incr	Min iter	Min force eval	CG iter
Hyperelasticity							
BFGS	126	139	2018	1	550	2764	
LBFGS500	126	139	2018	1	591	2932	
TR	126	139	2018	1	829	830	3234
TR-icho	126	139	2018	1	272	273	369
Hyperelastoplasticity							
BFGS	130	150	2160	1	555	2741	
LBFGS500	130	149	2193	1	626	3663	
TR	130	150	2152	1	1173	1174	5842
TR-icho	130	151	2177	1	171	172	404

The computational performances of the methods are summarized in the first main row of Table 2.6. It can be seen that the preconditioned TR method required less than 10% of the force evaluations than the quasi-Newton methods. The original TR method required approximately 30% of the force evaluations of the quasi-Newton methods.

3D hyperelasticity, Type II, $10 \times 10 \times 10$ **FEs**

In the last set of simulations, the constitutive model is switched from the elastic one to the elastoplastic one, whilst the rest remains the same as in the previous set of simulations. The predicted force-time diagrams in Fig. 2.41 show that all optimization solvers were applied only once and that they all predicted the same results. The right column of Fig. 2.42 shows some deformations predicted during the simulations.

The second main row in Table 2.6 indicates that the L-BFGS method required 34% more force evaluations than the BFGS method. The original TR method required 57% less force evaluations than the BFGS method, and the preconditioned TR method required 94% less force evaluations than the BFGS method - of course, at the expense of preconditioning.

2.6 Conclusion

Newton's method with bilateral contact enforcement and a constant penalty value was the main solver of the contact simulations presented in this chapter (and in the next chapter, for that matter). If Newton's method failed to converge or if the same active set was encountered for the same increment, the current time increment was cut in half and Newton's method attempted again to solve the new time increment. If this strategy did not provide a converged solution after 10 cut-backs were made, Newton's method was exchanged for an optimization solver.

In the contact scenarios in this chapter, three cases can be identified. This chapter has demonstrated that, for certain increments, Newton's method alone would have been sufficient if a larger penalty stiffness had been used. Because this would have yielded drastically longer simulation times for the optimization solvers, this was not performed.

For other time increments, Newton's method indeed did not converge due to snap-back, but the snap-back was a numerical artifact of the node-to-segment contact framework. In such cases, mortar methods are likely to avoid the need for an optimization solver.

In other cases, Newton's method also did not converge due to snap-back, but the snap-back was physically supposed to occur and completely unrelated to the employed contact framework.

The optimization solvers performed well for the first two cases. For the third case, i.e. when snap-back is physically correct and unrelated to the contact framework, the change of deformation of the deformable body was so substantial that not all optimization solvers predicted the same minimizer. Such differences seem to appear more easily in 3D scenarios than in 2D scenarios.

The optimization solvers under investigation were the BFGS method, the L-BFGS method, the trust region (TR) method using Steihaug-Toint's conjugate gradient method and the TR method with the same but preconditioned conjugate gradient method. If the methods converged to a different minimizer, the two quasi-Newton methods would normally yield the same minimizer and the two TR methods would yield the other minimizer.

The TR methods would generally require fewer gradient evaluations than the quasi-Newton methods, with the preconditioned TR method being the clear winner. Most of the times, the preconditioned TR method would require approximately 3 to 7 times less gradient evaluations, but for one simulation setup, this number reached 20. The preconditioned TR method also requires fewer conjugate gradient iterations than the original TR method. On the other hand, additional time is required to construct the preconditioner.

Part of the reason that the TR methods require fewer gradient evaluations than quasi-Newton methods is that they are not hindered by a relatively time-consuming line search. At the same time, the line search ensures that the quasi-Newton methods do not cross energy barriers, which is unphysical. This is the most likely reason that the quasi-Newton methods appear to be more robust for the contact simulations of this chapter.

An interesting part of this chapter was that the optimization solvers were used not only in the context of hyperelasticity, but also in the context of hyperelastoplasticity. This is unique to the best of the author's knowledge and definitely in the context of unilateral contact simulations. The elastoplastic simulations required the optimization solvers less frequently than the elastic simulations. The optimization solvers performed equally well for hyperelasticity as for hyperelasticity.

Although the conclusion chapter of this thesis presents additional conclusions related to this chapter, a final comment is in place with regard to the finite element interpolation employed. The only other two studies that have investigated optimization solvers for quasi-static unilateral contact simulations (and did not consider snap-back and such large deformations as in this chapter) have used linear triangular finite elements. In this chapter, however, bilinearly quadrilateral and trilinearly hexagonal elements were employed. It is quite possible that if linear triangular and tetrahedral elements had been used, the optimization solvers would have predicted the same outcome results.

Chapter 3

Neural networks for contact detection in deformable-to-rigid unilateral contact mechanics

3.1 Introduction

The current chapter focuses on the capabilities of two neural networks (NNs) to replace the contact detection algorithm in the simulations of the previous chapter. The contact detection algorithm was not explicitly introduced in the previous chapter, but is elaborately explained in the current chapter.

It is worth noting that the current chapter employs the same simulations, test setups, constitutive laws, discretizations, hyperparameters for the solvers, and the same node-to-segment contact framework (including the penalty value) as the previous chapter. Only two aspects are different. First, only the 3D rigid leader-body is used as the test problem in the current chapter. Hence, the current chapter does not treat the 2d test problem of the previous chapter. The reason is that the geometry of the rigid leader-body of the 2D test problem is deemed too simple to pose an interesting test case. The second difference is that the type I 3D simulation of the previous chapter is replaced by another simulation. The reason for this will become clear later in this chapter.

The two NNs are a network called Neural-Pull and a multi-task NN that is newly proposed in this chapter. Neural-Pull was selected from the broad offer of NNs developed by the computer vision community to rapidly emulate the signed distance between a surface and a point in space. Neural-Pull was selected from all available NNs because of its loss function. The loss function uses only one contribution to train Neural-Pull for both its primary output (i.e. the signed distance), as well as for its gradient. Normally, the loss function requires two contributions to train an NN for its primary output and the primary output's first-order derivatives with respect to the NN's input.

The second NN is not taken from the literature but is newly proposed in the current chapter. It is a so-called multi-task NN and is tailored to mimic some tasks of the contact detection algorithm simultaneously. In fact, it only emulates the most time-consuming tasks of the conventional algorithm that require the least accuracy. By not replacing the last task of the conventional algorithm, which is responsible for the algorithm's accuracy, potential errors of the multi-task NN are envisioned to not occur in the algorithm's final output that is required for the simulation.

Table 3.1: List of symbols

Symbol	**Description**
$\mathbf{X}$	location vector in undeformed follower-body
$\mathbf{u} = \mathbf{u}(\mathbf{X})$	displacement vector of follower-body
$\mathbf{x} = \mathbf{X} + \mathbf{u}(\mathbf{X})$	location vector in deformed follower-body
C	set with all follower-vertices
C_{act}	set with follower-vertices in contact ($C_{\text{act}} \subseteq C$)
$\underline{\xi}$	parametric surface coordinates
p	patch number
P	set with all patch numbers
P_{act}	set with exact patch numbers associated with fol.-vertices in C_{act}
$\hat{P}_{\text{act}}$	set with candidate patch numbers associated with fol.-vertices in C_{act}
$\mathbf{X}_{\text{lead}} = \mathbf{X}_{\text{lead}}(p, \underline{\xi})$	location vector on rigid leader-surface
$\mathbf{n}_{\text{lead}} = \mathbf{n}_{\text{lead}}(p, \underline{\xi})$	outward pointing normal of the rigid leader-surface
g	signed distance
x	function input/1 of 3 global directions
y	output of perceptron or of NN/1 of 3 global directions
z	input of activation function/1 of 3 global directions
f	function
$\mathbf{f}$, $\underline{\mathbf{f}}$	force vector, column with force vectors
$\mathbf{K}$, $\underline{\underline{\mathbf{K}}}$	2nd-order stiffness tensor, matrix with 2nd-order stiffness tensors
w, $\underline{w}$, $\underline{\underline{W}}$	weight, column of weights of perceptron, matrix of weights of layer
b, $\underline{b}$	bias, column of biases of a layer
α	stepsize, i.e. learning rate for NNs
μ	mean
σ^2	variance
ϵ	positive constant
$\underline{\underline{m}}$	matrix with first moment estimates
$\underline{\underline{s}}$	matrix with second moment estimates

Some comments are required to relate some of the mathematical notation and concepts of the previous chapter to those of the current chapter. First, the reader may have noticed that nearly all the letters of the alphabet have already been used as mathematical symbols in the previous chapter. Consequently, the meaning of some symbols is changed in the current chapter. Table 3.1 lists the most important symbols used in the current chapter, including all those whose meaning has changed. Second, an important difference from the previous chapter is that in the previous chapter, the parametric surface coordinates $\underline{\xi}$ were sufficient to reach any location on the leader surface. Thus, a simplified dependency

$\mathbf{X}_{\text{lead}}(\underline{\xi})$ was considered to ease the notation of the previous chapter. This simplification cannot be used in the current chapter. The current chapter considers that the leader surface is subdivided into patches. Within each patch, the parametric surface coordinates are bounded between zero and one, i.e. $0 \leq \xi_1 \leq 1$, $0 \leq \xi_2 \leq 1$. Thus, the dependency $\mathbf{X}_{\text{lead}}(p, \underline{\xi})$ is considered in the current chapter, where p denotes the number of the patch.

The outline of the remainder of this chapter is as follows. The next section discusses the workings of the conventional contact detection algorithm and how it is used in the full simulation algorithm. Section 3 provides a concise explanation of the concepts of NNs. Section 4 discusses Neural-Pull, whilst the proposed multi-task NN is presented in Section 5. Section 6 presents the generation of the training data, compares the accuracy of Neural-Pull with that of the proposed multi-task NN, and investigates the performance of the contact simulations enhanced by the multi-task NN relative to those that exploit the conventional contact detection algorithm. Finally, a short set of conclusions is presented in section 7.

3.2 Contact detection algorithm

The first subsection of the current section discusses the purpose of the contact detection algorithm (CDA) and the individual parts of the CDA in detail. The second subsection discusses its place within the entire contact simulation algorithm.

3.2.1 Purpose and implementation

The contact simulations of this thesis consider a deformable follower-body and a rigid leader-surface in frictionless unilateral node-to-surface contact. Gregory patches are used to smooth the rigid leader-surface. The rigid leader-surface does not translate or rotate.

In this context, the conventional CDA is responsible for computing which location on the rigid leader-surface corresponds to the shortest distance between a follower-vertex of interest (i.e. a vertex on the follower-surface) and the rigid leader-surface. To do so, the CDA must solve a minimization problem for each follower-vertex. This minimization problem is also known as the closest-point projection and reads for a single follower-vertex:

$$p^*, \underline{\xi}^* = \underset{\substack{p \in P \\ 0 \leq \xi_1 \leq 1 \\ 0 \leq \xi_2 \leq 1}}{\operatorname{argmin}} \quad \left\| \mathbf{x} - \mathbf{X}_{\text{lead}}(p, \underline{\xi}) \right\|, \tag{3.1}$$

where $\|\bullet\|$ denotes the L^2-norm, $\mathbf{x}$ the location vector of the follower-vertex in the deformed configuration, and $\mathbf{X}_{\text{lead}}$ the location vector of the rigid leader-surface and is a function of patch number p and two parametric surface coordinates stored in $\underline{\xi}$. P denotes the index set with all patch numbers. For completeness, the location vector of the follower vertex in the deformed configuration can be expressed in terms of its location vector in the reference configuration, $\mathbf{X}$, and its displacement vector relative to the reference configuration, $\mathbf{u}$, as follows:

$$\mathbf{x} = \mathbf{X} + \mathbf{u}(\mathbf{X}). \tag{3.2}$$

Once the patch and parametric surface coordinates that minimize the closest distance are available, the signed distance and its gradient and Hessian (with respect to the displacement vector of the follower-vertex) can rapidly be evaluated. The signed distance, g, is formulated as:

$$g = \big(\mathbf{x} - \mathbf{X}_{\text{lead}}(p^*, \underline{\xi}^*)\big) \cdot \mathbf{n}_{\text{lead}}(p^*, \underline{\xi}^*), \tag{3.3}$$

where the leader-surface's outward-pointing normal vector is denoted by $\mathbf{n}_{\text{lead}}$. Once the closest-point projection of Eq. (3.1) is solved, the CDA also evaluates the signed distance - at hardly any additional costs. It uses signed distance g to determine whether the follower-vertex of interest is located inside the rigid leader-body ($g < 0$), outside the leader-body ($g > 0$), or exactly on top of its surface ($g = 0$).

Once the CDA has considered all follower-vertices in index set C, the CDA constructs i) C_{act}: a set of follower-vertices that are currently penetrating the leader-body, ii) P_{act}: a set that indicates which patch number contains the closest-point projection of each follower-vertex in C_{act}, and iii) $\underline{\underline{\xi}}^*$: a matrix that provides the closest-point projections for each follower-vertex in C_{act}.

The CDA consists of three individual algorithms. The first algorithm is called the broad phase and filters out the follower-vertices that are located far away from the leader-body. For each follower-vertex that has passed the broad phase, the narrow phase precisely computes its patch number and parametric surface coordinates. The narrow phase accomplishes this in two steps. In the first algorithm of the narrow phase, suitable initial guesses for the parametric surface coordinates are calculated. In the narrow phase's second algorithm, Newton's method is used to compute the parametric surface coordinates exactly.

The 'filtering' performed by the broad phase starts with the generation of a convex hull around each patch of the leader-surface. In this thesis, the convex hull is a bounding sphere (see Fig. 3.2). The center of the bounding sphere, $\mathbf{X}^{\text{BS}}_{\text{lead}}$, is set as the average location vector of those of the twenty control points of the Gregory patch. Subsequently, the distance between the bounding sphere's center and the control point that is located the furthest is calculated. The radius of the bounding sphere, r^{BS}, is then set as this distance multiplied by a factor between 1 and 2. The use of the factor helps to prevent that no follower-vertices near the patch edges remain undetected. Because the leader-surface does not translate nor rotate during the course of a simulation, the centers and radii of all bounding spheres are calculated only at the start of each simulation.

A pseudo-algorithm for the broad phase is presented in Alg. 5. It essentially consists of a double for-loop, where the outer loop loops over all follower-vertices (in C) and the inner loop loops over all patches (in P). The result of the broad phase algorithm are two sets, $\hat{C}_{\text{act}}$ and $\hat{P}_{\text{act}}$. Set $\hat{C}_{\text{act}}$ stores the indices of follower-vertices that are located in the bounding sphere of at least one patch. Each component in $\hat{P}_{\text{act}}$ stores the associated patch numbers. Fig. 3.2 graphically illustrates that the index of a follower-vertex may be present more than once in $\hat{C}_{\text{act}}$. Then it will also have more associated patches stored in $\hat{P}_{\text{act}}$.

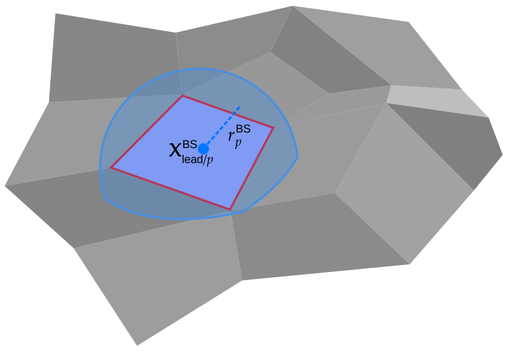
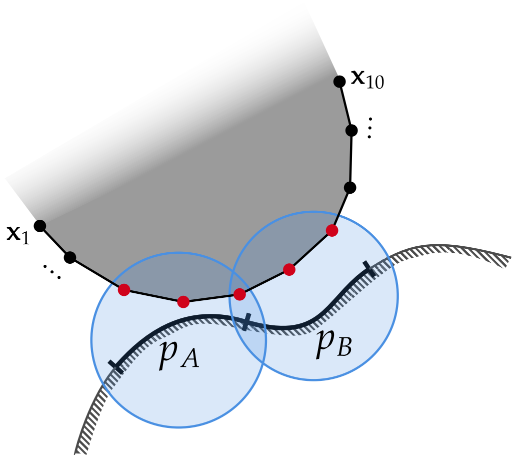


Figure 3.1: Broad phase: a bounding sphere around a patch of the lead surface.



	p_A	p_B
$\mathbf{x}_1$		
$\mathbf{x}_2$		
$\mathbf{x}_3$	candidate	
$\mathbf{x}_4$	candidate	
$\mathbf{x}_5$	candidate	candidate
$\mathbf{x}_6$		candidate
$\mathbf{x}_7$		candidate
$\mathbf{x}_8$		
$\mathbf{x}_9$		
$\mathbf{x}_{10}$		

Figure 3.2: Broad phase: Sketch of ten follower-vertices and whether or not they will be selected by the broad phase for further inspection in the narrow phase. The table on the right illustrates that five follower-vertices will be selected by the broad phase, and one will appear twice in $\hat{C}_{\text{act}}$ because it is associated with two patches.

Algorithm 5 Broad phase: Candidate selection

```
    INPUT: $\underline{\mathbf{u}}$, $C$, $P$, $\underline{\mathbf{X}}^{\text{BS}}_{\text{lead}}$, $\underline{r}^{\text{BS}}$
 1: $\hat{C}_{\text{act}} = \{\}$
 2: $\hat{P}_{\text{act}} = \{\}$
 3: for $i \in C$ do
 4:     $\mathbf{x}_i = \mathbf{X}_i + \mathbf{u}_i$
 5:     for $p \in P$ do
 6:         if $\|\mathbf{x}_i - \mathbf{X}^{\text{BS}}_{\text{lead}/p}\| \leq r^{\text{BS}}_p$ then
 7:             add $i$ to $\hat{C}_{\text{act}}$
 8:             add $p$ to $\hat{P}_{\text{act}}$
 9:         end if
10:     end for
11: end for
    OUTPUT: $\hat{P}_{\text{act}}$, $\hat{C}_{\text{act}}$
```

Algorithm 6 Narrow phase 1: Calculate initial guesses $\underline{\underline{\hat{\xi}}}$ (detailed alg. in Appendix B)

```
INPUT: $\underline{\mathbf{u}}$, $\hat{C}_{\text{act}}$, $\hat{P}_{\text{act}}$
1: for $i \in \hat{C}_{\text{act}}$ do
2:     $\mathbf{x}_i = \mathbf{X}_i + \mathbf{u}_i$
3:     for $p \in \hat{P}_{\text{act}}$ do
4:         Given $\mathbf{x}_i$ and $p$, determine $\underline{\hat{\xi}}_i$ via recursive seeding
5:     end for
6: end for
OUTPUT: $\underline{\underline{\hat{\xi}}}$
```

Once the broad phase is completed, the set $\hat{C}_{\text{act}}$ stores the indices of follower-vertices that are close to the leader-surface. The associated set $\hat{P}_{\text{act}}$ stores the potential patches that may contain the closest-point projections. Both sets are handed to the first stage of the narrow phase. The narrow phase will first calculate for each combination of follower-vertex/associated patch in $\hat{C}_{\text{act}}$ and $\hat{P}_{\text{act}}$, a suitable initial guess of parametric surface coordinates $\underline{\hat{\xi}}$. It accomplishes this by seeding a 5×5 point grid of parametric surface coordinates on the patch associated with the follower-vertex (left image in Fig. 3.3). The algorithm calculates the distance between each grid point and the follower-vertex of interest. Subsequently, the algorithm determines which grid point is closest to the follower-vertex of interest (red point in the left image of Fig. 3.3) and seeds another 5×5 grid around the grid point closest to the follower-vertex of interest (right image in Fig. 3.3). The area difference between two subsequent grids is $\frac{49}{1024}$ (if the grid remains fully located within the patch, i.e. $0 \leq \xi_1 \leq 1$, $0 \leq \xi_2 \leq 1$). The recursive seeding of ever-decreasing grids is repeated a maximum of eight times, or until the parametric surface coordinates associated with the closest distance between two consecutive grids change by no more than 0.005. Algorithm 6 presents a highly simplified pseudocode.

After applying the broad phase and the first narrow phase algorithm, the results are i) set $\hat{C}_{\text{act}}$ of follower-vertices that are close to the leader-surface, ii) set $\hat{P}_{\text{act}}$ of potential patches that contain the closest-point projections, and iii) matrix $\underline{\underline{\hat{\xi}}}$ that stores the initial guess for each combination of follower-vertex/associated patch in $\hat{C}_{\text{act}}$ and $\hat{P}_{\text{act}}$. The next step of the narrow phase is to deploy Newton's method to calculate the closest-point projections accurately, i.e. $\underline{\underline{\xi}}^*$. Alg. 7 presents this in a simplified fashion. After the accurate closest-point projection of each follower-vertex is computed, the algorithm also evaluates the signed distance to determine which follower-vertex penetrates the leader-body (and which patch contains its closest-point projection).

The total CDA, i.e. Algs. 5, 6 and 7 together, do not compute the gradient and Hessian of the signed distance, but they can be rapidly evaluated using the final outputs of the CDA. As the previous chapter has shown, the signed distance's gradient and Hessian with respect to the follower-vertex's displacement vector, $\mathbf{u}$, read:

$$\left.\frac{\mathrm{d}\,g}{\mathrm{d}\,\mathbf{u}}\right|_{p^*,\underline{\xi}^*} = \mathbf{n}_{\text{lead}}(p^*, \underline{\xi}^*) = \mathbf{n}^*_{\text{lead}}, \tag{3.4}$$

$$\left.\frac{\mathrm{d}^2\,g}{\mathrm{d}\,\mathbf{u}^2}\right|_{p^*,\underline{\xi}^*} = \left.\frac{\mathrm{d}\,\mathbf{n}_{\text{lead}}}{\mathrm{d}\,\mathbf{u}}\right|_{p^*,\underline{\xi}^*} = \frac{\mathrm{d}\,\mathbf{n}^*_{\text{lead}}}{\mathrm{d}\,\mathbf{u}}. \tag{3.5}$$

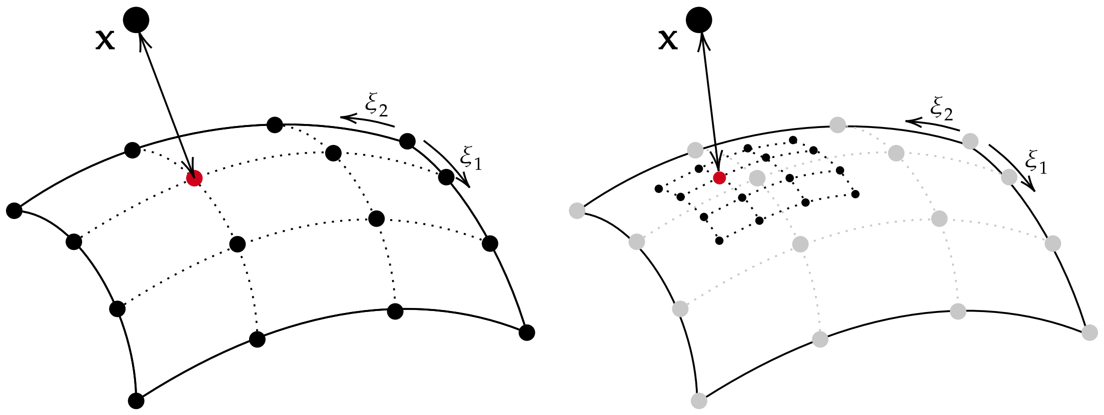


Figure 3.3: Recursive seeding to calculate initial guess $\underline{\hat{\xi}}$ for Newton's method. Left: Initial grid. Right: Second grid centered around the closest point found in the initial grid.

Algorithm 7 Narrow Phase 2: Compute $\underline{\underline{\xi}}^*$, C_act and P_act

INPUT: $\underline{\mathbf{u}}$, $\hat{C}_\text{act}$, $\hat{P}_\text{act}$, $\underline{\underline{\hat{\xi}}}$

```
 1: $C_\text{act} = \{\}$
 2: $P_\text{act} = \{\}$
 3: for $i \in \hat{C}_\text{act}$ do
 4:     $\mathbf{x}_i = \mathbf{X}_i + \mathbf{u}_i$
 5:     for $p \in \hat{P}_\text{act}$ do
 6:         Given $\mathbf{x}_i$, $p$ and $\underline{\hat{\xi}}_i$, use Newton's method to calculate $\underline{\xi}_i^*$
 7:         if $0 \le \underline{\xi}_i^* \le 1$ then
 8:             Compute $g(\mathbf{x}_i, p, \underline{\xi}_i^*)$
 9:             if $g \le 0$ then
10:                 add $i$ to $C_\text{act}$
11:                 add $p$ to $P_\text{act}$
12:             end if
13:         end if
14:     end for
15: end for
```

OUTPUT: $\underline{\underline{\xi}}^*$, C_act, P_act

3.2.2 The contact detection algorithm in the simulation algorithm

Algs. 8, 9 and 10 present the locations of the contact detection algorithm within a trust region (TR) method, a quasi-Newton method, and Newton's method. To ease the notation, only the most relevant dependencies are incorporated. For instance, the history variables of the hyperelastoplasticity are completely omitted.

It is perhaps the most straightforward to start with the TR method of Alg. 8. In this algorithm, the TR method computes a potential update ($\underline{\mathbf{h}}$) of the current estimate ($\underline{\mathbf{u}}$). For the newly proposed estimate ($\underline{\tilde{\mathbf{u}}} = \underline{\mathbf{uh}}$), the gradient must be evaluated. However, before the gradient can be evaluated, the CDA must evaluate the contact situation for the newly proposed estimate $\underline{\tilde{\mathbf{u}}}$.

The pseudo-algorithm for the quasi-Newton method of Alg. 9 resembles that of the TR method, but includes two differences. First, the quasi-Newton method computes the proposed search direction $\underline{\mathbf{h}}$ in a different manner (line 2). Second, the quasi-Newton method deploys a line search that computes a suitable stepsize α in the direction of $\underline{\mathbf{h}}$. Nevertheless, for each newly proposed estimate $\underline{\tilde{\mathbf{u}}}$, the gradient evaluation in line 7 depends on the results of the CDA (line 6).

Algorithm 8 Location of the CDA inside the TR method.

```
INPUT: $C$, $P$, unconverged $\underline{\mathbf{u}}$, $C_{\text{act}}$, $P_{\text{act}}$, $\underline{\underline{\xi}}^*$
1: while no convergence do
2:     Evaluate $\underline{\underline{\mathbf{K}}} = \underline{\underline{\mathbf{K}}}(\underline{\mathbf{u}}, C_{\text{act}}, P_{\text{act}}, \underline{\underline{\xi}}^*)$
3:     $\underline{\mathbf{h}} = \mathcal{F}\left(\underline{\underline{\mathbf{K}}}, \underline{\mathbf{f}}\right)$
4:     $\underline{\tilde{\mathbf{u}}} = \underline{\mathbf{u}} + \underline{\mathbf{h}}$
5:     $C_{\text{act}}, P_{\text{act}}, \underline{\underline{\xi}}^* = \textbf{CDA}(\underline{\tilde{\mathbf{u}}}, C, P)$
6:     Evaluate $\underline{\tilde{\mathbf{f}}} = \underline{\mathbf{f}}(\underline{\tilde{\mathbf{u}}}, C_{\text{act}}, P_{\text{act}}, \underline{\underline{\xi}}^*)$
7: end while
OUTPUT: $\underline{\mathbf{u}}$, $C_{\text{act}}$, $P_{\text{act}}$, $\underline{\underline{\xi}}^*$
```

Algorithm 9 Location of the CDA inside the quasi-Newton method.

```
INPUT: $C$, $P$, unconverged $\underline{\mathbf{u}}$, $C_{\text{act}}$, $P_{\text{act}}$, $\underline{\underline{\xi}}^*$
1: while no convergence do
2:     $\underline{\mathbf{h}} = \mathcal{F}\left(\underline{\underline{\mathbf{L}}}, \underline{\mathbf{u}}, \underline{\mathbf{f}}\right)$
3:     while no convergence do
4:         Suggest $\alpha$
5:         $\underline{\tilde{\mathbf{u}}} = \underline{\mathbf{u}} + \alpha\underline{\mathbf{h}}$
6:         $C_{\text{act}}, P_{\text{act}}, \underline{\underline{\xi}}^* = \textbf{CDA}(\underline{\tilde{\mathbf{u}}}, C, P)$
7:         Evaluate $\underline{\tilde{\mathbf{f}}} = \underline{\mathbf{f}}(\underline{\tilde{\mathbf{u}}}, C_{\text{act}}, P_{\text{act}}, \underline{\underline{\xi}}^*)$
8:     end while
9: end while
OUTPUT: $\underline{\mathbf{u}}$, $C_{\text{act}}$, $P_{\text{act}}$, $\underline{\underline{\xi}}^*$
```

Whereas the TR method and the quasi-Newton method employ the entire CDA to evaluate contact situations for each newly proposed estimate, this is not performed in Newton's method for the contact simulations of this chapter. As can be observed in Alg. 10, the entire CDA is only applied once Newton's method finishes. Besides the standard outputs C_{act}, P_{act} and $\underline{\underline{\xi}}^*$, also the results of the broad phase (i.e. $\hat{C}_{\text{act}}$ and $\hat{P}_{\text{act}}$, see Alg. 5), are stored for later use.

Within Newton's method itself, only a part of the CDA is used to evaluate the contact situation for each update of the estimate. This is presented as **CDA'** in Alg. 10 and only constitutes the two algorithms of the narrow phase (Algs. 6 and 7). The broad phase is here thus bypassed, which is why the broad phase's results in line 8 of Alg. 10 must be stored as well. Another difference between the standard CDA and the simplified CDA (**CDA'** in Alg. 10) is that the simplified CDA is not allowed to decide which follower-vertices are penetrating the rigid leader-body. Thus, the only task of the simplified CDA in Newton's method is to compute which patch contains the closest-point projection and what is the closest-point projection (i.e. $\underline{\xi}^*$). The reason for this particular use of the CDA in Newton's method relative to its use in the quasi-Newton and TR methods is that the contact constraints are enforced in a bilateral manner if Newton's method is the solver but in a unilateral manner if the optimization solvers are used. The previous chapter has discussed this in considerable detail.

Algorithm 10 Location of the CDA inside Newton's method.

```
INPUT: $C$, $P$, unconverged $\underline{\mathbf{u}}$, $C_{\text{act}}$, $P_{\text{act}}$, $\underline{\underline{\xi}}^*$, $\hat{C}_{\text{act}}$, $\hat{P}_{\text{act}}$
1: while no convergence do
2:     Evaluate $\underline{\underline{\mathbf{K}}} = \underline{\underline{\mathbf{K}}}(\underline{\mathbf{u}}, C_{\text{act}}, P_{\text{act}}, \underline{\underline{\xi}}^*)$
3:     Solve for $\underline{\mathbf{h}}$:   $\underline{\underline{\mathbf{K}}} \cdot \underline{\mathbf{h}} = -\underline{\mathbf{f}}$
4:     $\underline{\mathbf{u}} = \underline{\mathbf{u}} + \underline{\mathbf{h}}$
5:     $C_{\text{act}}, P_{\text{act}}, \underline{\underline{\xi}}^* = \textbf{CDA'}(\underline{\mathbf{u}}, \hat{C}_{\text{act}}, \hat{P}_{\text{act}})$
6:     Evaluate $\underline{\mathbf{f}} = \underline{\mathbf{f}}(\underline{\mathbf{u}}, C_{\text{act}}, P_{\text{act}}, \underline{\underline{\xi}}^*)$
7: end while
8: $C_{\text{act}}, P_{\text{act}}, \underline{\underline{\xi}}^*, \hat{C}_{\text{act}}, \hat{P}_{\text{act}} = \textbf{CDA}(\underline{\mathbf{u}}, C, P)$
OUTPUT: $\underline{\mathbf{u}}$, $C_{\text{act}}$, $P_{\text{act}}$, $\hat{P}_{\text{act}}$, $\underline{\underline{\xi}}^*$
```

3.3 Review of neural networks

This section reviews some basic details of the structures and principles of artificial neural networks (ANNs), which are to replace the CDA. Thus, the current section serves as a theoretical basis for the following sections which detail the ANNs that mimic the CDA. If the review below is insufficient, studies such as [104, 105] may provide additional details.

3.3.1 From perceptron to multi-layer neural network

ANNs are built upon the concept of interconnected units or neurons that process data hierarchically. This subsection introduces the perceptron as the fundamental building block, followed by activation functions, which allow for non-linear transformations. Finally, the current subsection describes multi-layer neural networks, which enable the modelling of complex input-output relationships.

The perceptron (see Fig. 3.4) is a computational model inspired by biological neurons. It functions as a linear binary classifier and transforms a set of input signals into a single output signal based on a weighted sum of the inputs. Mathematically, the perceptron computes a weighted sum z according to:

$$z = \sum_{i=1}^{n} w_i x_i + b, \tag{3.6}$$

where x_i denotes the i^{th} input signal, w_i its associated weight, and b the perceptron's bias term. Subsequently, the perceptron applies an activation function $f(z)$ to z, producing its output, y:

$$y = f(z). \tag{3.7}$$

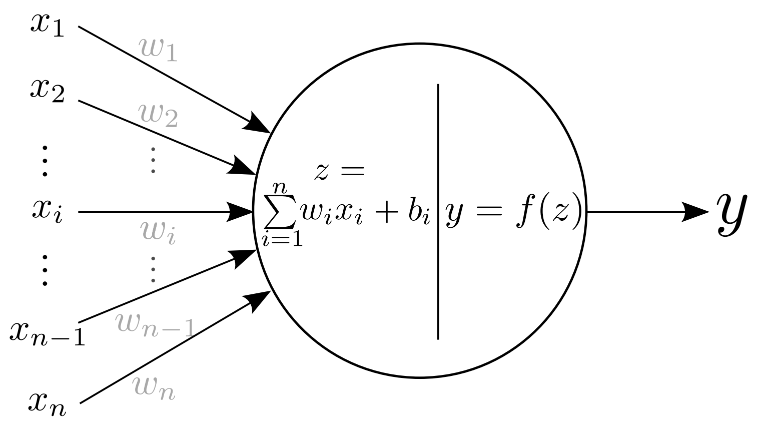


Figure 3.4: Representation of a perceptron that takes inputs $x_1, \ldots, x_n$ and outputs the activated value y.

For a binary classification problem, a common activation function is the (Heaviside) step function:

$$f(z) = \begin{cases} 1 & \text{if } z \geq 0, \\ 0 & \text{if } z < 0. \end{cases} \tag{3.8}$$

This makes the perceptron's output y binary and describes the two possible classes. Geometrically, the perceptron divides the input space into two regions with a hyperplane defined by the weights and bias. Points on one side of the hyperplane belong to one class, while points on the other side belong to the other class.

While the step function is foundational, its inability to handle non-linear relationships limits its applications. This motivates the use of other activation functions, enabling the learning of more complex patterns. Without non-linear activation functions, a network composed of multiple perceptrons would still behave like a single linear model, regardless of its depth. Some common activation functions that partially fulfil these criteria are presented in Fig. 3.5.

Arguably, the most widely used activation function in ANNs is the rectified linear unit (ReLU). This function reads:

$$f_{\text{ReLU}}(z) = \max(0, z). \tag{3.9}$$

It resembles the step function in that it only activates when the weighted sum z reaches a threshold. Additionally, it provides an output signal that is sensitive to the weighted sum z once the activation threshold is attained.

The choice of activation function depends on the task at hand and the network's architecture. ReLU is particularly prevalent in deep learning thanks to its computational efficiency and ability to mitigate vanishing gradients. In some scenarios, however, a more gradual activation of the percetron's output is more suitable. In the case of binary classification, the sigmoid function may yield a useful choice (but may suffer from vanishing gradient issues in deeper networks). The Sigmoid activation reads:

$$f_{\text{Sigmoid}}(z) = \frac{1}{1 + \exp(-z)}. \tag{3.10}$$

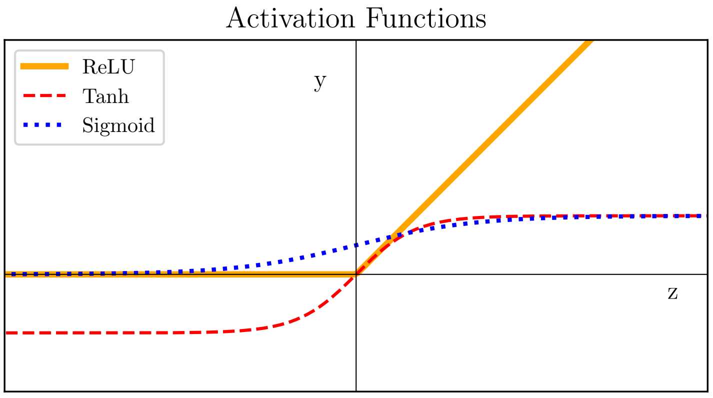


Figure 3.5: Graphical representation of three frequently used activation functions.

If zero-centered outputs are useful, the tanh function can be used instead. The tanh activation reads:

$$f_{\tanh}(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}. \tag{3.11}$$

To overcome the limitations of single-layer perceptrons to emulate more complex phenomena, multi-layer neural networks are generally used. These networks consist of multiple hierarchically organized layers of neurons, including an input layer, one or more hidden layers, and an output layer (see Fig. 3.6). Each neuron computes a weighted sum of its inputs, applies an activation function, and passes the result to all neurons of the next layer. By stacking multiple hidden layers, the network can model complex, non-linear relationships between inputs and outputs.

Mathematically, for a network with n_{L} layers, the computation at layer l can be expressed as:

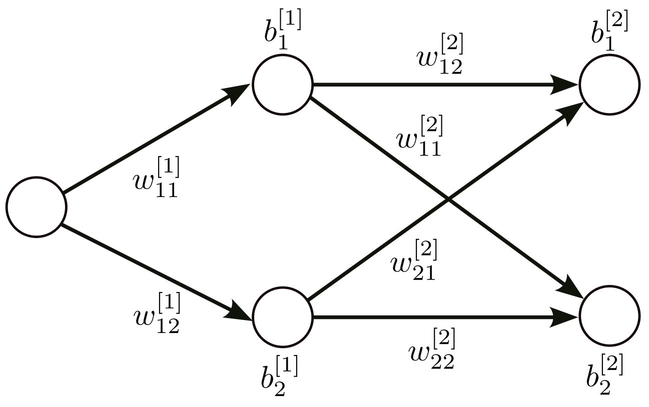
$$\underline{z}^{[l]} = \underline{\underline{W}}^{[l]} \underline{a}^{[l-1]} + \underline{b}^{[l]}, \tag{3.12}$$

where $\underline{z}^{[l]}$ is the linear combination of inputs for layer l, $\underline{\underline{W}}^{[l]}$ and $\underline{b}^{[l]}$ are the weights and biases of layer l and $\underline{a}^{[l-1]}$ is the activation output from the previous layer ($\underline{a}^{[0]}$ denotes to the network's input features in this notation). The activation function is then applied element-wise to compute the activations for layer l:

$$\underline{a}^{[l]} = \underline{f}(\underline{z}^{[l]}). \tag{3.13}$$

This layered structure makes multi-layered networks more suitable to approximate highly complex functions for a wide range of tasks, including classification, regression, and feature extraction.

The next subsection concentrates on important concepts that are necessary to identify the values of the weights and biases. The identification of the weights and biases is an optimization problem that is called 'training' in machine learning terminology.



Input Layer $\in \mathbb{R}$ Hidden Layer $\in \mathbb{R}^2$ Output Layer $\in \mathbb{R}^2$

Figure 3.6: A feedforward neural network consisting of an input layer, one hidden layer and an output layer. Signals travel from left to right. Each connection is associated with a weight. Weight $w_{ij}^{[l]}$ is associated with the connection between the i^{th} neuron of layer $[l-1]$ and the j^{th} neuron of layer $[l]$. Each neuron in layers other than the input layer is associated with a bias ($b_j^{[l]}$).

3.3.2 Concepts for training

The training of a neural network involves adjusting its parameters—weights and biases—so that its predictions closely match the target values. This process is guided by three key steps: the forward pass, loss computation, and back-propagation.

In a forward pass, input data is propagated through the network to produce predictions. For a single sample (i), the network computes the activations at each layer as follows:

$$\underline{z}^{[l](i)} = \underline{\underline{W}}^{[l]} \underline{a}^{[l-1](i)} + \underline{b}^{[l]}, \tag{3.14}$$

where $z^{[l](i)}$ denotes the linear combination of inputs at layer l for sample (i), and $a^{[l-1](i)}$ denotes the activation output from the previous layer (where $a^{[0](i)}$ denotes the network's input features). The activation function f is then applied element-wise to compute:

$$\underline{a}^{[l](i)} = \underline{f}(\underline{z}^{[l](i)}). \tag{3.15}$$

This sequential process continues until the output layer produces the final prediction $\hat{y}^{(i)}$. Once the prediction $\hat{y}^{(i)}$ is generated, the performance of the network is evaluated by comparing it with the target values $y^{(i)}$. A loss function measures the error for a single sample (i). A widely used loss function for regression ANNs is the mean squared error (MSE), which reads:

$$\mathcal{L}_{\text{MSE}}^{(i)} = \|\underline{\hat{y}}^{(i)} - \underline{y}^{(i)}\|^2, \tag{3.16}$$

On the other hand, the cross-entropy (CE) loss function is often used for classification ANNs and reads:

$$\mathcal{L}_{\text{CE}}^{(i)} = -\sum_{j=1}^{n_C} \underline{y}_j^{(i)} \log\left(\underline{\hat{y}}_j^{(i)}\right), \tag{3.17}$$

where n_C denotes the number of classes, and $\underline{\hat{y}}_j^{(i)}$ (the j^{th} number of column $\underline{\hat{y}}_j^{(i)}$) denotes the predicted probability for class j.

The loss measures introduced quantify the error for each sample. In practice, the NN must learn its weights and biases for a wide variety of samples. Cost function $\mathcal{J}$ aggregates the loss over n_m training samples as follows:

$$\mathcal{J} = \frac{1}{n_\mathrm{m}} \sum_{i=1}^{n_\mathrm{m}} \mathcal{L}^{(i)}, \tag{3.18}$$

where $\mathcal{L}$ denotes an unspecified loss function.

Minimizing this cost function with respect to the NN's weights and biases is the objective of the training process. To minimize the cost function, the NN must adjust its weights and biases. To know how to adjust the weights and biases, the gradient of the cost function with respect to the weights and biases must be computed.

Computing the NN's gradient heavily relies on the chain rule for NNs that consist of several layers. For layer l, the gradient of the cost function with respect to $z^{[l](i)}$ is computed as:

$$\underline{\delta}^{[l](i)} = \frac{\partial \mathcal{L}^{(i)}}{\partial \underline{z}^{[l](i)}}. \tag{3.19}$$

The error $\underline{\delta}^{[l](i)}$ is then back-propagated through the network. Using the chain rule, the gradients of the cost function with respect to the weights and biases are computed as:

$$\frac{\partial \mathcal{J}}{\partial \underline{\underline{W}}^{[l]}} = \frac{1}{n_m} \sum_{i=1}^{n_m} \underline{\delta}^{[l](i)} \cdot \left(\underline{a}^{[l-1](i)}\right)^T, \tag{3.20}$$

$$\frac{\partial \mathcal{J}}{\partial \underline{b}^{[l]}} = \frac{1}{n_m} \sum_{i=1}^{n_m} \underline{\delta}^{[l](i)}. \tag{3.21}$$

The gradient indicates how much the cost function changes with respect to each parameter. Using an optimization algorithm, such as stochastic gradient descent (SGD) or Adam, the weights and biases are updated to minimize the cost function:

$$\underline{\underline{W}}^{[l]} \leftarrow \underline{\underline{W}}^{[l]} - \alpha \frac{\partial \mathcal{J}}{\partial \underline{\underline{W}}^{[l]}}, \tag{3.22}$$

$$\underline{b}^{[l]} \leftarrow \underline{b}^{[l]} - \alpha \frac{\partial \mathcal{J}}{\partial \underline{b}^{[l]}}, \tag{3.23}$$

where α denotes the learning rate.

This iterative process of forward pass, loss computation, and back-propagation forms the core of how neural networks learn. The next subsection will review a number of optimization algorithms that are specifically formulated to efficiently and accurately minimize NN's cost function, i.e. to efficiently and accurately train an NN. Challenges specific to the training of NNs, such as over-fitting and vanishing gradients, are also addressed.

3.3.3 Training strategies

Gradient descent is considered the backbone minimization method for NN training due to its simplicity. Gradient updates the current estimate of the optimal parameter values in every iteration, taking a step in the direction in which the cost function locally decreases the most. Often, however, minimization based on gradient descent is time-consuming. The cost function is also characterized by numerous local minima. For these reasons, a variety of adaptations have been proposed [106, 107, 108]. Some important ones are summarized in this subsection.

A well-known method is stochastic gradient descent (SGD). Like conventional gradient descent, it updates the parameters of the network (weights and biases) using the gradient of the cost function with respect to their variations. The difference is that not all samples are considered at once, but only a part of them. Such a subset is called a batch, and for every iteration, another batch of samples is considered. For completeness of the notation below, SGD writes the update of the parameters (weights and biases) as follows:

$$\underline{\underline{W}}^{[l]} \leftarrow \underline{\underline{W}}^{[l]} - \alpha \frac{\partial \mathcal{J}_{\text{batch}}}{\partial \underline{\underline{W}}^{[l]}}, \quad \underline{b}^{[l]} \leftarrow \underline{b}^{[l]} - \alpha \frac{\partial \mathcal{J}_{\text{batch}}}{\partial \underline{b}^{[l]}}, \tag{3.24}$$

where $\mathcal{J}_{\text{batch}}$ denotes the cost function computed for a batch of samples.

SGD, like gradient descent methods, may converge slowly if the objective function presents large differences in the different components of the gradient. In such scenarios, the dominant descent step oscillates, whereas the more convenient (although less pronounced) direction of descent is largely neglected. In those cases, considering the momentum of the iterative process can partially alleviate the issue of slow convergence.

One method developed to include momentum is Adagrad (Adaptive Gradient) [106], which scales learning rates individually for each parameter. This can enable rapid progress of infrequently updated parameters. Adagrad updates the weights as follows:

$$\underline{\underline{q}}_t = \underline{\underline{q}}_{t-1} + \left(\frac{\partial \mathcal{J}_{\text{batch}}}{\partial \underline{\underline{W}}^{[l]}} \right)^2, \quad \underline{\underline{W}}^{[l]} \leftarrow \underline{\underline{W}}^{[l]} - \frac{\alpha}{\sqrt{\underline{\underline{q}}_t + \epsilon}} \frac{\partial \mathcal{J}_{\text{batch}}}{\partial \underline{\underline{W}}^{[l]}} \tag{3.25}$$

where $\underline{\underline{q}}_t$ denotes the cumulative sum of squared gradients, and $(\bullet)^2$ and $\frac{\bullet}{\sqrt{\bullet}}$ here and below are applied in an element-wise fashion. Subscrips $t-1$ and t refer to the previous and the current iteration of the training (optimization process). The biases are furthermore updated in the same manner as the weights, including their own cumulative sum of squared gradients.

The cumulative sum of squared gradients in Adagrad, $\underline{\underline{q}}_t$, depends on those of all previous iterations. Consequently, the progress per iteration can substantially reduce during the course of the training process. To alleviate this deficiency, the formulation of the momentum can be fine-tuned to emphasize the influence of the most recent iterates, at the expense of that of earlier iterates. The following way of updating yields an exponentially decay of the contribution of previous iterates:

$$\underline{\underline{m}}_t = \beta \underline{\underline{m}}_{t-1} + (1-\beta) \frac{\partial \mathcal{J}_{\text{batch}}}{\partial \underline{\underline{W}}^{[l]}}, \tag{3.26}$$

$$\underline{\underline{W}}^{[l]} \leftarrow \underline{\underline{W}}^{[l]} - \alpha \underline{\underline{m}}_t, \tag{3.27}$$

where $\underline{\underline{m}}_t$ denotes the first moment estimate and β the momentum coefficient. Again, the biases are updated in the same fashion, but not presented here for ease of presentation.

An alternative to Adagrad that uses the exponential decay of previous iterations is called Adadelta [107]. Whereas the learning rate in Adagrad often reduces too rapidly, the exponential decay in Adadelta often improves the performance. In addition to incorporating the exponential decay with respect to Adagrad, Adadelta also normalizes the gradient with respect to its magnitude. This may improve the performance as the training data in each batch is different. The update of the weights in Adadelta is formulated as follows:

$$\underline{\underline{s}}_t = \beta \underline{\underline{s}}_{t-1} + (1-\beta)\left(\frac{\partial \mathcal{J}_{\text{batch}}}{\partial \underline{\underline{W}}^{[l]}}\right)^2, \tag{3.28}$$

$$\underline{\underline{W}}^{[l]} \leftarrow \underline{\underline{W}}^{[l]} - \frac{\Delta \underline{\underline{W}}_t}{\sqrt{\underline{\underline{s}}_t + \epsilon}}, \tag{3.29}$$

$$\Delta \underline{\underline{W}}_t = \frac{\sqrt{\Delta \underline{\underline{s}}_t + \epsilon}}{\sqrt{\underline{\underline{s}}_t + \epsilon}} \cdot \frac{\partial \mathcal{J}_{\text{batch}}}{\partial \underline{\underline{W}}^{[l]}}, \tag{3.30}$$

where $\underline{\underline{s}}_t$ denotes the decaying average of squared gradients and $\Delta\underline{\underline{s}}_t$ the decaying average of squared parameter updates. β denotes the decay rate and the small constant ϵ improves the numerical stability.

Another popular training algorithm is the Adam (Adaptive Moment Estimation) optimization algorithm [108]. Adam updates the weights as follows:

$$\underline{\underline{m}}_t = \beta_1 \underline{\underline{m}}_{t-1} + (1-\beta_1)\frac{\partial \mathcal{J}_{\text{batch}}}{\partial \underline{\underline{W}}^{[l]}}, \tag{3.31}$$

$$\underline{\underline{s}}_t = \beta_2 \underline{\underline{s}}_{t-1} + (1-\beta_2)\left(\frac{\partial \mathcal{J}_{\text{batch}}}{\partial \underline{\underline{W}}^{[l]}}\right)^2, \tag{3.32}$$

$$\tilde{\underline{\underline{m}}}_t = \frac{\underline{\underline{m}}_t}{1-\beta_1^t}, \quad \tilde{\underline{\underline{s}}}_t = \frac{\underline{\underline{s}}_t}{1-\beta_2^t}, \tag{3.33}$$

$$\underline{\underline{W}}^{[l]} \leftarrow \underline{\underline{W}}^{[l]} - \alpha \frac{\tilde{\underline{\underline{m}}}_t}{\sqrt{\tilde{\underline{\underline{v}}}_t + \epsilon}}, \tag{3.34}$$

where $\underline{\underline{m}}_t$ and $\underline{\underline{s}}_t$ denote the first and second moment estimates (running averages of gradients and squared gradients, respectively), while $\tilde{\underline{\underline{m}}}_t$ and $\tilde{\underline{\underline{s}}}_t$ are their bias-corrections. Bias corrections prevent vanishing terms of the very first iterations (i.e. for small t). β_1 and β_2 denote the exponential decay rates for $\underline{\underline{m}}_t$ and $\underline{\underline{s}}_t$.

Unlike Adam, which combines momentum with adaptive learning rates, Adadelta focuses on maintaining unitless and consistent updates by normalizing gradient magnitudes and parameter changes over time. While Adam is often favoured for its robust performance across tasks, Adadelta is particularly well-suited for problems where learning rate tuning is challenging or where parameters need to adjust dynamically over a wide range of scales.

Besides the different optimization techniques employed in the training of ANNs, other improvements have also been developed in the last decades. A suggestion that provides regularization is called 'batch normalization'. Batch normalization reduces internal covariate shifts, improves gradient flow, and makes training less sensitive to initialization and learning rate choices. It is especially effective in deep networks. Batch normalization for a layer, l, starts with the following expression:

$$\underline{\tilde{z}}^{[l]} = \frac{\underline{z}^{[l]} - \mu}{\sqrt{\sigma^2 + \epsilon}},$$

where μ and σ^2 denote the mean and variance of the batch, and ϵ again denotes a small constant that prevents division by zero. The normalized values are then scaled and shifted with the help of learnable parameters $\underline{\underline{\gamma}}$ and $\underline{\kappa}$ as follows

$$\underline{\bar{z}}^{[l]} = \underline{\underline{\gamma}}\,\underline{\tilde{z}}^{[l]} + \underline{\kappa}.$$

Moreover, deep NNs often suffer from vanishing gradients. This means that training an NN becomes harder as the number of layers of the NN increases. One structural improvement in these cases is so-called 'skip connections'. Skip connections in residual networks (ResNets) pass inputs or intermediate activations to deeper layers:

$$\underline{a}^{[l+1]} = \underline{f}(\underline{z}^{[l]} + \underline{a}^{[l-1]}), \tag{3.35}$$

where $\underline{a}^{[l-1]}$ denotes the activation layer $l-1$, and $\underline{z}^{[l]}$ denotes the linear combination of inputs for layer l. This concept can be generalized and used to share information between any layer.

Using optimization algorithms and structural enhancements, such as skip connections, ANNs can be trained faster and more accurately. Several of these techniques are applied to the proposed multi-task ANN of section 3.5 that is to replace the CDA.

3.4 Neural-Pull

The current section introduces and discusses Neural-Pull as a potential replacement for the conventional CDA. Neural-Pull is, in fact, not an ANN, but a loss function tailored to train an NN with the signed distance as its primary output. The loss function of Neural-Pull trains the NN not only for its primary output, i.e. the signed distance, but also for the gradient of the signed distance with respect to its input, i.e. the location vector of the follower-vertex of interest. The particularity of the Neural-Pull loss function is that measures the loss in terms of the signed distance and the gradient in one single contribution. In contrast, the conventional way to quantify the loss with respect to both the primary output and the gradient of the primary output with respect to the NN's input would be to formulate a separate loss function for the primary output and a separate loss function for the gradient.

Thus, in the context of the contact simulations of this thesis, Neural-Pull considers an NN that uses the current location vector of a follower-vertex of interest, $\mathbf{x}$, as its input. The NN's primary output is signed distance $\hat{g}$, where the hat on top of the signed distance indicates that the NN's prediction for the signed distance is an approximation and not necessarily perfect. Differentiating the NN's output with respect to its input yields the following expressions of the gradient and Hessian that are used in the current section:

$$\frac{\mathrm{d}\,\hat{g}}{\mathrm{d}\,\mathbf{x}} = \frac{\mathrm{d}\,\hat{g}}{\mathrm{d}\,\mathbf{u}} = \hat{\mathbf{n}}^*_{\text{lead}} = \nabla_x \hat{g}, \tag{3.36}$$

where use is made of Eqs. (3.2) and (3.4).

With this notation, the following approximation for the location vector on the rigid leader-surface that corresponds to the closest-point projection of $\mathbf{x}$ can be formulated:

$$\hat{\mathbf{X}}_{\text{lead}} = \mathbf{x} - \hat{g}\frac{\nabla_x \hat{g}}{\|\nabla_x \hat{g}\|}. \tag{3.37}$$

The Neural-Pull loss function is then formulated as follows:

$$\mathcal{L}_{\text{pull}} = \|\hat{\mathbf{X}}_{\text{lead}} - \mathbf{X}_{\text{lead}}\|, \tag{3.38}$$

which indeed consists only of a single contribution. Particular of the Neural-Pull loss function is furthermore that it does not require the target signed distance, even though it is the NN's primary output. Instead, the target location vector on the rigid leader-surface is required, i.e. $\mathbf{X}_{\text{lead}}$.

It may be clear that an ANN trained with the Neural-Pull loss function does not exactly emulate the aforementioned conventional CDA. The reason is that the conventional CDA outputs C_{act}, P_{act} and $\underline{\underline{\xi}}^*$, whereas the NN trained with Neural-Pull outputs $\hat{g}$ and its first-order and second-order derivatives with respect to the location vector of the follower-vertex of interest, i.e. $\mathbf{x}$. Nevertheless, hardly any implementation efforts are required to adjust the full contact simulations (i.e. Algs. 8, 9 and 10) such that they are compatible with a NN trained using the Neural-Pull loss function.

3.5 Multi-task neural network

In Section 3.2, the conventional CDA was dissected in three separate algorithms. First, the broad phase (Alg. 5) finds follower-vertices close to the rigid leader-surface and stores them with potential patches in sets $\hat{C}_{\text{act}}$ and $\hat{P}_{\text{act}}$. Second, the first algorithm of the narrow phase (Alg. 6) determines suitable initial guesses for parametric surface coordinates $\hat{\underline{\underline{\xi}}}$ for the patches stored in $\hat{P}_{\text{act}}$. Third, the second algorithm of the narrow phase (Alg. 7) computes the exact closest-point projections, i.e. parametric surface coordinates $\underline{\underline{\xi}}^*$.

Unlike the NN trained by the Neural-Pull loss function of the previous section, the approach proposed in this section aims to replace the broad phase algorithm and the first algorithm of the narrow phase, whereas the second algorithm of the narrow phase remains untouched. To this purpose, a multi-task ANN is proposed. A multi-task NN is an ANN that emulates more than one task simultaneously. Fig. 3.7 highlights the difference between a multi-task NN that is constructed to perform two tasks and two separate NNs that each perform one task.

Based on Fig. 3.7, one can conclude that a multi-task NN is indeed a single NN, because the different branches responsible for each task share information during their evaluation. Consequently, a multi-task NN has to be trained for all tasks simultaneously. In contrast, a combination of NNs, where each NN is responsible for a single task, enables training each NN separately. This makes the training of the multi-task NN that is to replace a combination of NNs more time-consuming than the training of the individual NNs. On

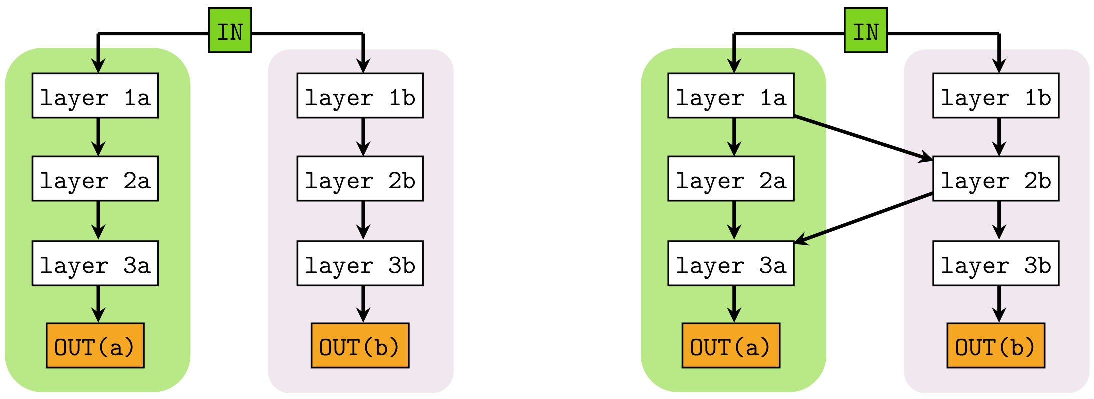


Figure 3.7: Illustrative comparison of conventional NNs (left) versus a multi-task NN (right). Left: Two NNs are presented that each emulate a separate task. Right: An NN emulates the same tasks as the two NNs on the right, but exchanges information between the green and pink branches. The exchange of information makes it a single NN, with the consequence that it must be trained as one. It is called 'multi-task' because it emulates two tasks simultaneously.

the other hand, thanks to the sharing of information between the branches, multi-task NNs can be substantially more accurate than a combination of conventional ANNs.

A simple representation of inputs and outputs of the envisioned multi-task NN is presented in Fig. 3.8. The input of the multi-task NN is the location vector of a follower-vertex of interest, i.e. $\mathbf{x}$ but represented by scalars x, y and z in Fig. 3.8. The multi-task ANN emulates three tasks. First, a (rough) signed distance approximation enables the discarding of follower vertices that are definitely not close to the rigid leader surface. As a second and substantially more important task, a classification is performed that finds the closest patch to the follower-vertex of interest. In fact, the classification does not find a single patch, but ranks the possibilities of all patches. The third and final task, equally important as the second task, is to provide a suitable initial guess for the parametric surface coordinates.

Particular in Fig. 3.8 is that not two parametric surface coordinates are emulated (i.e. the two scalars in $\underset{\sim}{\hat{\xi}}$ for the follower-vertex of interest), but $2n_\mathrm{p}$, where n_p denotes the total number of patches. The problem that would have arisen if only a single pair of parametric surface coordinates would be emulated is illustrated in Fig. 3.9 for the case in which only one parametric surface coordinate is of interest. The sketch shows that the parametric surface coordinate experiences sudden steps because it is bounded between zero and one within each patch. An ANN that is trained by means of regression would have difficulties in treating these sudden changes. The multi-task ANN has the advantage that deep layers, associated with the different tasks, can share information. In doing so, the geometric information relevant to all patches is shared between the classification branch and the projection branch.

Important for this type of construction of the multi-task NN is that only the specific parametric surface coordinates for the associated patch are trained. In other words, for a follower-vertex associated with patch 5 for instance, the multi-task NN should only train for the two parametric surface coordinates associated with patch 5. If the multi-task NN would also train for all other parametric surface coordinates, the accuracy of their

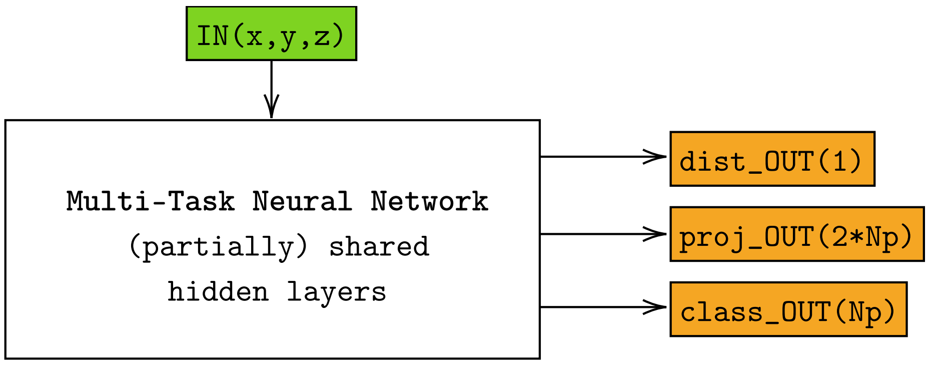


Figure 3.8: General concept of the proposed Multi-Task neural network, highlighting the inputs and outputs. The inputs (green) are the three scalars of the location vector of follower-vertex of interest. The outputs (orange) are a scalar quantifying the signed distance, $2n_\mathrm{p}$ parametric surface coordinates, and n_p scalars between zero and one that rank the patches most likely to contain the closest-point projection.

prediction reduces for the instances that they are the ones of interest.

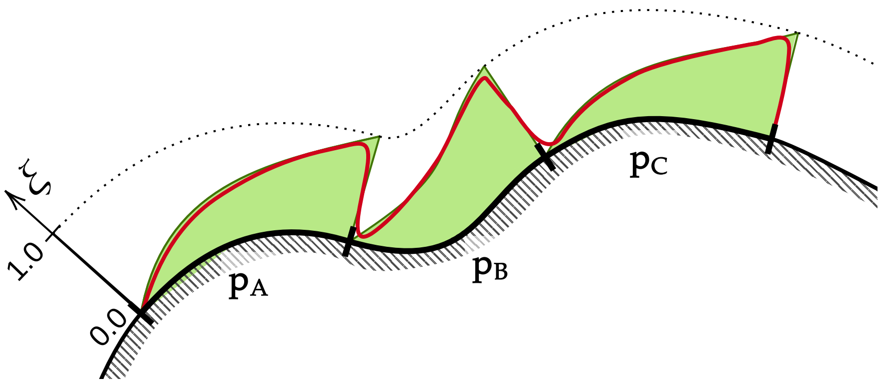
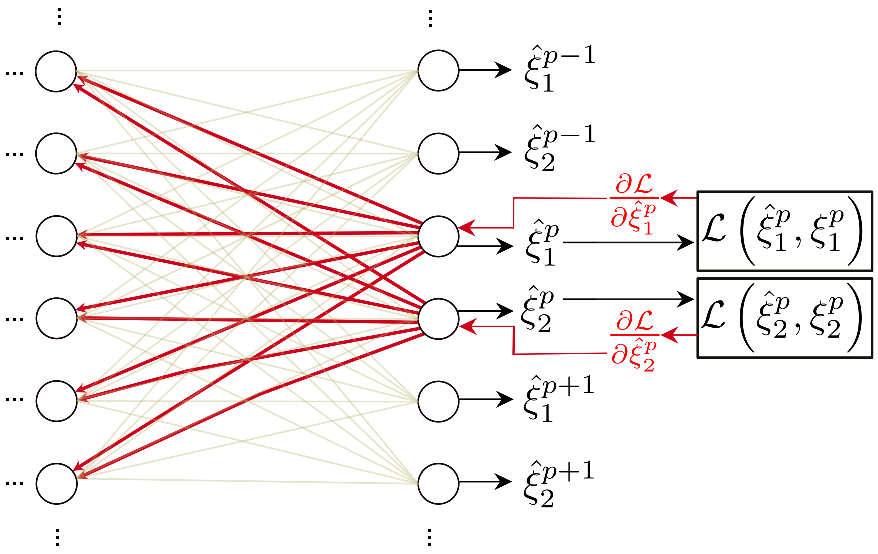


Figure 3.9: 2-D Representation: A single projection regression output (red line) for a multi-patch body introduces significant errors at patch boundaries. The exact surface coordinate ξ is depicted in green

To ensure that back-propagation occurs only from the appropriate output pair (ξ_1, ξ_2), a segmented regression approach is proposed. This is illustrated graphically in Fig. 3.10. It is achieved using the following loss function:

$$\mathcal{L}_\mathrm{proj} = \left\|\hat{\hat{\underline{\underline{\xi}}}}_\mathrm{p} - \underline{\xi}^*\right\|^2, \tag{3.39}$$

where $\underline{\xi}^*$ denotes the (two) target parametric surface coordinates for the combination of the follower-vertex and patch of interest (which is not the initial guess for the parametric surface coordinates that the multi-task NN is supposed to predict). Furthermore, $\hat{\hat{\underline{\underline{\xi}}}}$ denotes the matrix containing all $2n_\mathrm{p}$ parametric surface coordinate predictions that the multi-task NN outputs. Two hats are placed on top of the matrix to unify the notation with respect to the aforementioned equations of this chapter. One is used because the prediction of the multi-task NN is only an approximation. The other one is because the multitask NN predicts the initial guess for the parametric surface coordinates and not the exact ones.



$\hat{\xi}_1^{p-1}$
$\hat{\xi}_2^{p-1}$
$\frac{\partial \mathcal{L}}{\partial \hat{\xi}_1^p}$
$\hat{\xi}_1^p$
$\mathcal{L}\left(\hat{\xi}_1^p, \xi_1^p\right)$
$\hat{\xi}_2^p$
$\frac{\partial \mathcal{L}}{\partial \hat{\xi}_2^p}$
$\mathcal{L}\left(\hat{\xi}_2^p, \xi_2^p\right)$
$\hat{\xi}_1^{p+1}$
$\hat{\xi}_2^{p+1}$

Figure 3.10: Segmented regression ensures that back-propagation (red arrows) occurs only for the (ξ_1, ξ_2) pair corresponding to the true closest patch p.

However, the most important aspect here is that the subscript p denotes that only the parametric surface coordinates associated with patch p are extracted.

Another way to formulate this loss function, that is more in line with common NN notation and suits the precise implementation reported in Appendix C, is as follows:

$$\mathcal{L}_{\text{proj}} = \sum_{p=1}^{n_p} (\hat{\xi}_{\tau,p} - \xi^*_{\tau,p})^2 y_{\text{class},p}, \quad \tau \in \{1, 2\}, \tag{3.40}$$

where $y_{\text{class},p}$ denotes the one-hot encoding of the target patch number, $\hat{\hat{\xi}}_{\tau,p}$ and $\xi^*_{\tau,p}$ denote the multi-task NN's prediction and target value of parametric surface coordinate τ on patch p.

The total cost function for this task can then be written as follows:

$$\mathcal{J}_{\text{proj}} = \frac{1}{n_m} \sum_{i=1}^{n_m} \mathcal{L}^{(i)}_{\text{proj}}, \tag{3.41}$$

with n_m denotes the number of training samples. Consequently, the total cost function for the entire multi-task NN reads:

$$\mathcal{J} = \mathcal{J}_{\text{MSE}} + \mathcal{J}_{\text{proj}} + \mathcal{J}_{\text{CE}}, \tag{3.42}$$

where $\mathcal{J}_{\text{MSE}}$ denotes the cost function for the signed distance (using the loss function of Eq. (3.16)) and $\mathcal{J}_{\text{CE}}$ the cost function for the patch classification (using the loss function of Eq. (3.17)).

3.6 Results

The current section presents the generation of the training data, the accuracy of the relevant quantities obtained with Neural-Pull, the accuracy of the relevant quantities obtained with the multi-task NN, and finally both the accuracy and performance of the multi-task NN as a replacement of the CDA in the contact simulations. First, however, the unilateral contact simulations of the previous chapter are concisely repeated, with one difference with respect to the previous chapter highlighted.

The simulations consider frictionless unilateral node-to-segment contact between a 3D deformable follower-cube and a rigid leader-surface that neither translates nor rotates. Gregory patches are responsible for the smoothness of the rigid leader-surface. 96 Gregory patches are used, i.e. $n_\mathrm{p} = 96$, $P = \{1, 2, \ldots, 96\}$. The cube's outer finite element (FE) vertices function as follower-vertices. In some simulations, the cube is made to behave hyperelastically and in others hyperelastoplastically. Different trilinear hexagonal discretizations with eight Gauss quadrature points per FE are furthermore applied to the cube.

The difference from the previous chapter is that simulation type III is introduced, whereas simulation type I is not considered anymore. As Fig. 3.11 illustrates, the displacement vectors of all vertices of the cube's top surface are prescribed during simulation type III. The cube only translates in the positive x-direction during type III simulations.

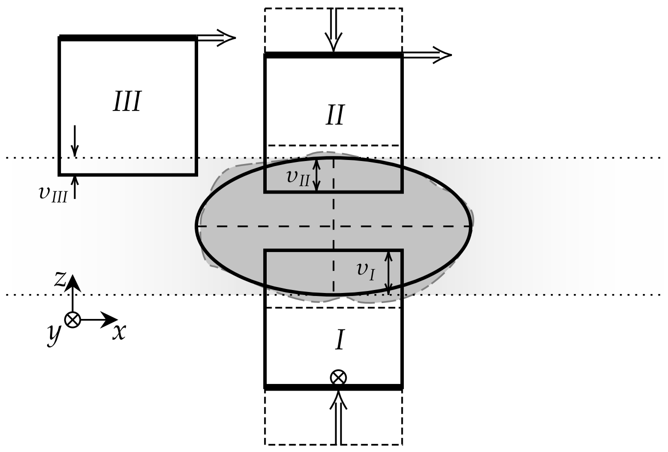


Figure 3.11: Sketches of the three types of 3D simulations considered in this thesis. The cubic deformable follower-body of size $4.00 \times 4.00 \times 4.00$ is presented thrice and indicated by Roman numerals I, II and III. The ellipsoidal rigid leader-body is presented in grey. Simulations of type I are not considered in the current chapter. $v_{II} = 1.00$ and $v_{III} = 0.50$ denote the maximal overlaps in z-direction that occurs during the two types of simulations.

3.6.1 Training data

The training and validation data are generated by sampling points around and inside the rigid leader-body. The bounding box for these points is defined by extending the minimum and maximum coordinates of the rigid leader-body by 10% in each direction, as shown in Fig. 3.12. A uniform cloud of $200 \times 200 \times 200$ points is generated, resulting in 8×10^6 points. For each point, a collection of quantities is stored. The collection consists of the index of the patch that contains the closest-point projection, the parametric surface coordinates of the closest-point projection, the signed distance, the signed distance's gradient $\frac{\partial g}{\partial \mathbf{u}} = \mathbf{n}^*_\mathrm{lead}$ and the six unique components of the symmetric Hessian tensor $\frac{\partial^2 g}{\partial \mathbf{u}^2}$.

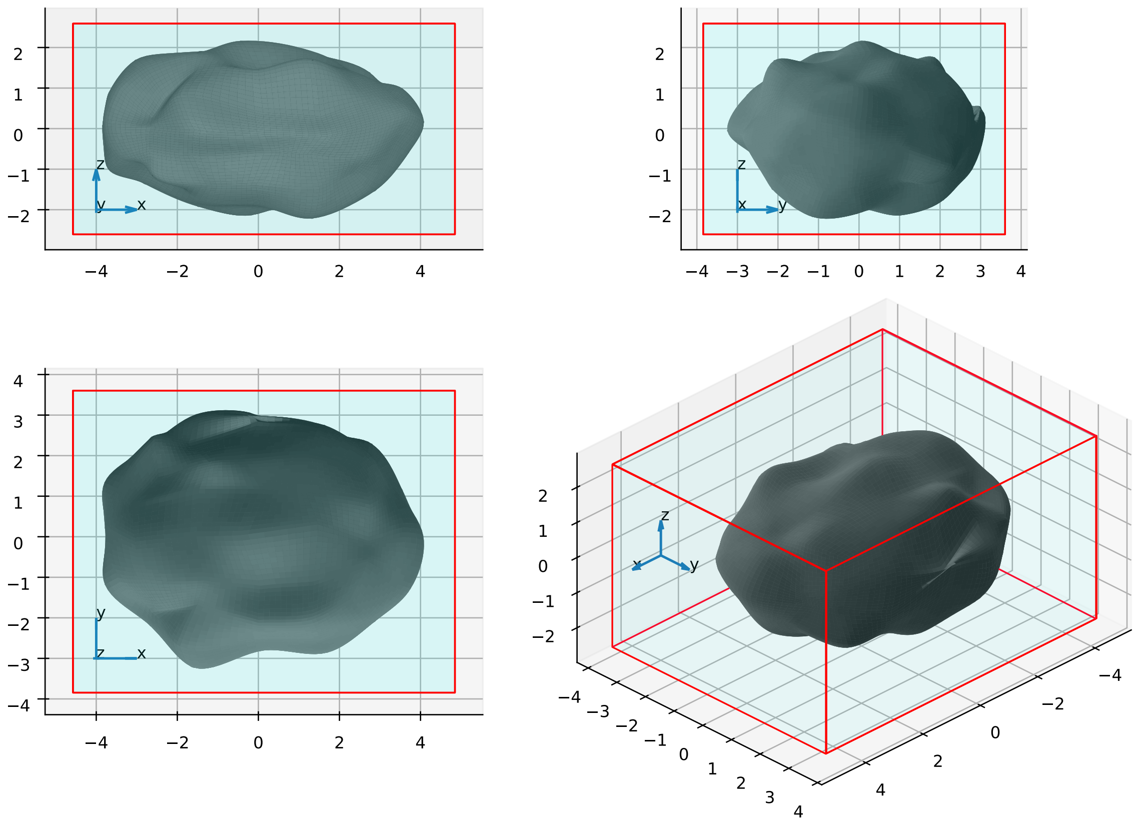


Figure 3.12: The rigid leader-surface enclosed in the bounding box used for the generation of training data.

From the generated point cloud, only points with a signed distance $g \in [-0.5, 1.5]$ are retained for further use. This reduces the size of the dataset to approximately 5.6 million, thereby accelerating the training. It also concentrates the dataset on relevant regions near the rigid leader-surface, thereby improving the accuracy of the NNs where it matters. To ensure robust model validation, 20% of the generated points are randomly selected for validation purposes during NN training. For illustration purposes, Fig. 3.13 shows heatmaps at $x = 0$ for the signed distance and the x-component of $\mathbf{n}^*_{\text{lead}}$. The images highlight the spatial variation of the quantities.

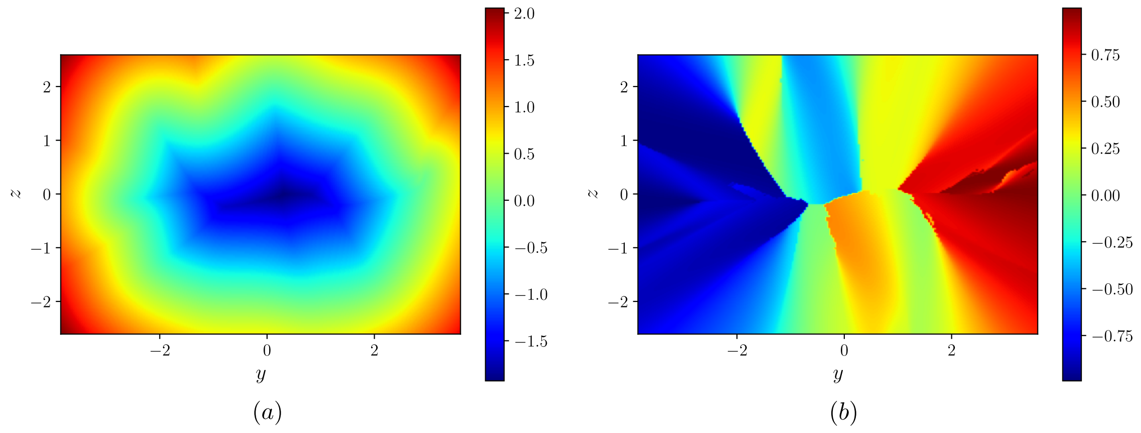


Figure 3.13: Heatmaps for a single slice of the rigid leader surface in the plane with its normal in x-direction: (a) signed distance g, (b) x-component of $\mathbf{n}^*_{\text{lead}}$.

3.6.2 Neural-Pull

The conventional ANN trained with the Neural-Pull loss function consists of four hidden layers, each containing 512 neurons. The ReLU activation function is used for all neurons, except for those of the layer before the output layer. The sigmoid activation function is used for these neurons. The Adam optimizer is used for training, with a learning rate of $\alpha = 10^{-4}$. The batch size was set to 512 samples. It is relevant to mention that during investigations not reported here, the number of hidden layers was varied, as were the number of neurons per layer, the activation functions, the hyperparameters of Adam, as well as that Adadelta was investigated as the training algorithm.

The accuracy of the Neural-Pull ANN is expressed in terms of absolute errors that are computed for the validation data only. The expressions for the errors presented below for the emulated signed distance, its gradient and Hessian, respectively, read:

$$e(\hat{g}) = |\hat{g} - g|, \tag{3.43}$$

$$e(\hat{\mathbf{n}}_{\text{lead}}) = \sqrt{(\hat{\mathbf{n}}_{\text{lead}} - \mathbf{n}^*_{\text{lead}}) \cdot (\hat{\mathbf{n}}_{\text{lead}} - \mathbf{n}^*_{\text{lead}})}, \tag{3.44}$$

$$e\left(\widehat{\frac{\mathrm{d}\,\mathbf{n}_{\text{lead}}}{\mathrm{d}\,\mathbf{u}}}\right) = \sqrt{\left(\widehat{\frac{\mathrm{d}\,\mathbf{n}_{\text{lead}}}{\mathrm{d}\,\mathbf{u}}} - \frac{\mathrm{d}\,\mathbf{n}^*_{\text{lead}}}{\mathrm{d}\,\mathbf{u}}\right) : \left(\widehat{\frac{\mathrm{d}\,\mathbf{n}_{\text{lead}}}{\mathrm{d}\,\mathbf{u}}} - \frac{\mathrm{d}\,\mathbf{n}^*_{\text{lead}}}{\mathrm{d}\,\mathbf{u}}\right)}, \tag{3.45}$$

where the hats on top of the symbols refer to outputs of the NN and the symbols without hats refer to their target quantities.

The error of the NN's primary output, the signed distance, is presented as a distribution in Fig. 3.14. The distribution in Fig. 3.14 shows that the error does not depend on the

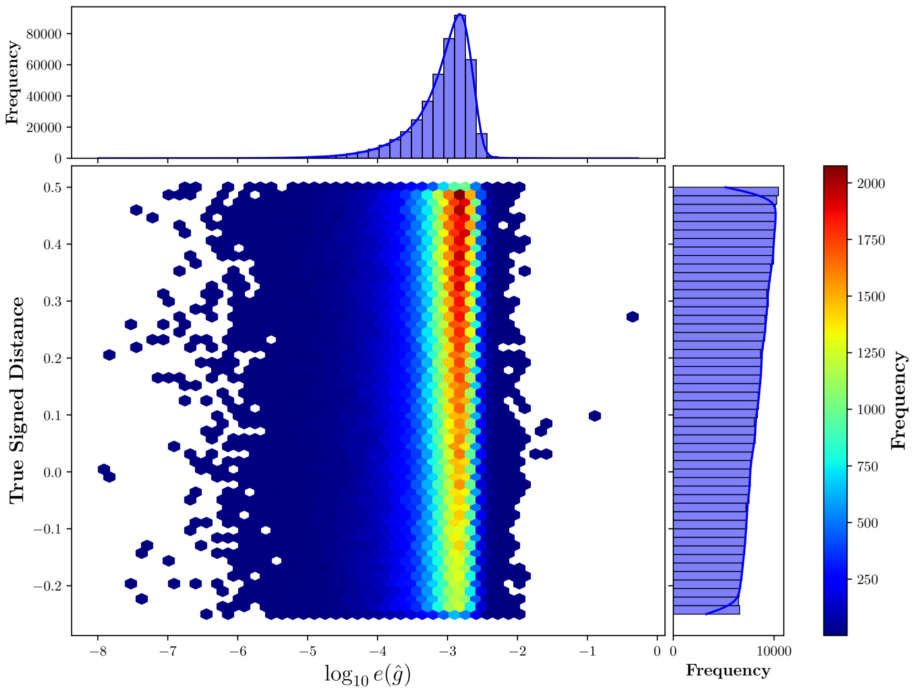


Figure 3.14: Neural-Pull: Error distribution for the signed distance.

distance between the point and the rigid leader-surface (vertical error). It also shows that the worst error of the signed distance is approximately 10^{-2} (neglecting two outliers), and the best error is approximately 10^{-8}. The mean error is approximately 10^{-3}. This can be deemed accurate considering that the smallest length of the ellipsoid's semi-axis is 2.00 units (see Fig. 3.12).

Fig. 3.15 presents the same distribution for the gradient of the signed distance. This distribution also shows no significant effect of the signed distance on accuracy. The mean error of approximately $10^{-1.75} = 0.018$ is not too bad considering that the target vector is a normal vector, i.e. has a length of one. Nevertheless, for some validation points the error reaches values of more than 1. The reason for this becomes clear from comparing the image on the left in Fig. 3.13 to the one on the right. The left image, which shows the field of the signed distance is visually continuous. The right image, however, which shows the field of (the x-component of) $\mathbf{n}^*_{\text{lead}}$ and is currently under investigation, shows that $\mathbf{n}^*_{\text{lead}}$ is not continuous. Hence, the ANN's predictions for the signed distance's gradient (i.e. $\hat{\mathbf{n}}_{\text{lead}}$) are less accurate.

Finally, Fig. 3.16 again presents the same distribution, but this time for the Hessian. The error increases relative to the one for the gradient (Fig. 3.15), but a direct comparison is not applicable since the error for the gradient is defined using the L^2-norm (Eq. (3.44)) and the one for the Hessian using the Frobenius norm (Eq. (3.45)). It is worth to realize that the Neural-Pull loss function does not train the NN for the Hessian, whereas it does so for the gradient (and the signed distance).

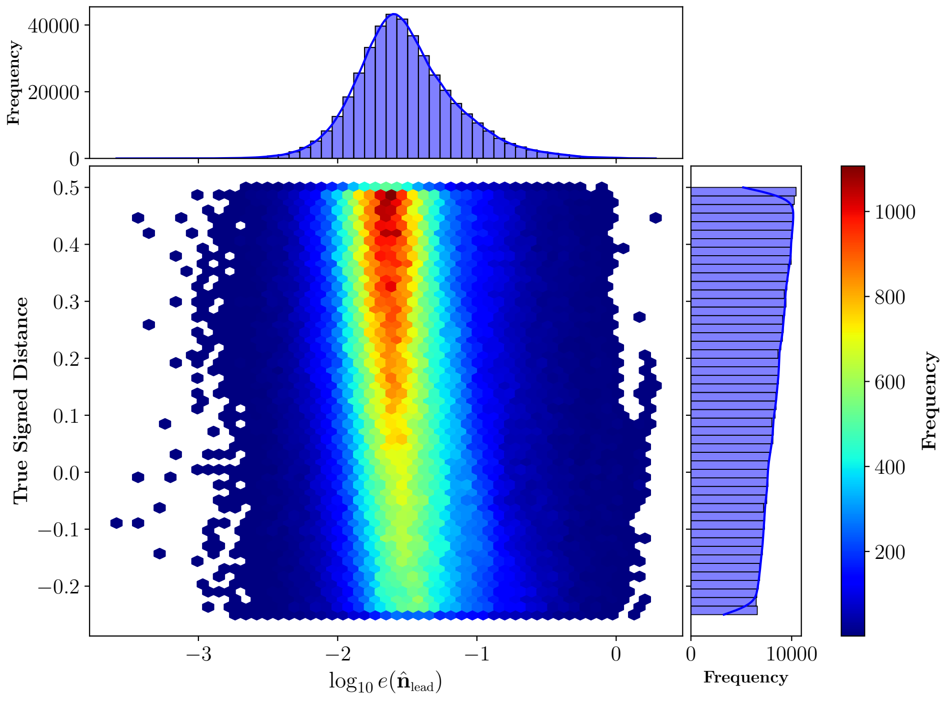


Figure 3.15: Neural-Pull: Error distribution for the gradient.

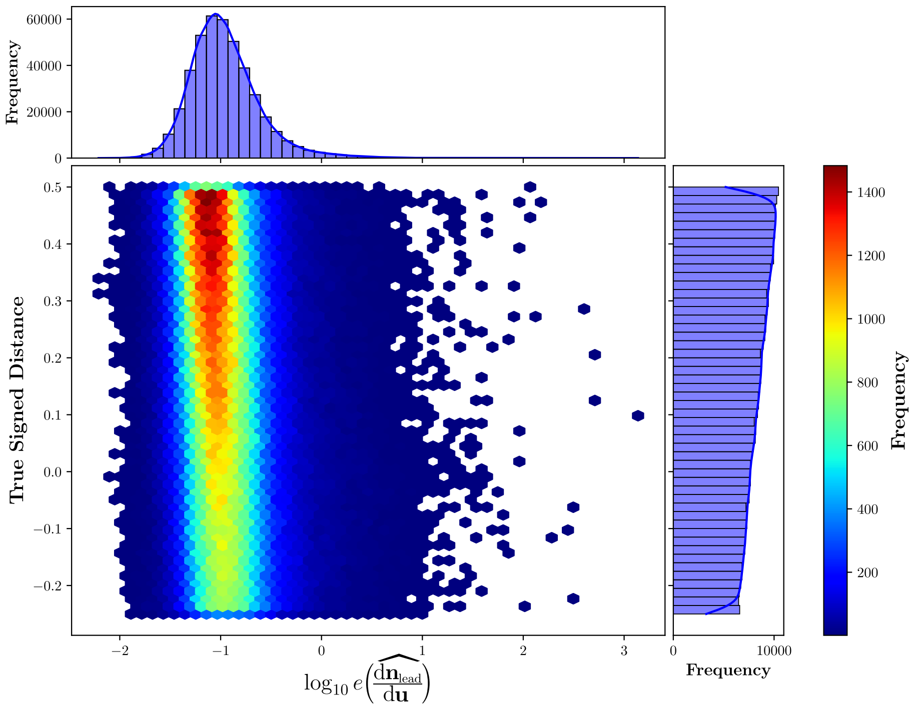


Figure 3.16: Neural-Pull: Error distribution for the Hessian.

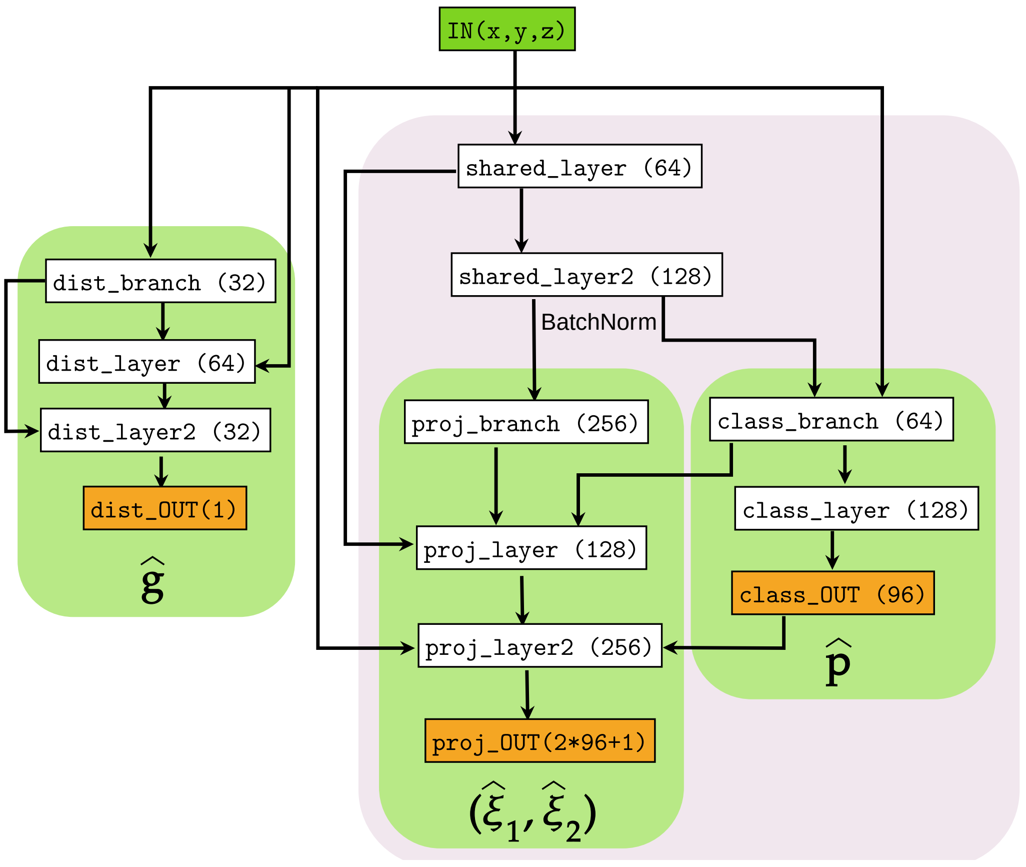


Figure 3.17: Multi-task NN: Architecture. Every arrow represents a full connection between layers. Layers show the number of neurons between parenthesis. Every task branch is enclosed in a green rectangle.

Although the accuracy of the NN trained by the Neural-Pull loss function is undoubtedly sufficient to visualize shapes as reported in [89], use of the NN in the overall simulation algorithm did not yield convergence of Newton's method. Consequently, no additional results could be generated for this NN.

3.6.3 Multi-task neural network

The final architecture of the multi-task ANN is presented in Fig. 3.17. The number of outputs for each layer (i.e. the number of neurons) is specified in parentheses. The neurons of the hidden layers employ the ReLU activation function. The neurons of the output layers do not use any activation function, except those of the classification branch which use the softmax activation function. One can furthermore observe that the output of the regression task is of size $2n_\mathrm{p} + 1$. One additional output is required for the training using the segmented regression of the previous section. After training, however, this additional output is ignored. The multi-task ANN also uses several skip-connections as can be seen in Fig. 3.17.

The multi-task ANN is trained for 1000 epochs with the Adadelta optimizer. The Adam optimizer was also examined, but performed worse. The learning rate is set to $\alpha = 0.001$, the decay rate to $\beta = 0.95$ and the denominator conditioner to $\epsilon = 10^{-7}$. Each batch contains 32 samples. The cost functions of the different tasks are shown in Fig.3.18 for

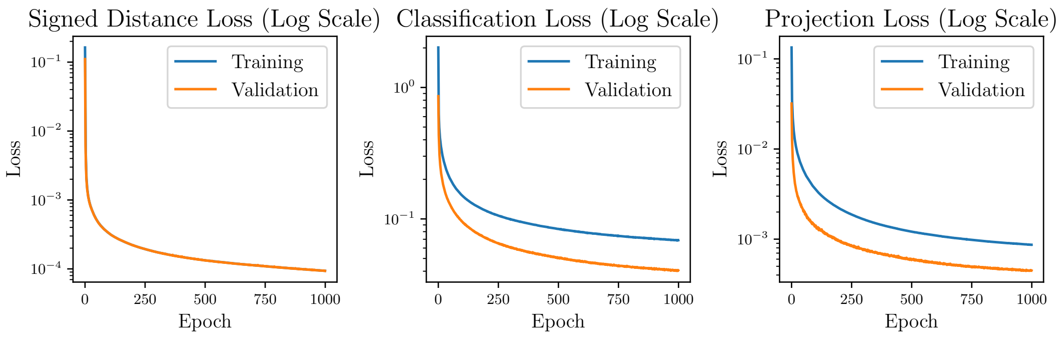


Figure 3.18: Multi-task NN: The cost functions of the different tasks. Left diagram: signed distance, center diagram: patch classification, right diagram: parametric surface coordinates.

the course of the training. The curves show stably decreasing trends. The classification and projection tasks show a correlated learning behaviour due to the connections between the two branches (see Fig. 3.17).

Accuracy of the multi-task neural network

As mentioned in section 3.5, the multi-task NN's signed distance predictions are only envisioned to serve as a filter. For this reason, a somewhat reduced accuracy of the signed distance prediction is acceptable, and a rather modest sub-architecture for the signed distance is used (left branch in Fig. 3.17). Despite this, Fig. 3.19 shows that the mean error of the signed distance is only one order of magnitude larger than that of the NN trained with Neural-Pull (10^{-2} versus 10^{-3}). Consequently, the multi-task NN in the contact simulations filters at a threshold of $g = 0.10$, which is a fairly conservative choice as can be seen in Fig. 3.19.

Fig.3.20 illustrates the accuracy of the patch classification of the multi-task NN by means of three confusion matrices. The first axis of the confusion matrix corresponds to the target indices and the second axis to the predicted indices. By recording the combination of target index and predicted index for a single sample, the counter of one square of the confusion matrix is increased by one. This is repeated for all samples. Thus, in a perfect scenario, only the squares on the diagonal are populated and the counters of the off-diagonal squares remain zero.

The top-left confusion matrix in Fig. 3.20 visualizes the multi-task NN's classification prediction without any training. The confusion matrix indicates that the multi-task NN effectively classifies randomly. After training with five epochs however, the top-right confusion matrix in Fig.3.20 illustrates that the multi-task NN has already learned substantially, since the multi-task NN maps substantially more samples to the diagonal squares - which it is supposed to do. The bottom-left confusion matrix in Fig. 3.20 illustrates the capabilities of the multi-task NN after it is trained for 1000 epochs. Relative to the top-right confusion matrix, it can be seen that the counters of the diagonal squares have increased even more at the desired expense of the counters of the off-diagonal squares after training is completed for 1000 epochs.

The final confusion matrix is not perfect because the counters of some off-diagonal squares are not zero. To investigate the mistakes of the multi-task ANN, the confusion matrix

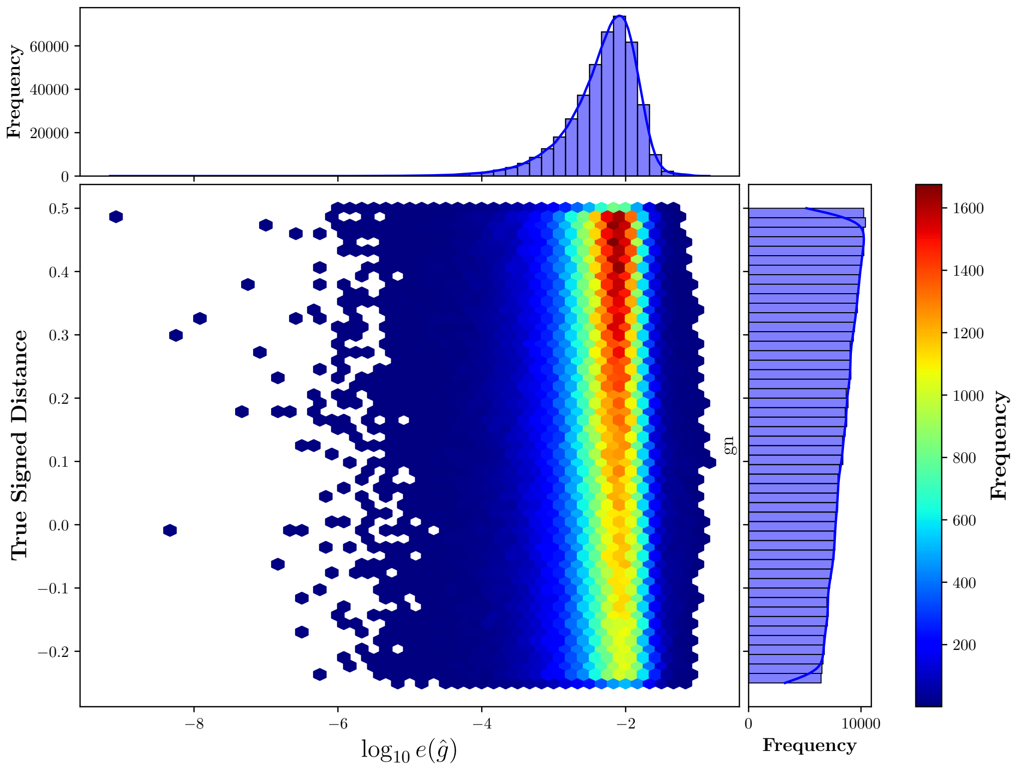


Figure 3.19: Multi-task NN: Error distribution of the signed distance.

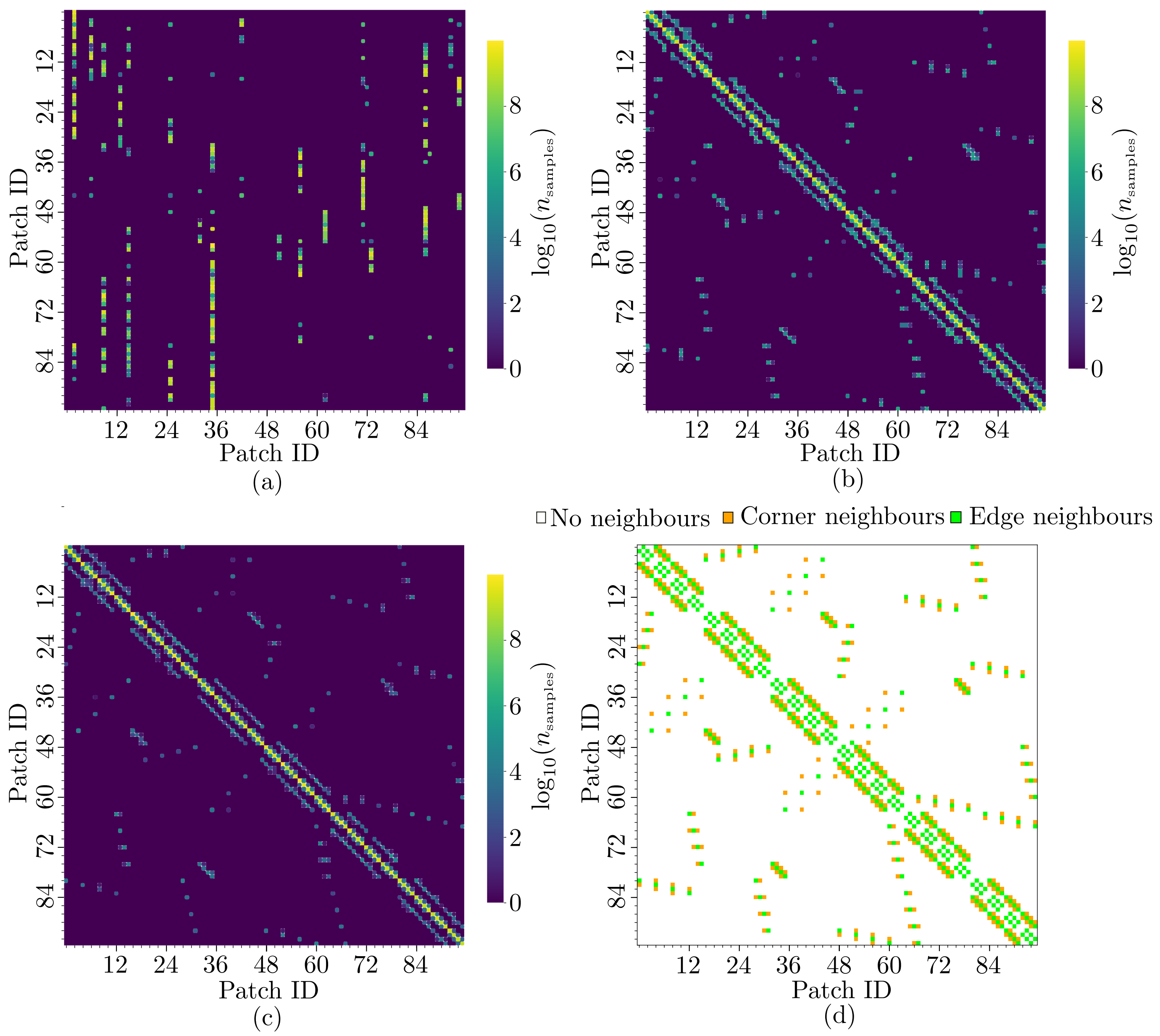


Figure 3.20: Multi-task NN: Confusion matrices for closest patch prediction after a) 0 epochs, b) 5 epochs, c) 1000 epochs of training. The neighbourhood matrix of the 96 patches is shown in diagram d).

can be compared to the neighbourhood matrix in the bottom-left of Fig.3.20. The neighbourhood matrix illustrates the neighbours of each patch. A white square means that two patches are not neighbours, a green square means that two patches share an edge, and an orange square means that two patches share a vertex. The similarity of the confusion matrix and the neighbourhood matrix indicates that if the multi-task ANN makes a mistake, it has predicted the patch with which the target patch shares an edge in most of the cases.

Next, the accuracy of the multi-task NN's predictions of the parametric surface coordinates is investigated. To this purpose, the L^2-norm (Euclidian norm) of the difference between the predicted and target parametric surface coordinates is recorded for each validation sample. Fig. 3.21 presents the error distribution individually for each patch. Important to realize is that the results in Fig. 3.21 are generated without interference of the classification branch of the multi-task NN. In other words, potential mistakes of the classification predictions have no influence on Fig. 3.21.

Fig. 3.21 shows that the average error is approximately $10^{-1.6} = 0.025$. This seems like a decent result for two reasons. The first reason is that each parametric surface coordinate is bounded between 0 and 1 and hence, an error of 0.025 relative to a value 1 is not bad. The second reason is that the multi-task NN will only be used to predict the initial guesses of the parametric surface coordinates and not the exact ones. Hence, Newton's method will subsequently be employed to compute the exact parametric surface coordinates (since the second algorithm of the narrow phase of Alg. 7 will remain untouched and will remain in place in the envisioned CDA.

Fig. 3.21 illustrates two more points. First, the accuracy of the predicted parametric surface coordinates clearly depends on the patch. Second, for some patches, the maximum error seems substantial.

In addition, it is interesting to investigate if the accuracy of the multi-task NN's prediction of the parametric surface coordinates depends on the location inside a patch. For this purpose, Fig. 3.22 presents the error distributions of the individual parametric surface coordinates as a function of the target values for patch 50. The distributions show that the errors do not seem to depend on the location inside patch 50.

Accuracy of the hybrid CDA

The next investigation focuses on the accuracy of the newly envisioned CDA that results from replacing the broad phase algorithm (Alg. 5) and the first algorithm of the narrow phase (Alg. 6) with the multi-task NN (Fig. 3.17). Thus, the current investigation concentrates on the accuracy that results from combining the multi-task NN and Alg. 7.

Similarly to the presentation of the accuracy of the NN trained with Neural-Pull, the same errors of the signed distance, the signed distance's gradient and Hessian are presented in the left column that the hybrid CDA has predicted for the validation samples. Interestingly, the hybrid CDA's predictions of the signed distance, its gradient and Hessian are near machine precision for the vast majority of the validation samples, and terribly large for a small minority. Newton's method in Alg. 7 must be credited for reaching near machine precision errors. The substantially large errors that occur for a small minority of the samples is caused by inaccuracies of both the patch classification branch and the branch responsible for the parametric surface coordinate predictions.

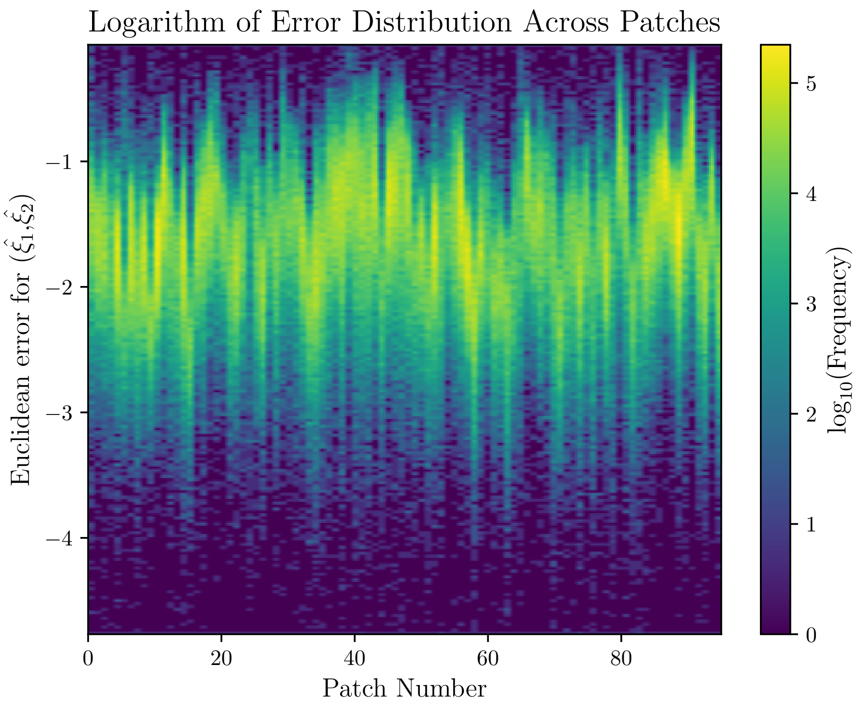


Figure 3.21: Multi-task NN: Error distribution of the parametric surface coordinates for each patch.

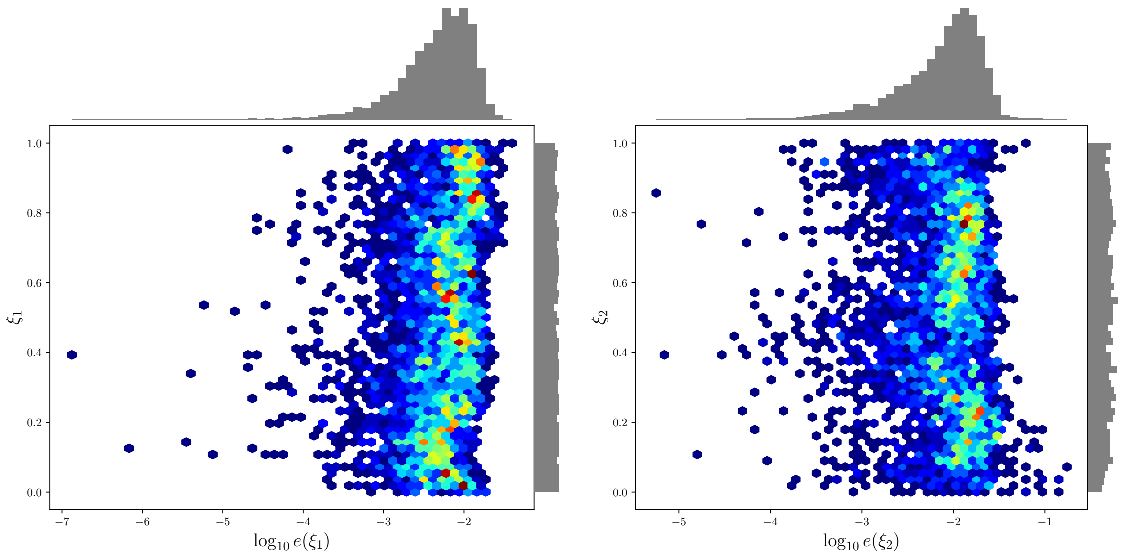


Figure 3.22: Multi-task NN: Error distribution per paramatric surface coordinate for patch 50.

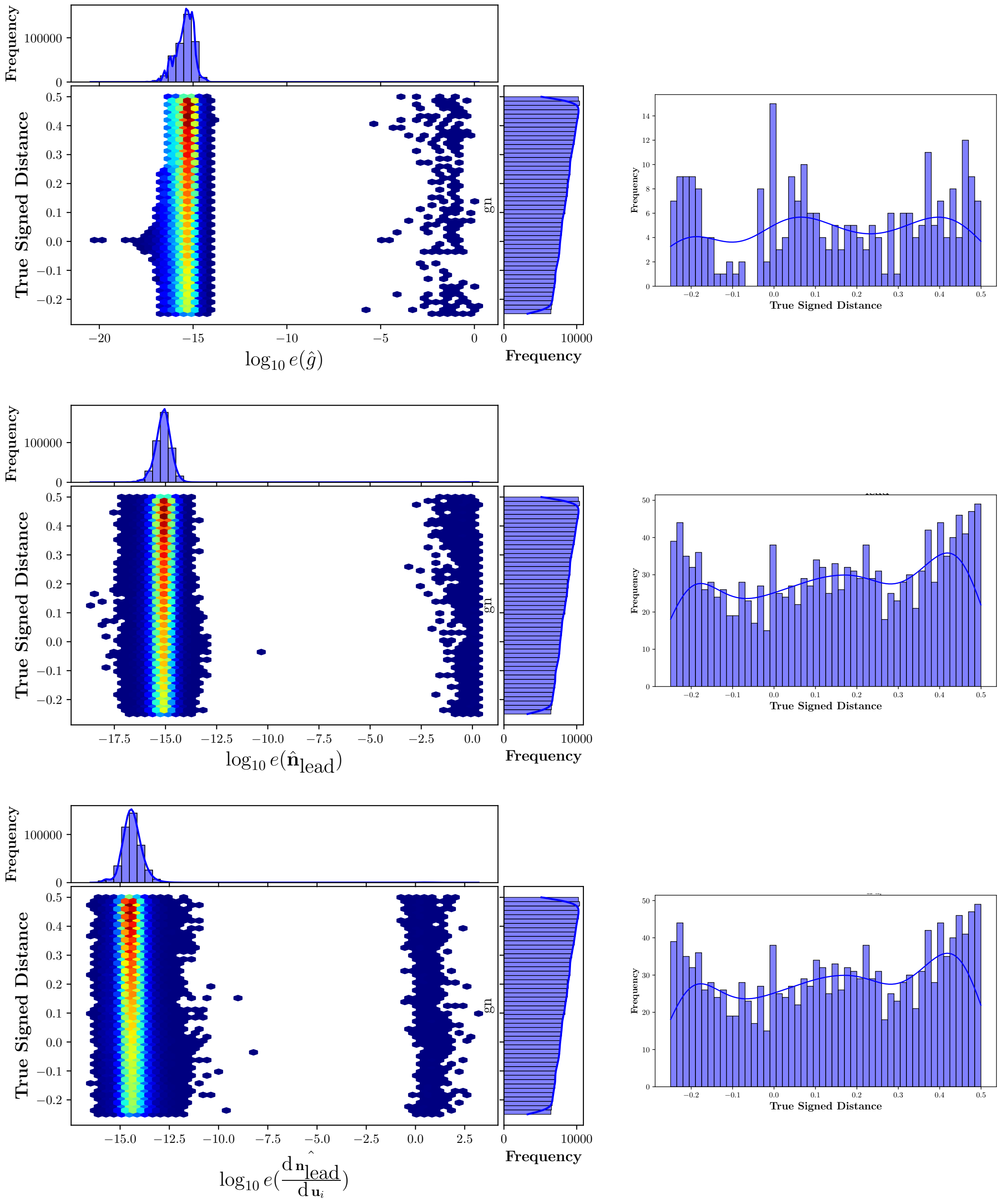


Figure 3.23: Hybrid CDA. Left column: Error distributions of the signed distance, its gradient and Hessian. Right column: Each distribution on the left is split in two at an error of 10^{-10}. The distribution of the validation samples that fall in the domain on the right of an error of 10^{-10} is presented as a function of the signed distance.

More precisely, if the distributions in the left column of Fig. 3.23 are split into two domains at an error of 10^{-10}, it can be stated that the error of the signed distance falls below 10^{-10} for 99.94% of the validation samples. The errors of the gradient and the Hessian fall below 10^{-10} for 99.54% of the validation samples. The next question is whether or not the validation samples with a substantial error (i.e. those with an error of more 10^{-10}) show some dependency on the signed distance. The distributions in the right column of Fig. 3.23 show that this is not the case for the signed distance. For the gradient and Hessian, however, an increase of the signed distance increases the chance of a completely wrong prediction of the hybrid CDA.

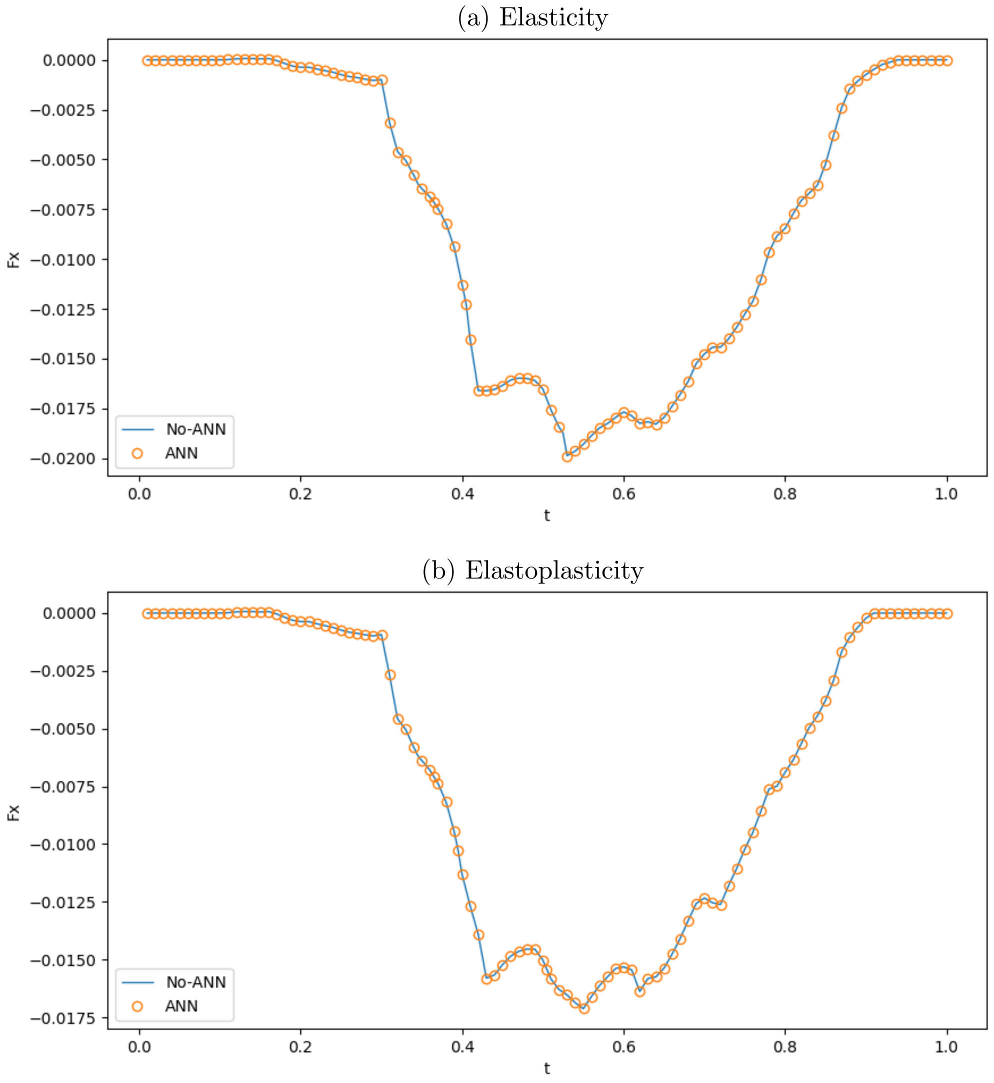


Figure 3.24: Simulation type II, $5 \times 5 \times 5$ FEs. The forces in x-direction predicted by the conventional simulation and the hybrid CDA simulation as a function of the pseudo-time.

Performance of the hybrid CDA contact simulations

Finally, the performance of contact simulations enhanced by the hybrid CDA is investigated. First, a simulation of type II is considered in which the deformable follower-cube behaves elastically and is discretized using $5 \times 5 \times 5$ FEs ($n_{\text{var}} = 540$). This simulation is completely solved by Newton's method. The forces in the x direction predicted by the

Table 3.2: Performance of simulations that do not require minimization solvers. Times (in seconds) taken for different parts of the simulation. The last column does not have units since it reports the number of Newton iterations, including accepted and rejected Newton increments.

type	deformable	method	Total	K+solv	f_{mat}	CDA'+f_{con}	CDA	Newt iter
II	5×5 × 5 elastic	no ANN	406.88	56.41	14.69	196.79	127.63	1029
		ANN	197.72	47.32	11.90	111.94	18.28	982
	5×5 × 5 plastic	no ANN	387.45	49.39	38.20	167.31	121.31	891
		ANN	236.04	49.05	36.64	113.90	19.27	884
III	15×15 × 15 elastic	no ANN	13922.90	8010.07	1173.25	2443.12	1905.63	2811
		ANN	10180.17	7487.15	1101.67	1015.80	204.27	2855
	15×15 × 15 plastic	no ANN	13203.27	7511.77	1224.12	2340.70	1851.06	3261
		ANN	9409.99	6940.42	1147.12	881.90	182.05	2943

conventional simulation and the hybrid CDA simulation over the course of the simulation are presented in the top diagram of Fig. 3.24. The forces match nearly perfectly. Some minor differences do occur, but they can be barely observed with the naked eye.

The computational times of the two methods for this simulation are presented in the first two rows of Table 3.2 (type II, 5 × 5 × 5, elastic). Also, the performance indicates that the two simulations behave slightly differently, since the number of Newton iterations is slightly different. As a consequence, the times required to construct the stiffness matrix, to solve the linear system (K + solve in Table 3.2) and to construct the part of the force column associated with the material behaviour (f_{mat} in Table 3.2) are also somewhat different, although they are not directly affected by the CDA.

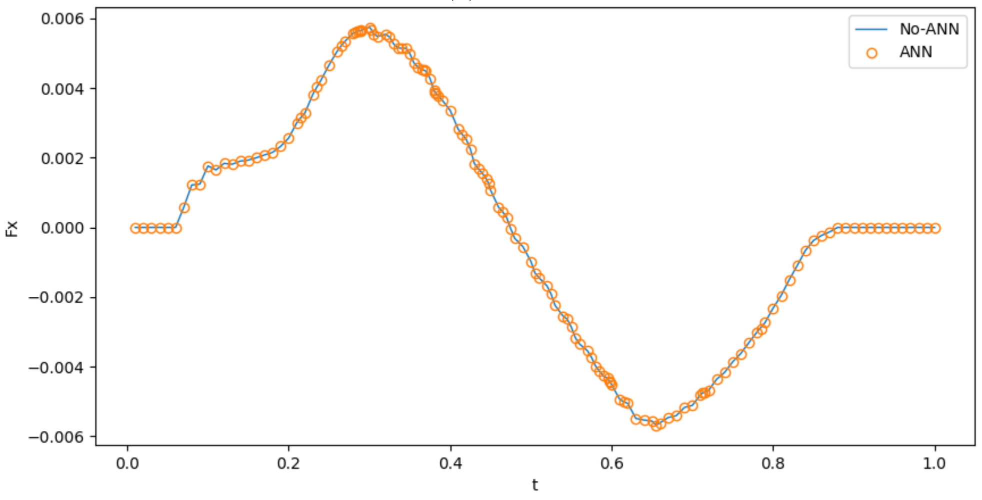
In this contact simulation, which is completely based on Newton's method, the CDA is exploited twice (see Alg. 10); once in its entirety when Newton's method has converged (CDA in Table 3.2), and once in its simplified form for each Newton iteration (reported together with the construction of the force column associated with contact in Table 3.2). Consequently, the most significant accelerations can be observed when the CDAs are employed. For this simulation, the hybrid CDA accelerates the simulation by a factor of two.

The next simulation setup is the same as the previous one, but elastoplasticity instead of elasticity is considered. The force-time curves in the bottom diagram of Fig. 3.24 show an excellent match. In this case, it is expected that the relative acceleration obtained by the hybrid CDA is somewhat less because the return mapping algorithm entails that more time is required to construct the part of the force column associated with the constitutive material law.

The third and fourth rows in Table 3.2 indeed show that the construction of the force column associated with the constitutive material law is substantially increased compared to the previous elastic simulation. Consequently, the relative acceleration has also reduced relative to the elastic simulation, as can be observed by comparing the total simulation times. It can furthermore be observed that the simulation time of the elastoplastic simulation with the conventional CDA has been reduced relative to that of its elastic equivalent (even though the return mapping is performed). This is caused by fact that 10% less Newton iterations are required in the elastoplastic simulation.

Next, the simulation type is changed from II to III, hyperelasticity is considered, and a discretization of 15 × 15 × 15 FEs is used ($n_{\text{var}} = 11520$). The force-time curves presented for this simulation in the top diagram of Fig. 3.25 match perfectly. Since type III simulations were not considered in the previous chapter, the deformations predicted for different pseudo-times for hyperelasticity are presented in the left column of Fig. 3.26 for completeness.

(a) Elasticity



(b) Elastoplasticity

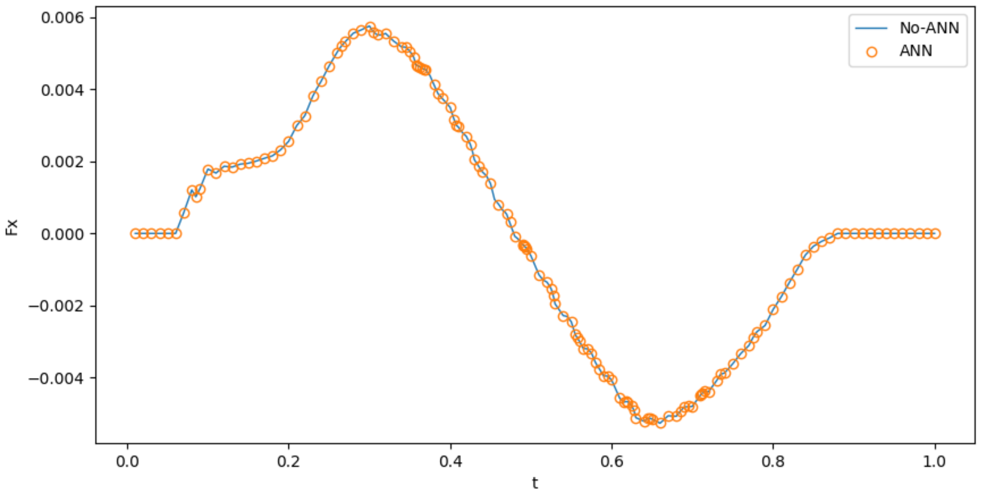


Figure 3.25: Simulation type III, $15 \times 15 \times 15$ FEs. The forces in x-direction predicted by the conventional simulation and the hybrid CDA simulation as a function of the pseudo-time.

What makes this simulation interesting in relation to the previous ones is that the number of variables is drastically increased (540 versus 11520). Consequently, the time of the linear solver in relation to the total simulation time is substantially increased. Consequently, the relative time savings obtained by the hybrid CDA are less, although they substantially increase in the absolute sense, as the fifth and sixth rows in Table 3.2 indicate.

Because the acceleration achieved with the hybrid CDA is only a factor of 1.4, an important note is to be made here. Solving the systems of linear equations is performed with a direct

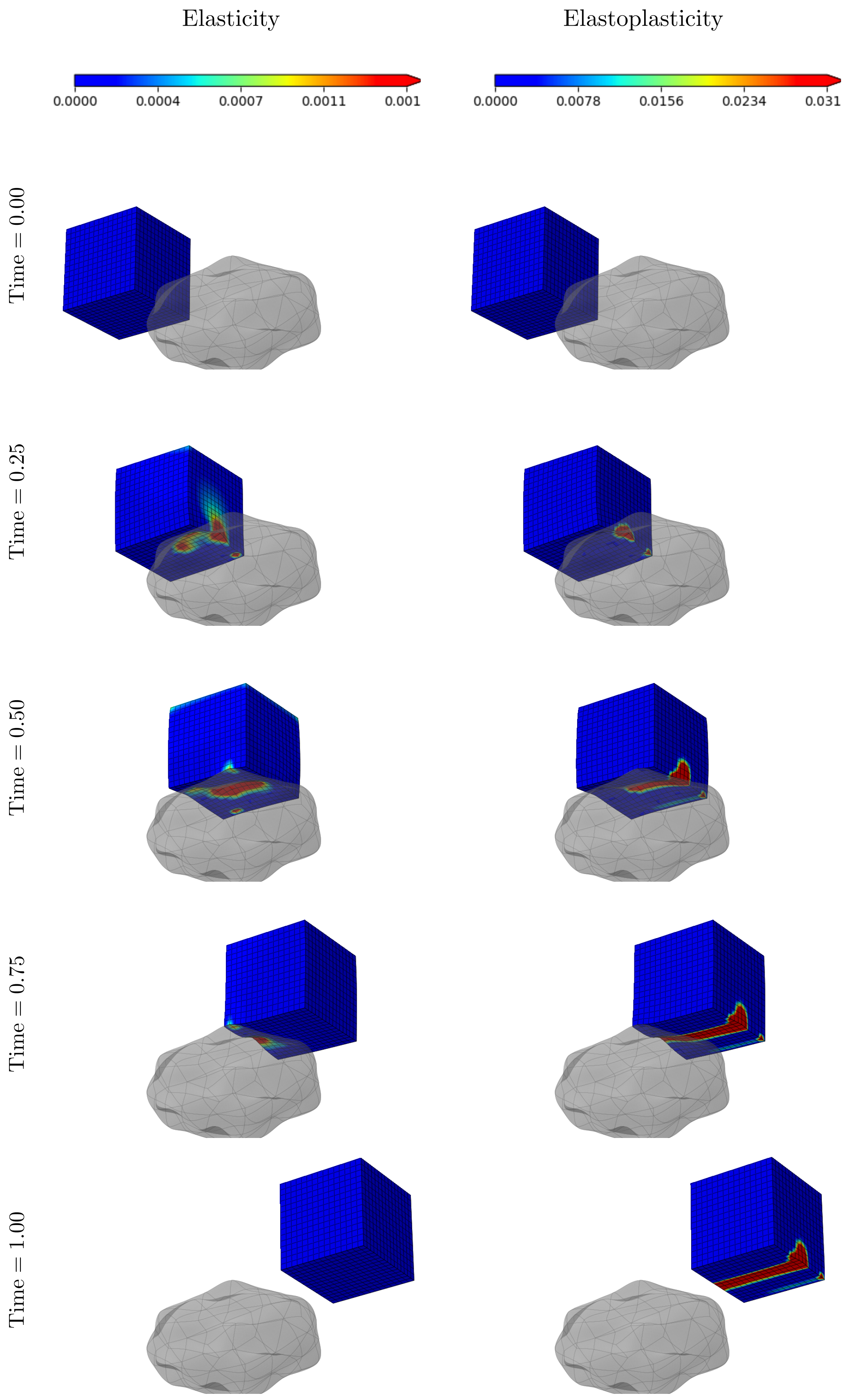


Figure 3.26: 3D simulation type III, $15 \times 15 \times 15$ FEs. Some deformations predicted using Newton's method with ANN-driven contact detection.

Table 3.3: Performance of simulations that require minimization solvers. Times (in seconds) taken for different parts of the simulation. The last three columns are counters and they include iterations of accepted and rejected increments.

type	deformable	method	Total	K+solv	K+MINsolv	f_{mat}	CDA+f_{con}	Newt iter	Min iter	Min force eval
II	10×10 × 10 elastic	BFGS	6824.40	1126.94	254.33	549.38	4128.92	2024	556	2898
		BFGS-ANN	4729.26	1148.66	253.31	541.62	2573.19	2059	567	2860
		TR	7004.45	1126.80	2876.51	344.48	1913.87	2024	735	736
		TR-ANN	4333.68	1153.79	1804.54	317.60	867.62	2059	463	464
	10×10 × 10 plastic	TR	5934.61	1021.14	1769.31	632.14	1868.62	2017	260	261
		TR-ANN	9977.79	1879.82	4512.81	1029.35	2323.00	3310	688	712

solver in the contact simulations of this thesis. If an iterative solver would be used instead, the time to solve the systems would be drastically reduced. Consequently, the acceleration of the hybrid CDA would be substantially more noticable in the total simulation times.

Next, the same simulation setup is retained, but the elastoplastic constitutive law is employed. The force-time curve in the bottom diagram of Fig. 3.25 shows minor differences with that of the elastic simulation (top diagram of Fig. 3.25). A look at the right column of Fig. 3.26 confirms that some plastic deformation develops, but the plastic multiplier does not exceed 0.031 during the course of the simulation. Consequently, the times for this simulation reported in the seventh and eight rows of Table 3.2 (type III, $15 \times 15 \times 15$, plastic) are similar to those for the elastic simulation.

The next simulation setup goes back to type II, considers hyperelasticity and a discretization of $10 \times 10 \times 10$ FEs ($n_{\text{var}} = 3630$). This simulation setup is interesting because the simulation switches the solver from Newton's method to a true minimization solver for one time increment. This can be observed in the predicted force that is presented over the course of the simulation in the top diagram of Fig. 3.27. The curve predicted by the conventional simulation matches the one predicted by the hybrid CDA simulation extremely well.

Next, the same simulation setup is used, but elastoplasticity instead of elasticity is considered. The force-time curves in the bottom diagram of Fig. 3.27 again match very well and show that the Newton solver is twice replaced for a minimization solver. The predictions of the quasi-Newton methods are not reported because they did not converge - neither the hybrid CDA simulations nor the conventional one.

The performance reported in the two bottom rows of Table 3.2 shows that the simulation with the hybrid CDA took substantially longer than the conventional one. Apparently, the mistakes of the hybrid CDA yielded a substantial increase in the number of TR iterations and Newton iterations, yet the converged results did not change.

Lastly, the reason for the lack of simulations of type I is explained. The simulations of type I require more frequently that the Newton solver is exchanged for a minimization solver (see e.g. Fig. 3.28). Furthermore, the minimization problems that the solvers must tackle are also more complex - as derived from the magnitude of the discrete force jumps. Thus, the minimization solvers seem to be more affected by the mistakes introduced by the hybrid CDA than Newton's method. On the other hand, this cannot be stated with complete certainty because the contact constraints are applied in a bilateral fashion when Newton's method is deployed, whereas they are applied in a unilateral fashion if a minimization

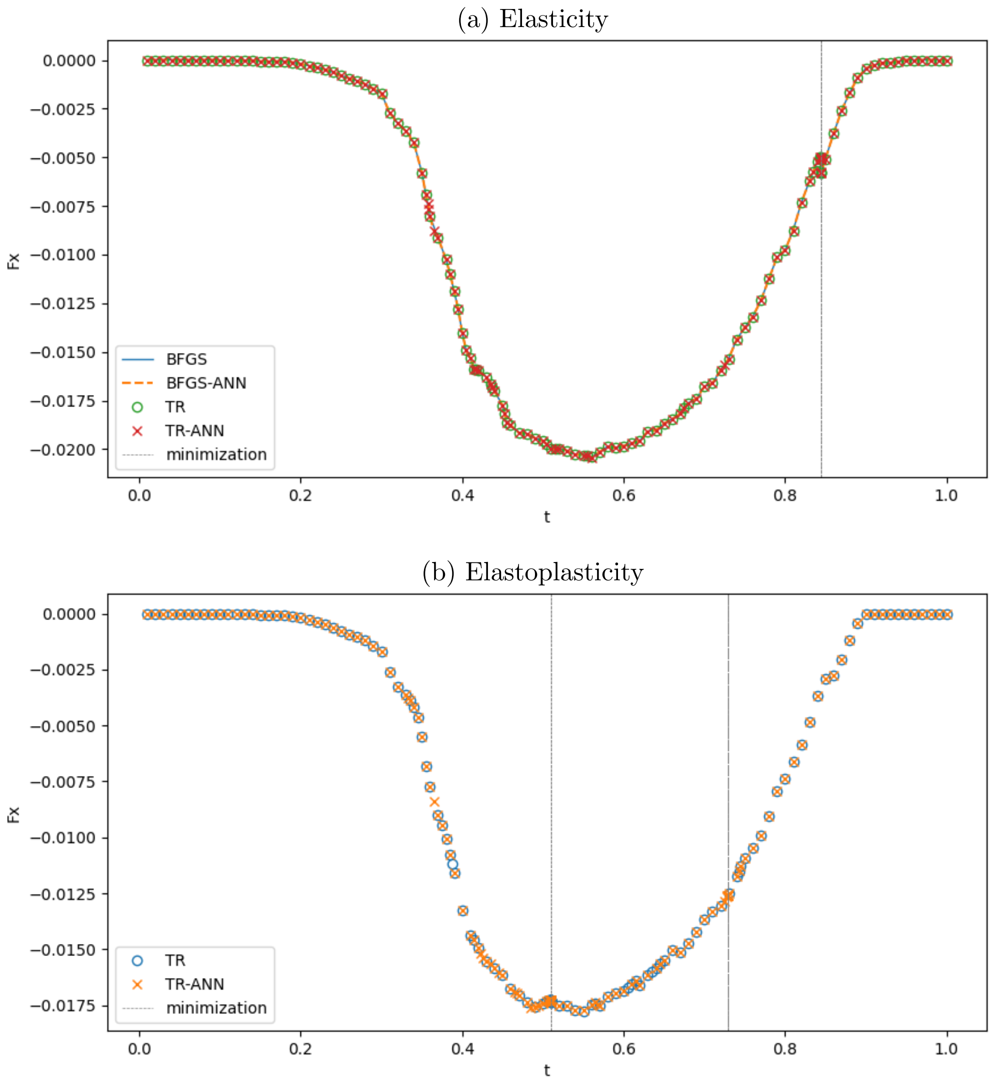


Figure 3.27: Simulation type II, 10×10×10 FEs, requiring minimization solvers. The forces in x-direction predicted by the conventional simulations and the hybrid CDA simulations as a function of the pseudo-time.

solver is used.

3.7 Summary

The results presented in this chapter focus on evaluating two neural network-based approaches for accelerating the contact-specific computation processes in deformable-to-rigid unilateral contact simulations. First, the Neural-Pull method was tested, achieving high levels of accuracy in predicting the signed distance function; however, the accuracy proved insufficient for contact mechanics simulations, particularly for its derivatives. The second approach, a multi-task neural network, showed promising performance for its three tasks: signed distance prediction, patch classification, and surface projection. The signed distance task effectively filtered potentially active nodes with a modest sub-architecture, achieving an error of approximately 10^{-2} and enabling small broad-phase thresholds to mantain a

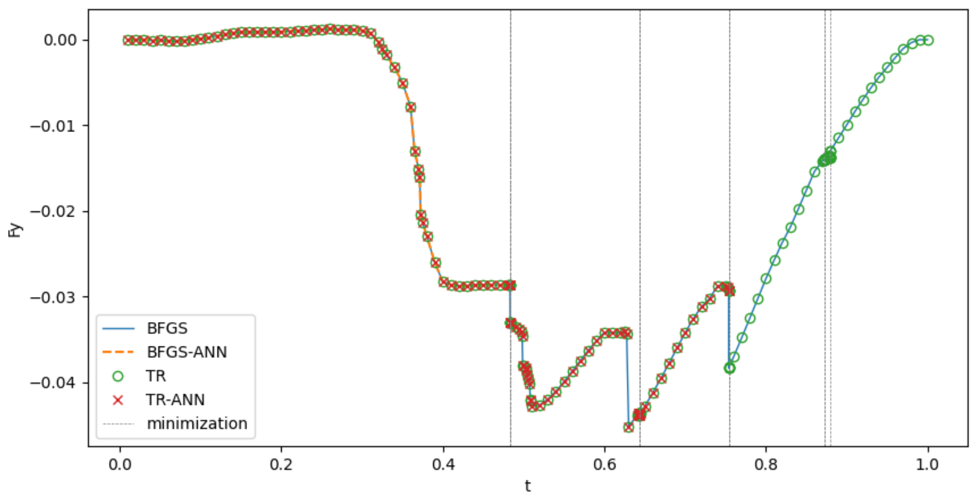


Figure 3.28: Simulation type I, $5 \times 5 \times 5$ FEs, requiring minimization solvers. The forces in x-direction predicted by the conventional simulations and the hybrid CDA simulations as a function of the pseudo-time.

conservative filter of possibly active nodes. The patch classification task reached an overall accuracy of 98.7%, with misclassifications mainly concentrated among neighbouring patches, which appears to be a reasonable outcome. Similarly, the surface projection task demonstrated uniformly low errors across coordinates, with typical values on the order of 10^{-2}.

To assess the precision of the MTNN as a whole, the signed distance and its derivatives have been computed directly from the predictions of $\underline{\underline{\hat{\xi}}}$ and the highest probabilities of $\hat{p}$. In this scenario, a larger dataset was tested and the predicted values for the signed distance and its derivatives have attained very high precision for the vast majority of the points tested. Furthermore, the simulations performed have shown results with high accuracy in most of the cases and an improvement in the computation times. The times employed to compute the active set show a clear tendency to reduce, as well as the times of the contact-specific parts. In some cases, the overall contact-specific times did not improve, but in those cases, this effect was due to having more iterations required. This can be linked to accuracy.

Chapter 4

Conclusions

Deformable bodies in unilateral contact simulations frequently undergo snap-through and snap-back phenomena. Whereas [31] has shown that optimization solvers are capable to treat snap-through phenomena in contact mechanics, Chapter 2 has investigated whether optimization solvers are also able to treat snap-back phenomena that may occur in unilateral contact simulations.

The test problems focused on frictionless contact between a deformable follower-body and a rigid leader-body. A node-to-segment contact framework was used in combination with Gregory patches to smoothen the rigid leader-surface. The investigation involved four different minimization algorithms, hyperelasticity and hyperelastoplasticity, 2D simulations with bilinear quadrilateral finite elements, 3D simulations with trilinear hexahedral finite elements and different element sizes. The conclusions can be summarized as follows:

- Several times that the simulations switched from Newton's method to an optimization solver were caused by the specifities of the node-to-segment contact framework used. Hence, minimization algorithms seem to improve the robustness of node-to-segment frameworks.

- In other instances, the simulations switched solvers because a snap-back occurred that was physically desired and unrelated to the employed contact framework. Hence, optimization algorithms may also be interesting for mortar frameworks.

- In general, the snap-backs occur less frequently in the simulations where plasticity has been considered, which is expected since the elastic energy stored that could potentially be released is less than in the elastic case.

- It is reasonable to predict that if friction were to be considered, the snap-back effect would not be as abrupt as it has been observed for the frictionless examples studied in this thesis. Furthermore, fewer snap-backs would be expected to happen in every case for frictional cases since, similarly to the effects observed in the plastic cases, the elastic recovery would be restricted.

- Minimization algorithms are able to treat hyperelastoplasticity in displacement-driven contact simulations.

- The trust region (TR) methods outperform the quasi-Newton methods in terms of the number of force evaluations. Of the two TR methods, the preconditioned one requires substantially fewer force evaluations (and conjugate gradient iterations) than the original, unpreconditioned one. The price to pay for this improvement is the construction of the preconditioner.

- The TR methods predict non-physical deformations more often than the quasi-Newton methods. This may be explained by the fact that the line search of the quasi-Newton method guarantees that no energy barriers are crossed during the minimization.

- Because the objective function is quickly polluted by machine precision for an increase in the number of FEs, the use of the objective function is circumvented by replacing it by the gradient (i.e. the force column)as suggested by [31]. However, this remedy only works to some extent. Hence, for true engineering problems, other treatments must be exploited or formulated. Domain decomposition methods and/or multigrid methods may form potential remedies.

All minimization algorithms may predict non-physical deformations. This may be avoided by (combinations of) several approaches:

- The node-to-segment method can be replaced with a mortar method at the price of increased simulation times. This would also reduce the number of times that the optimization solver is deployed.

- Incorporating self-contact may improve the quasi-Newton predictions. It is questionable whether it will also improve those of the TR methods.

- The accuracy of the TR subproblem's minimizer is limited once the TR boundary is encountered. The reason is that the TR methods of Chapter 2 stop searching as soon as the TR boundary is encountered, since the location where the search direction crosses the TR boundary is taken as the minimizer of the TR subproblem. More advanced methods that continue to explore the TR boundary as soon as the boundary is encountered (e.g. [47, 109]) may prevent the erroneous deformations predicted by the TR methods of Chapter 2.

- Changing or adjusting the constitutive model so that the stored energy increases more substantially for moderate deformations may avoid the prediction of non-physically exotic deformations.

- Domain decomposition methods or multigrid methods may reduce the chance that non-physical deformations occur.

- Instead of linear quads and hexahedrons, linear triangles and tets may be used. Triangular and tetrahedral discretizations increase the 'stiffness' of deformable bodies compared to the use of quads and hexahedrons and may, therefore, reduce the chance that non-physical deformations occur. However, limiting the approach to a particular type of finite element is not a truly satisfying remedy.

The direct prediction of the contact quantities with Neural-pull does not provide enough accuracy to be used for the integration of their contributions into the mechanical system.

The neural network model was tested for the detection and solving of contact simulations with the reference body as an obstacle for a given deformable body. It fails to approximate a good guess for Newton iterations, resulting in no convergence of the simulations at the starting point of the contact between the deformable body and the given obstacle. This is coherent and indeed expected with respect to the order of the errors computed from the approximated fields. The performance of the model can be considered accurate in the context of computer vision and robotics, but in the case of computational mechanics, it fails in terms of applicability as the algorithms need a higher order of accuracy for computing reaction forces and modified material stiffness at the contact points in the context of penalty-based contact simulations. In conclusion, current methods in artificial intelligence are not sufficient to assist the aforementioned mechanical simulations, providing evidence of the need for more powerful architectures to consider direct AI accelerators for simulations in contact mechanics.

On the other hand, a multi-task neural network model has been proposed, which enables the replacement of parts of contact detection in conditions such as smooth, irregular 3D rigid surfaces in contact with deformable solids. The model has been trained, achieving similar accuracy when compared to solving the simulations with more traditional methods of contact detection and, in all the observed cases, performing the contact-specific computations in less time per iteration. When the active set strategy for contact nodes is considered, the contact detection carried out at the end of each increment results in a significant improvement.

For finer meshes, the computation time specifically related to contact detection shows significant acceleration. However, while the overall simulation time improves, it does not exhibit the same level of speed-up. This is likely because a simple linear solver was used for the Newton iterations. Employing an iterative solver is expected to better reflect the acceleration achieved in contact computations within the total simulation time.

Within the MTL model, the subparts perform their tasks, and together they are capable of predicting the correct p closest patch and the respective (ξ_1^p,ξ_2^p) surface projection values that, along with the position of the contact node, allow accurate values for g and its derivatives, failing to find the correct contact quantities in fewer than 3 of every 1000 cases for points in the range inspected. Despite this level of accuracy, the completion of a simulation cannot be guaranteed, especially for cases that cannot be solved using Newton's method and instead rely on other methods, such as minimization, which require many more iterations and, therefore, more contact evaluations.

The segmented regression sub-architecture of the projection subtask achieves its objective, avoiding error concentrations at the edges for the patches considered, showing that selective back-propagation can be employed in neural networks like the one used in this thesis, where multiple regressions of different specimens of the same class (in this case, 96 different Gregory patches) need to be approximated. At the same time, the errors introduced by these approximations are small enough to use their estimations as an initial guess to find the exact (ξ_1,ξ_2) projection using only Newton for this purpose, completely avoiding the need to recur to other iterative methods such as the recursive seeding method presented.

Appendix A

Sensitivity to penalty value and contact constraint variants

This appendix presents a brief study of how the stiffness parameter k affects the responses obtained for the 2d simulations described in section 2.5 for a mesh of 5×5. It also describes how the bilateral contact variant performs, as well as the variant with the adaptive penalty stiffness (see section 2.2.3). Each simulation is initially discretized in time using 100 increments that solved using Newton's method. If Newton's method fails to converge after 15 iterations, a cut-back of the time increment is performed and attempted again using Newton's method. If more than 20 cut-backs for the same time increment are attempted (in which case the initial time increment is reduced by a factor of 2^{20}), the simulation is stopped.

A.1 Penalty stiffness for unilateral constraints

First, a series of simulations are performed for the 2D case mentioned using the values $k \in \{10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}\}$. Higher values of the penalty parameter k guarantee a more realistic situation of unilateral contact in the sense that the penetrations of the follower nodes in the leader surface are reduced. Figure A.1 shows the number of cut-backs needed in each time increment as Newton failed to converge. It can be observed that the simulations using $k = 10^{-2}$ and $k = 10^{-1}$ reach the furthest and all others stop at the same time. However, the penetrations observed in these two cases cases are unphysical and, furthermore, their horizontal force-displacement curves deviate from the forces of the more realistic cases (those with higher penalty stiffness k).

A.2 Adaptive penalty stiffness for unilateral constraints

The contact variant with the adaptive penalty stiffness of Eq. (2.97) is studied next for different values of $g_{\max}$. In each case, the minimum penetration value is taken as $g_{\min} = 1/2\, g_{\max}$. The initial stiffness for each follower node is $k_0 = 1.0$, and the stiffness correction value κ is set to 0.1.

Figure A.2 shows the results of three simulations with values of $g_{\max}$ of 10^{-5}, 10^{-4} and 10^{-3}. The three cases reach a similar simulation time with similar horizontal reaction

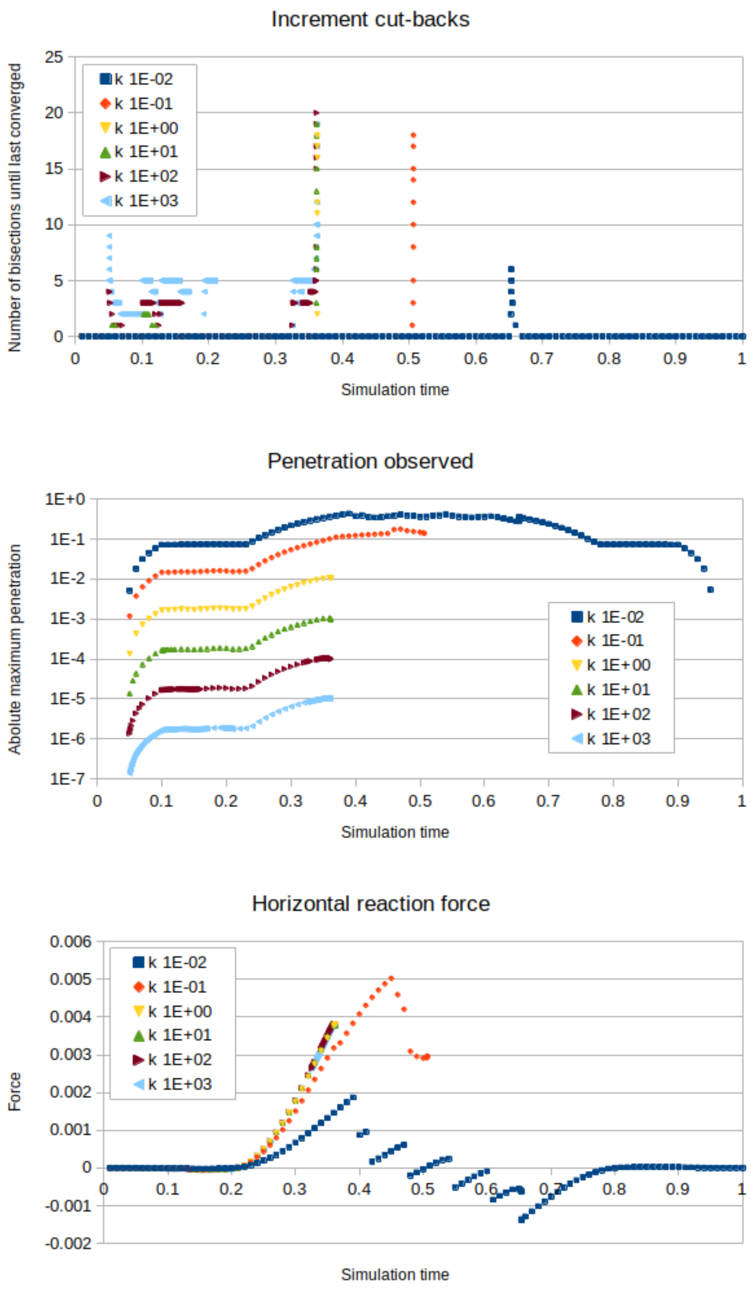


Figure A.1: Responses for the unilaterally-constrained simulation for different values of contact penalty k.

forces. This time, the number of cut-backs needed when Newton's method does not converge has decreased. However, some increment repetitions are required in order to adjust the stiffness parameters when the gap limits are violated after converged increments. From the *Contact stiffness* graph, it can be seen that a stiffness values of $k = 10$ result in maximum penetrations of 10^{-3} throughout the simulations, which is considered acceptable for this study, as it provides force responses similar to those of stricter maximum penetration values.

A.3 Bilateral constraints (Active set method)

Finally, the last method described in 2.2.3 is studied for penalty values of $k \in \{10^2, 10^3, 10^4, 10^5, 10^6\}$. The cut-backs needed are only visible at the most critical part of the simulations (see Figure A.3). This shows that there are no longer in-and-out behaviours from nodes during the iterations of Newton's method. The cost of adopting this is reflected in having to repeat increments whenever it is necessary to update the set of nodes in contact over which the bilateral contact constraints are applied. These repetitions, however, are much less frequent than the cut-backs in the unilaterally-constrained versions of the same simulation.

After these quick sensitivity studies, it has been decided to adopt the bilateral constraint to address the increments that use Newton's method. For the 2D case presented here, the maximum number of permitted Newton iterations is 15, after which a cut-back is implemented to attempt solving a smaller increment with Newton's method. If more than 10 consecutive cut-backs occur, it is assumed that Newton's method is insufficient to resolve the configuration at the current simulation time; therefore, alternative solution methods must be considered.

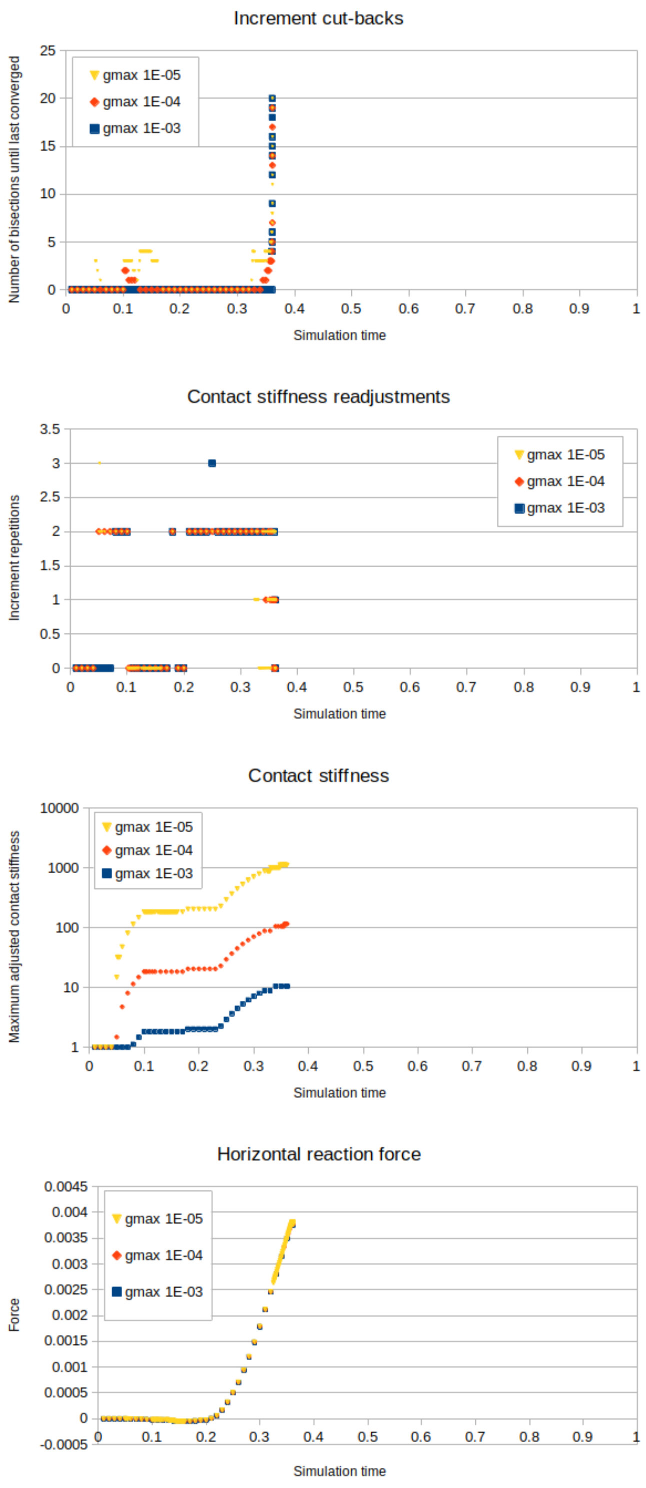


Figure A.2: Unilaterally-constrained cases with adaptive contact penalty stiffness for different values of maximum penetration allowed $g_{\max}$.

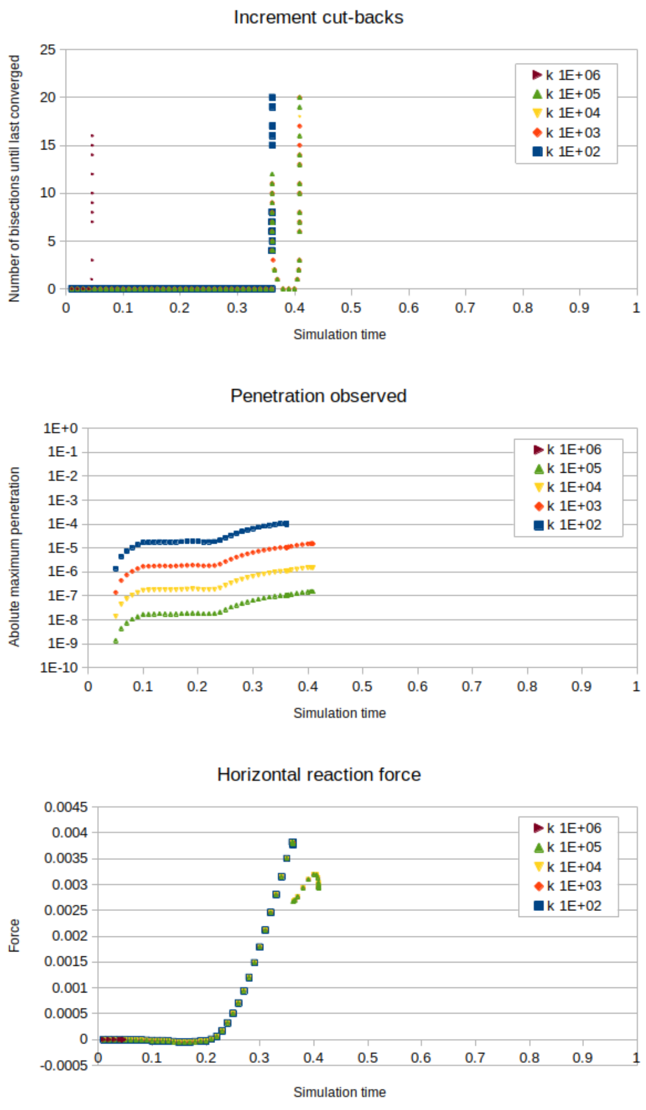


Figure A.3: The bilaterally-constrained case (Active set method). Responses for different contact penalty values k.

Appendix B

Recursive Seeding Strategy

The algorithm of recursive seeding employed in this thesis can be summarized as follows:

Algorithm 11 Recursive Seeding for Initial Guess ξ_0

```
1: Input: Follower node $\mathbf{x}$, initial parametric domain bounds $[\xi_1^0, \xi_1^1]$, $[\xi_2^0, \xi_2^1]$, seeding
   level $n_{\text{seed}}$, recursion depth $n_{\text{rec}}$, maximum recursion depth $n_{\text{rec}}^{\max}$.
2: Output: Refined initial guess $\hat{\underline{\xi}} = (\hat{\xi}_1, \hat{\xi}_2)$.
3: Initialize $\hat{\xi}_1^{\min}, \hat{\xi}_2^{\min} \leftarrow \underline{0}$.
4: $d_{\min} \leftarrow \|\mathbf{x} - \mathbf{Grg}(0,0)\|$.
5: Define step size $\Delta\xi_1, \Delta\xi_2 \leftarrow (\xi_1^1 - \xi_1^0)/n_{\text{seed}}, (\xi_2^1 - \xi_2^0)/n_{\text{seed}}$.
6: for $\xi_1 \in [\xi_1^0, \xi_1^1]$ with step size $\Delta\xi_1$ do
7:     for $\xi_2 \in [\xi_2^0, \xi_2^1]$ with step size $\Delta\xi_2$ do
8:         Compute $\mathbf{X}_{\text{lead}} = \mathbf{Grg}(\xi_1, \xi_2)$.
9:         $d = \|\mathbf{x} - \mathbf{x}_{\text{lead}}\|$.
10:        if $d < d_{\min}$ then
11:            $d_{\min}, \hat{\xi}_1^{\min}, \hat{\xi}_2^{\min} \leftarrow d, \xi_1, \xi_2$.
12:        end if
13:    end for
14: end for
15: if $n_{\text{rec}} < n_{\text{rec}}^{\max}$ then
16:     Refine bounds:
            $\xi_1^0, \xi_1^1 \leftarrow \max(0, \hat{\xi}_1^{\min} - \Delta\xi_1), \min(1, \hat{\xi}_1^{\min} + \Delta\xi_1)$
            $\xi_2^0, \xi_2^1 \leftarrow \max(0, \hat{\xi}_2^{\min} - \Delta\xi_2), \min(1, \hat{\xi}_2^{\min} + \Delta\xi_2)$
17:     Return RecursiveSeeding($\mathbf{x}, [\xi_1^0, \xi_1^1], [\xi_2^0, \xi_2^1], n_{\text{seed}}, n_{\text{rec}} + 1, n_{\text{rec}}^{\max}$).
18: end if
19: Return $\hat{\xi}_1^{\min}, \hat{\xi}_2^{\min}$.
```

The recursive seeding strategy addresses convergence issues when finding, for a specific point in space point, the closest projection point on a surface by systematically refining the initial guess through a hierarchical search. This is depicted in Fig.3.3 for the two first search iterations. By dividing the parametric space into progressively smaller regions and evaluating candidate points, the method ensures that Newton's method begins close enough to the global minimum for reliable convergence. However, this approach is computationally expensive and becomes infeasible for large-scale simulations. This motivates the need for alternative approaches, such as using artificial neural networks (ANNs), as proposed in the next chapter, to provide sufficiently accurate initial guesses while avoiding the cost of recursive refinement.

Limitations of Recursive Seeding. Although recursive seeding provides a robust mechanism for improving the reliability of Newton's method, it is computationally intensive. Each level of recursion involves evaluating multiple candidate points across the parametric space, leading to significant overhead, particularly for large-scale simulations with complex geometries. This motivates the exploration of alternative approaches, such as leveraging ANNs to predict initial guesses, which can bypass the need for recursive refinement altogether.

Appendix C

Loss function for Segmented regression: Implementation Example

The implementation of the custom loss function in TensorFlow Keras is presented in Listing C.1. In this numerical implementation, some modifications have to be made from the one shown in Eq.3.40. That is, an additional term had to be introduced for the training set of the task and consequently, it is also introduced as output. The reason for this is that we want to access the true closest patch p in order to calculate the loss as in Eq.3.40 and, for the package to do this, the output layers must have an output corresponding with this label, however, the value of this output is neither considered during training nor for the predictions.

```python
...
projection_extra_true = np.zeros((n_data,2*96+1))    # the '+1' is to store the
    patch number to be used in the custom loss function

for i in range(n_data):
    p_i = int(y_classification[i])
    projection_extra_true[i,[p_i,p_i+96]] = projection_true[i]

projection_extra_true[:,2*96] = classification_true.ravel()     # <-- adds true
    patch label as last projection label

def custom_loss(projection_extra_true, projection_extra_pred):
    # Extract patch/class labels and convert to one-hot encoding
    p_idx = tf.cast(projection_extra_true[:, -1], tf.int32)
    y_class = tf.one_hot(p_idx, depth=96)

    # Extract regression targets for xi1, xi2
    xi1_true = projection_extra_true[:, :96]
    xi2_true = projection_extra_true[:, 96:192]
    xi1_pred = projection_extra_pred[:, :96]
    xi2_pred = projection_extra_pred[:, 96:192]

    # Compute weighted sum of squared errors using the true patch label
    xi1_loss = tf.reduce_sum(y_class * tf.square(xi1_true - xi1_pred), axis=1)
    xi2_loss = tf.reduce_sum(y_class * tf.square(xi2_true - xi2_pred), axis=1)

    # Compute mean loss across all samples
    total_loss = tf.reduce_mean(xi1_loss + xi2_loss)
    return total_loss
```

Listing C.1: Customized Loss Function for Projection Coordinates. Example in Python code. In this example for 96 patches, the first 96 output terms correspond to $\hat{\xi}_1^p$, the next 96 terms are $\hat{\xi}_2^p$ ($p \in \{1, ..., 96\}$) and the last added term is p, the true patch for the sample.

Appendix D

Bibliography

[1] L. Chen, M. Magliulo, M. Elig, and L. Beex, “A mechanical model for compaction of strands for wire ropes,” *International Journal of Solids and Structures*, vol. 269, p. 112178, 2023.

[2] L. Beex and R. Peerlings, “On the influence of delamination on laminated paperboard creasing and folding,” *Philosophical Transactions of the Royal Society A, Mathematical, Physical and Engineering Sciences*, vol. 370, pp. 1912–1924, 2012.

[3] N. Noraphaiphipaksa, A. Manonukul, A. Kanchanomai, and Y. Mutoh, “Fretting-contact-induced crack opening/closure behaviour in fretting fatigue,” *International Journal of Fatigue*, vol. 88, pp. 185–196, 2016.

[4] R. Hojjati-Talemi, M. Abdel Wahab, J. de Pauw, and P. de Baets, “Prediction of fretting fatigue crack initiation and propagation lifetime for cylindrical contact configuration,” *Tribology International*, vol. 76, pp. 73–91, 2014.

[5] J. Hallquist, G. Goudreau, and D. Benson, “Sliding interfaces with contact-impact in large-scale Lagrangian computations,” *Computer Methods in Applied Mechanics and Engineering*, vol. 51, pp. 107–137, 1985.

[6] G. Zavarise and L. De Lorenzis, “The node-to-segment algorithm for 2D frictionless contact: Classical formulation and special cases,” *Computer Methods in Applied Mechanics and Engineering*, vol. 198, pp. 3428–3451, 2009.

[7] P. Wriggers and L. Krstulovič-Opara, “On smooth finite element discretizations for frictional contact problems,” *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 80, pp. 77–80, 2000.

[8] V. Padmanabhan and T. Laursen, “A framework for development of surface smoothing procedures in large deformation frictional contact analysis,” *Finite Elements in Analysis and Design*, vol. 37, pp. 173–198, 2001.

[9] T. Belytschko, W. Daniel, and G. Ventura, “A monolithic smoothing-gap algorithm for contact-impact based on the signed distance function,” *International Journal for*

Numerical Methods in Engineering, vol. 55, pp. 101–125, 2002.

[10] M. Stadler, G. Holzapfel, and J. Korelc, " C^n continuous modelling of smooth contact surfaces using NURBS and application to 2D problems," *International Journal for Numerical Methods in Engineering*, vol. 57, pp. 2177–2203, 2003.

[11] D. Chamoret, P. Saillard, A. Rassineux, and J.-M. Bergheau, "New smoothing procedures in contact mechanics," *Journal of Computational and Applied Mathematics*, vol. 168, pp. 107–116, 2004.

[12] P. Litewka, "Hermite polynomial smoothing in beam-to-beam frictional contact," *Computational Mechanics*, vol. 40, pp. 815–826, 2007.

[13] D. Neto, M. Oliveira, and L. Menezes, "Surface smoothing procedures in computational contact mechanics," *Archives of Computational Methods in Engineering*, vol. 24, pp. 37–87, 2017.

[14] M. Magliulo, A. Zilian, and L. Beex, "Contact between shear-deformable beams with elliptical cross sections," *Acta Mechanica*, vol. 231, pp. 273–291, 2020.

[15] M. Magliulo, J. Lengiewicz, A. Zilian, and L. Beex, "Non-localised contact between beams withcircular and elliptical cross-sections," *Computational Mechanics*, vol. 65, pp. 1247–1266, 2020.

[16] M. Magliulo, J. Lengiewicz, A. Zilian, and L. Beex, "Beam-inside-beam contact: mechanical simulations of slender medical instruments inside the human body," *Computer Methods and Programs in Biomedicine*, vol. 196, p. 105527, 2020.

[17] M. Magliulo, J. Lengiewicz, A. Zilian, and L. Beex, "Frictional interactions for non-localized beam-to-beam and beam-inside-beam contact," *International Journal for Numerical Methods in Engineering*, vol. 122, pp. 1706–1731, 2021.

[18] C. Faccio J., A. Gay N., and P. Wriggers, "Spline-based smooth beam-to-beam contact model," *Computational Mechanics*, vol. 72, pp. 663–692, 2023.

[19] G. Zavarise and P. Wriggers, "A segment-to-segment contact strategy," *Mathematical and Computer Modelling*, vol. 28, pp. 497–515, 1998.

[20] T. McDevitt and T. Laursen, "A mortar-finite element formulation for frictional contact problems," *International Journal for Numerical Methods in Engineering*, vol. 48, pp. 1525–1547, 2000.

[21] M. Puso and T. Laursen, "A mortar segment-to-segment contact method for large deformation solid mechanics," *Computer Methods in Applied Mechanics and Engineering*, vol. 193, pp. 601–629, 2004.

[22] M. Puso and T. Laursen, "A mortar segment-to-segment frictional contact method for large deformations," *Computer Methods in Applied Mechanics and Engineering*, vol. 193, pp. 4891–4913, 2004.

[23] T. Duong, L. De Lorenzis, and R. Sauer, "A segmentation-free isogeometric extended mortar contact method," *Computational Mechanics*, vol. 63, pp. 383–407, 2019.

[24] M. Puso and J. Solberg, "A dual pass mortar approach for unbiased constraints and self-contact," *Computer Methods in Applied Mechanics and Engineering*, vol. 367, p. 113092, 2020.

[25] M. Faraji, A. Seitz, C. Meier, and W. Wall, "A mortar finite element formulation for large deformation lubricated contact problems with smooth transition between mixed, elasto-hydrodynamic and full hydrodynamic lubrication," *Tribology Letters*, vol. 71, p. 11, 2023.

[26] M. Crisfield, "An arc-length method including line searches and accelerations," *International Journal for Numerical Methods in Engineering*, vol. 19, pp. 1269–1289, 1983.

[27] B. Forde and S. Stiemer, "Improved arc length orthogonality methods for nonlinear finite element analysis," *Computers and Structures*, vol. 27, pp. 625–630, 1987.

[28] M. Geers, "Enhanced solution control for physically and geometrically non-linear problems. part ii - comparative performance analysis," *International Journal for Numerical Methods in Engineering*, vol. 46, pp. 205–230, 1999.

[29] B. van Hal, R. Peerlings, M. Geers, and O. van der Sluis, "Cohesive zone modeling for structural integrity analysis of ic interconnects," *Microelectronics Reliability*, vol. 47, pp. 1251–1261, 2007.

[30] R. Silveira, W. Pereira, and P. Gonçalves, "Nonlinear analysis of structural elements under unilateral contact constraints by a ritz type approach," *International Journal of Solids and Structures*, vol. 45, pp. 2629–2650, 2008.

[31] M. Tupek and B. Talamini, "Optimization-based algorithms for nonlinear mechanics and frictional contact," Tech. Rep. SAND2021-11497; 699443, Sandia National Laboratories, 2021.

[32] H. Houssein, S. Garnotel, and F. Hecht, "Regularized frictional contact problems with the interior point method," *Japan Journal of Industrial and Applied Mathematics*, vol. 40, pp. 775–807, 2023.

[33] V. Acary, F. Cadoux, C. Lemaréchal, and J. Malick, "A formulation of the linear discrete coulomb friction problem via convex optimization," *ZAMM Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 91, pp. 155–175, 2011.

[34] V. Acary, P. Armand, H. Minh Nguyen, and M. Shpakovych, "Second-order cone programming for frictional contact mechanics using interior point algorithm," *Optimization Methods and Software*, vol. 39, pp. 634–663, 2024.

[35] C. Broyden, "The convergence of a class of double-rank minimization algorithms 1. General considerations," *IMA Journal of Applied Mathematics*, vol. 6, pp. 76–90, 1970.

[36] R. Fletcher, "A new approach to variable metric algorithms," *The Computer Journal*, vol. 13, pp. 317–322, 1970.

[37] D. Goldfarb, “A family of variable-metric methods derived by variational means,” *Mathematics of Computation*, vol. 24, pp. 23–26, 1970.

[38] D. Shanno, “Conditioning of quasi-Newton methods for function minimization,” *Mathematics of Computation*, vol. 24, pp. 647–656, 1970.

[39] J. Nocedal and S. Wright, *Numerical optimization.* Springer Series in Operations Research and Financial Engineering, New York, NY: Springer Science+Business Media, LLC, 2006.

[40] J. Nocedal, “Updating quasi-newton matrices with limited storage,” *Mathematics of Computation*, vol. 35, pp. 773–802, 1980.

[41] D. Liu and J. Nocedal, “On the limited memory BFGS method for large scale optimization,” *Mathematical Programming*, vol. 45, pp. 503–528, 1989.

[42] S. Goldfeld, R. Quandt, and H. Trotter, “Maximization by quadratic hill-climbing,” *Econometrica*, vol. 34, pp. 541–551, 1966.

[43] D. Sorensen, “Newton’s method with a model trust region modification,” *SIAM Journal on Numerical Analysis*, vol. 19, pp. 409–426, 1982.

[44] R. Fletcher, *Practical methods of optimization.* John Wiley Sons, Ltd, 2000.

[45] P. Toint, *Towards an efficient sparsity exploiting Newton method for minimization*, pp. 57–88. London: Academic Press, 1981.

[46] T. Steihaug, “The conjugate gradient method and trust regions in large scale optimization,” *SIAM Journal on Numerical Analysis*, vol. 3, pp. 626–637, 1983.

[47] J. Erway, P. Gill, and J. Griffin, “Iterative methods for finding a trust-region step,” *SIAM Journal on Optimization*, vol. 20, pp. 1110–1131, 2007.

[48] R. Ogden, “Large deformation isotropic elasticity – on the correlation of theory and experiment for incompressible rubberlike solids,” *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, vol. 326, pp. 565–584, 1972.

[49] M. Latorre and F. Montáns, “What-you-prescribe-is-what-you-get orthotropic hyperelasticity,” *Computational Mechanics*, vol. 53, pp. 1279–1298, 2014.

[50] L. Beex, “Fusing the Seth–Hill strain tensors to fit compressible elastic material responses in the nonlinear regime,” *International Journal of Mechanical Sciences*, vol. 163, p. 105072, 2019.

[51] M. Ortiz and L. Stainier, “The variational formulation of viscoplastic constitutive updates,” *Computer Methods in Applied Mechanics and Engineering*, vol. 171, pp. 419–444, 1999.

[52] A. Mielke and T. Roubíček, *Rate-independent systems: theory and application*, vol. 193 of *Applied Mathematical Sciences.* New York, NY: Springer New York, 2015.

[53] O. Rokoš, L. Beex, J. Zeman, and R. Peerlings, "A variational formulation of dissipative quasicontinuum methods," *International Journal of Solids and Structures*, vol. 102-103, pp. 214–229, 2016.

[54] C. Carstensen, K. Hackl, and A. Mielke, "Non–convex potentials and microstructures in finite–strain plasticity," *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 458, pp. 299–317, 2002.

[55] B. Bourdin, G. Francfort, and J.-J. Marigo, "Numerical experiments in revisited brittle fracture," *Journal of the Mechanics and Physics of Solids*, vol. 48, pp. 797–826, 2000.

[56] G. Francfort and J.-J. Marigo, "Revisiting brittle fracture as an energy minimization problem," *Journal of the Mechanics and Physics of Solids*, vol. 46, pp. 1319–1342, 1998.

[57] L. Placidi, E. Barchiesi, and A. Misra, "A strain gradient variational approach to damage: a comparison with damage gradient models and numerical results," *Mathematics and Mechanics of Complex Systems*, vol. 6, pp. 77–100, 2018.

[58] B. Bourdin, G. Francfort, and J.-J. Marigo, "The variational approach to fracture," *Journal of Elasticity*, vol. 91, pp. 5–148, 2008.

[59] K. Pham and J.-J. Marigo, "Stability of homogeneous states with gradient damage models: size effects and shape effects in the three-dimensional setting," *Journal of Elasticity*, vol. 110, pp. 63–93, 2013.

[60] M. Puso and T. Laursen, "A 3D contact smoothing method using Gregory patches," *International Journal for Numerical Methods in Engineering*, vol. 54, pp. 1161–1194, 2002.

[61] I. Rocha, P. Kerfriden, and F. van der Meer, "Machine learning of evolving physics-based material models for multiscale solid mechanics," *Mechanics of Materials*, vol. 184, p. 104707, 2023.

[62] M. Maia, I. Rocha, P. Kerfriden, and F. van der Meer, "Physically recurrent neural networks for path-dependent heterogeneous materials: embedding constitutive models in a data-driven surrogate," *Computer Methods in Applied Mechanics and Engineering*, vol. 407, p. 115934, 2023. arXiv:2209.07320 [math].

[63] L. Wu, V. Nguyen, N. G. Kilingar, and L. Noels, "A recurrent neural network-accelerated multi-scale model for elasto-plastic heterogeneous materials subjected to random cyclic and non-proportional loading paths," *Computer Methods in Applied Mechanics and Engineering*, vol. 369, p. 113234, 2020.

[64] S. Vijayaraghavan, L. Wu, L. Noels, S. Bordas, S. Natarajan, and L. Beex, "A data-driven reduced-order surrogate model for entire elastoplastic simulations applied to representative volume elements," *Scientific Reports*, vol. 13, p. 12781, 2023. Publisher: Nature Publishing Group.

[65] L. Wu and L. Noels, "Recurrent neural networks (RNNs) with dimensionality reduction and break down in computational mechanics; application to multi-scale local-

ization step," *Computer Methods in Applied Mechanics and Engineering*, vol. 390, p. 114476, 2022.

[66] F. Hendriks, V. Menkovski, M. Doškář, M. Geers, and O. Rokoš, "Similarity equivariant graph neural networks for homogenization of metamaterials," 2024. arXiv:2404.17365 [cond-mat].

[67] L. Aarts and P. van der Veer, "Neural network method for solving partial differential equations," *Neural Processing Letters*, vol. 14, pp. 261–271, 2001.

[68] Y. Khoo, J. Lu, and L. Ying, "Solving parametric PDE problems with artificial neural networks," *European Journal of Applied Mathematics*, vol. 32, pp. 421–435, 2021.

[69] J. Blechschmidt and O. Ernst, "Three ways to solve partial differential equations with neural networks – A review," 2021. arXiv:2102.11802 [math].

[70] S. Deshpande, S. Bordas, and J. Lengiewicz, "MAgNET: A graph U-Net architecture for mesh-based simulations," *Engineering Applications of Artificial Intelligence*, vol. 133, p. 108055, 2024.

[71] S. Deshpande, R. Sosa, S. Bordas, and J. Lengiewicz, "Convolution, aggregation and attention based deep neural networks for accelerating simulations in mechanics," *Frontiers in Materials*, vol. 10, 2023. Publisher: Frontiers.

[72] S. Deshpande, J. Lengiewicz, and S. Bordas, "Probabilistic deep learning for real-time large deformation simulations," *Computer Methods in Applied Mechanics and Engineering*, vol. 398, p. 115307, 2022.

[73] M. Raissi, P. Perdikaris, and G. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.

[74] P. Gkinis, E. Koronaki, A. Skouteris, I. Aviziotis, and A. Boudouvis, "Building a data-driven reduced order model of a chemical vapor deposition process from low-fidelity CFD simulations," *Chemical Engineering Science*, vol. 199, pp. 371–380, 2019.

[75] C. Martin-Linares, Y. Psarellis, G. Karapetsas, E. Koronaki, and I. Kevrekidis, "Physics-agnostic and physics-infused machine learning for thin films flows: modelling, and predictions from small data," *Journal of Fluid Mechanics*, vol. 975, p. A41, 2023.

[76] E. Koronaki, A. Nikas, and A. Boudouvis, "A data-driven reduced-order model of nonlinear processes based on diffusion maps and artificial neural networks," *Chemical Engineering Journal*, vol. 397, p. 125475, 2020.

[77] E. Koronaki, P. Gkinis, L. Beex, S. Bordas, C. Theodoropoulos, and A. Boudouvis, "Classification of states and model order reduction of large scale Chemical Vapor Deposition processes with solution multiplicity," *Computers & Chemical Engineering*, vol. 121, pp. 148–157, 2019.

[78] G. Loachamín-Suntaxi, P. Papavasileiou, E. Koronaki, D. Giovanis, G. Gakis, I. Aviziotis, M. Kathrein, G. Pozzetti, C. Czettl, S. Bordas, and A. Boudouvis, "Discovering deposition process regimes: Leveraging unsupervised learning for process insights, surrogate modeling, and sensitivity analysis," *Chemical Engineering Journal Advances*, vol. 20, p. 100667, 2024.

[79] T. Daniel, F. Casenave, N. Akkari, and D. Ryckelynck, "Model order reduction assisted by deep neural networks (ROM-net)," *Advanced Modeling and Simulation in Engineering Sciences*, vol. 7, p. 16, 2020.

[80] T. Sahin, M. von Danwitz, and A. Popp, "Solving forward and inverse problems of contact mechanics using physics-informed neural networks," *Advanced Modeling and Simulation in Engineering Sciences*, vol. 11, p. 11, 2024.

[81] C. Goodbrake, S. Motiwale, and M. Sacks, "A neural network finite element method for contact mechanics," *Computer Methods in Applied Mechanics and Engineering*, vol. 419, p. 116671, 2024.

[82] Z. Lai, Q. Chen, and L. Huang, "Machine-learning-enabled discrete element method: Contact detection and resolution of irregular-shaped particles," *International Journal for Numerical and Analytical Methods in Geomechanics*, vol. 46, pp. 113–140, 2022.

[83] S. Huang, P. Wang, Z. Lai, Z.-Y. Yin, L. Huang, and C. Xu, "Machine-learning-enabled discrete element method: The extension to three dimensions and computational issues," *Computer Methods in Applied Mechanics and Engineering*, vol. 432, p. 117445, 2024.

[84] J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "DeepSDF: Learning continuous signed distance functions for shape representation," 2019. arXiv:1901.05103.

[85] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, "Occupancy networks: learning 3D reconstruction in function space," 2019. arXiv:1812.03828.

[86] S. Saito, Z. Huang, R. Natsume, S. Morishima, A. Kanazawa, and H. Li, "PIFu: Pixel-aligned omplicit function for high-resolution clothed human digitization," 2019. arXiv:1905.05172.

[87] P. Erler, P. Guerrero, S. Ohrhallinger, M. Wimmer, and N. Mitra, "Points2Surf: Learning implicit surfaces from point cloud patches," 2020. arXiv:2007.10453.

[88] B. Mildenhall, P. Srinivasan, M. Tancik, J. Barron, R. Ramamoorthi, and R. Ng, "NeRF: Representing scenes as neural radiance fields for view synthesis," Aug. 2020. arXiv:2003.08934.

[89] B. Ma, Z. Han, Y.-S. Liu, and M. Zwicker, "Neural-pull: learning signed distance functions from point clouds by learning to pull space onto surfaces," 2021. arXiv:2011.13495.

[90] K. Hackl, A. Mielke, and D. Mittenhuber, "Dissipation distances in multiplicative elastoplasticity," *Lecture Notes in Applied and Computational Mechanics*, vol. 12,

pp. 87–100, 2003.

[91] A. Mielke, "Existence of minimizers in incremental elasto-plasticity with finite strains," *SIAM Journal on Mathematical Analysis*, vol. 36, pp. 384–404, 2005.

[92] K. Krabbenhøft, "A variational principle of elastoplasticity and its application to the modeling of frictional materials," *International Journal of Solids and Structures*, vol. 46, pp. 464–479, 2009.

[93] W. Davidson, "Variable metric method for minimization," Tech. Rep. ANL-5990, Argonne National Laboratory, 1959.

[94] W. Davidson, "Variable metric method for minimization," *SIAM Journal on Optimization*, vol. 1, pp. 1–17, 1991.

[95] H. Matthies and G. Strang, "The solution of nonlinear finite element equations," *International Journal for Numerical Methods in Engineering*, vol. 14, pp. 1613–1626, 1979.

[96] R. Fletcher and C. Reeves, "Function minimization by conjugate gradients," *The Computer Journal*, vol. 7, pp. 149–154, 1964.

[97] E. Polak and G. Ribière, "Note sur la convergence de méthodes de directions conjugués," *Revue Française d'Automatique, Informatique, Recherche Opérationnelle*, vol. 3, pp. 35–43, 1969.

[98] M. Hestenes and E. Stiefel, "Methods of conjugate gradients for solving linear systems," *Journal of Research of the National Bureau of Standards*, vol. 49, p. 2379, 1952.

[99] J. Gregory, "Smooth interpolation without twist constraints," in *Computer Aided Geometric Design*, pp. 71–87, Elsevier, 1974.

[100] L. Badea and L. Gilormini, "Application of a domain decomposition method to elastoplastic problems," *International Journal of Solids and Structures*, vol. 21, pp. 643–656, 1994.

[101] P. Kerfriden, O. Allix, and P. Gosselet, "A three-scale domain decomposition method for the 3d analysis of debonding in laminates," *Computational Mechanics*, vol. 44, pp. 343–362, 2009.

[102] A. Namazifard and I. Parsons, "A distributed memory parallel implementation of the multigrid method for solving three-dimensional implicit solid mechanics problems," *International Journal for Numerical Methods in Engineering*, vol. 61, pp. 1173–1208, 2004.

[103] T. Ekevid, P. Kettil, and N. Wiberg, "Adaptive multigrid for finite element computations in plasticity," *Computers and Structures*, vol. 82, pp. 2413–2424, 2004.

[104] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning.* Adaptive computation and machine learning, Cambridge, Mass: The MIT press, 2016.

[105] C. Bishop, *Pattern recognition and machine learning.* Information science and statistics, New York: Springer, 2006.

[106] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization,"

[107] M. Zeiler, "ADADELTA: An adaptive learning rate method," 2012. arXiv:1212.5701.

[108] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017. arXiv:1412.6980.

[109] J. Erway and P. Gill, "A subspace minimization method for the trust-region step," *SIAM Journal on Optimization*, vol. 20, pp. 1439–1461, 2010.