UNIVERSITÀ DI PARMA

Dottorato di Ricerca in Ingegneria Industriale

XXX Ciclo

An Interior-Point Method for Non-Smooth Multibody System Dynamics

Coordinatore:

Chiar.mo Prof. Gianni Royer Carfagni

Tutor:

Prof. Alessandro Tasora

Dottorando: *Dario Mangoni*

Anni 2014/2017

Mere cleverness is not wisdom
Euripides

Summary

List of Figures

List of Tables

Abstract

Simulations of non-smooth dynamic of rigid and flexible body systems, as involving contacts, require the solution of complementarity problems that can be effectively addressed only with specific solving techniques. A wide variety of issues can arise from this kind of problems, such as ill-posed/redundant matrices, odd mass ratios of bodies in contact, and the concurrent presence of flexible and rigid bodies. The development of this thesis is motivated by the need for an improved scheme that is able to deal with all the aforementioned issues at once. One of the most promising solution strategies for large scale complementarity problems, namely the Interior-Point method, is proposed and implemented in its most performing variant, the primal-dual path-following scheme, according to a custom Mehrotra predictor-corrector scheme. The sparsity of the system is leveraged in order to improve performance, and a whole set of tools, from matrix classes to linear solver interfaces, has been developed in C++ language. The algorithm has been implemented into an open-source dynamic simulation library. Thanks to the novel solver, both unilateral and bilateral constraints, involving flexible and rigid bodies can be handled using a single time-stepping scheme.

Introduction

In the context of multibody dynamics it is clear that one of the most important phenomena is the exchange of reactions between bodies, being them rigid or flexible. These may occur, for example, if a joint has been placed between them or, again, if they will come in contact.

In the case of bilateral joints, the mathematical formulation is quite trivial and does not introduce any critical issue that has not already been effectively addressed e.g. redundant constraints.

However, in case of unilateral constraints, such those that arise from modeling contacts, the dynamical simulation — considering the non-smooth nature of the event — may get more complex, especially if the problems we are dealing with involve a large number of bodies.

The very first attempt to solve this class of problems comes from introducing a fictitious reaction force between the bodies that, as soon as the contact gets more *rigid* and the contact points get closer, it is destined to increase exponentially in order to avoid any inter-penetration. This results in a very stiff reaction, whose value by the way, has to be set completely on empirical base.

Since this *regularized* approach is affine to a smooth interpretation, it is suited to be fit into an ODE scheme [1–4]. However, this approach attracted some criticism because of the small time step required by the additional stiff-force field and, although in use by a large number of commercial softwares and it has been applied to many-bodies scenarios, it may lead to unexpectedly long simulation times.

A novel implementation, based on Differential Variational Inequalities (DVI) [5–

7] has been introduced in the recent years. This category includes both those formulations based on an acceleration-force complementarity framework [8–10] and on velocity-impulse schemes [11–13]. This thesis proceeds to rewrite the dynamic problem using vector measures that will allow for a more comprehensive dissertation of impulsive events in the latter form, going towards the Measure Differential Inclusion model (MDI) [12] where also critical scenario can successfully be handled (such as the Painlevé paradox [14, 15]).

This work focuses most on frictionless constraints, for which a set of benchmark have been proposed and run (chapter 5), but at the same time proposes a solving technique also for the most complex case of contacts with friction (section 4.14). This problem can be faced following various methods [16, 17], but basically the Coulomb friction law forces the frictional reactions to lie into a Coulomb cone. One choice is to construct a polyhedral pyramid [10, 18] that approximates the cone: this leads to Linear Complementarity Problems (LCP) [19]. However, a more refined formulation expects the direct use of cones: this may lead to Cone Complementarity Problems (CCP) [20] and this is the choice for this thesis.

From this context, this thesis work commences. The final aim is to provide a new solution method that effectively cope with dynamic problems expressed through a DVI/MDI framework in order to be able to handle unilateral contacts in a definitive scheme.

Firstly, the complexity of the topic is reflected in the development of the models that describes the dynamic system [21]. For this purpose, we present an alternative set of mathematical tools capable to trace the evolution of the problem parameters — mainly velocities, acceleration and reactions — considering their impulsive nature: in chapter 1, a brief overview of these tools, used mainly in chapter 2 and chapter 3, is provided. This chapter focuses on the generic containers that will be later used in order to develop the dynamic model. Moreover, we will underline how this non-smooth mechanical problems are perfectly mirrored by certain classes of optimization problems (namely QP and Cone QP) [22], for which a limited set of solvers already exists [23–26]. Because of its polynomial complexity [25], in this thesis we have chosen to develop and implement an Interior-Point solver.

In chapter 2 the optimality conditions are stated. This is particularly useful to understand the underlying theoretical background of the Interior-Point solvers. Starting from constrained and unconstrained optimization problems and their relative necessary and sufficient conditions, we investigate the optimality conditions for the set of problems that will be later object of discussion: Quadratic Programming (QP) in section 2.10 and Cone Quadratic Programming (Cone QP) in section 2.11. As the reader will see, the optimality conditions themselves represent the first step towards the resolution of the problem. In fact, iterative solvers— among which the Interior-Point — heavily rely on them and are the foundation of the numerical scheme.

The third chapter of the thesis is dedicated to the multibody part, in which the dynamic equations are subject to significant changes in order to address impulsive events in order to simulate perfectly-rigid contacts. This part is mainly provided to support the main idea that the contact formulation in non-smooth dynamic has close relationship to the class of QP problems on which the Interior-Point of the previous chapter is run. Although not widely comprehensive on all the in-detail aspects of the non-smooth formulation, chapter 3 guarantees a good base for those who are interested in looking for an alternative representation of rigid and flexible bodies simulations involving contacts. Moreover, it represents a fundamental step to understand the link between convex optimization problems and multibody dynamics.

After having discussed the theoretical aspects, chapter 4 finally explores the Interior-Point Method itself. In this text only the application to a QP and Cone QP problems is investigated. The Newton's iterative scheme is here re-proposed applied to the Karush-Kuhn-Tucker conditions system, that give the very first backbone to the IP algorithm, on which a wide set of improvements have been plugged in. The already well-known centering steps, together with the prediction-correction scheme proposed my Mehrotra, formed the whole picture of the implemented solver.

The algorithm is presented also in a synthetic and ready-to-use form for further reference (section 4.11), accompanied with a pseudo-code section 4.11.1 and a cut version of the C++ implementation (appendix B), where just some auxiliary functions have been omitted. After the main part of the chapter, also an early-developed Interior-Point algorithm for Cone QP is presented.

The results presented in chapter 5 show some critical scenarios in which the algorithm has been tested, together with some standard-cases simulations in which the performance is addressed.

The Interior-Point Method presented in this thesis is part of the open-source Project Chrono (`https://projectchrono.org/`), a suite of libraries and tools with focus on multibody dynamics with contacts, that has also extensively been used in the simulation loop in order to feed the solver with realistic inputs, originated by the dynamic simulation. The library includes also the collision detection algorithm, convex decomposition methods, finite elements as well as rigid bodies, together with a wide set of auxiliary tools that facilitate the construction of complex multibody systems. The tests proposed in chapter 5, indeed, are run thanks to the Chrono::Engine library.

Chapter 1

Mathematical Background

Impulsive events are not commonly suited to be described as continuous functions thus, for that task, a more involved formulation based on *measures*, instead of functions, has to be brought into play. Contacts are nothing less than one of such cases; their instantaneous nature, in fact, asks for a new non-smooth dynamic model, rewritten following these concepts [27].

At the same time, while fitting the contacts dynamic into equations of motion together with the relative constraint equations, another problem arises: the solving technique has to effectively deal with this new kind of mathematical system. Because of this, some background on numerical solving techniques and algorithms is presented.

The following sections summarize the basic concepts that are strictly needed to understand the subsequent development of the Interior Point Method discussed in this thesis.

1.1 Measures

The mathematical concept of *measures* is not so far away from the common way of thinking it. Basically it associates a number to a subset of a given space, that represents in some meaningful sense its "size". Although the topic is vast, we restrict

ourselves to some specific measures and related properties that are of some interest for our discussion.

Given X, a generic set, a σ-algebra over X, a *measure* ν is a function $\mu : \Sigma \to \bar{\mathbb{R}}$ that associates, to any subset Σ of X, a number $\nu(\Sigma)$ from the extended non-negative real set $\mathbb{R} \cup \{+\infty\}$ having the following properties:

Non-Negativity the measure is always non-negative $\nu(E) \geq 0 \quad \forall E \subset \Sigma$, if this does not apply the measure is referred as *signed measure*;

Null Empty Set it associates the null measure to empty sets: $\nu(\emptyset) = 0$;

Countable Additivity the measure of the countable union of pairwise disjoints sets is equal to the sum of the measures of each set. So, given an arbitrary collection $\{E_i\}_{i=1}^{\infty}$ it holds $\nu\left(\bigcup_k^{\infty} E_k\right) = \sum_{k=1}^{\infty} \nu(E_k)$.

Moreover, we can identify measures with relevant features that could be of some interest. The *Lebesgue* measure operates on subsets of n-dimensional Euclidean space $X \subset \mathbb{R}^n$ and it has a very intuitive meaning when associated to subsets of dimension 1, 2 or 3, since it translates into length, area, or volume of the given subset.

Another useful measure named after Johann Radon is the one that operates on Borel sets of locally-finite Hausdorff topological spaces. The *Radon measure* is such that $\forall x \in X, \exists \setminus_x : |\nu(\setminus_x)| < +\infty$ where $\setminus_x$ represents a open neighborhood of x. So, for every set contained in X, it follows that its Radon measure is finite.

Given the definition of measure, it can be also defined what is a *measurable space* that is the pair of the set on which the measure is defined, X, and the associated σ-algebra Σ and it is expressed as (X, Σ).

Applying two measures ν and μ to a measurable space (X, Σ) it is said that they are mutually singular $\nu \perp \mu$ if exists a set $\mathscr{M} \in \Sigma$ such that:

$$\mu(\mathscr{M}) = 0 \quad \wedge \quad \nu(X \setminus \mathscr{M} = 0)$$

A measure ν is *absolutely continuous* respect to the measure μ, or $\nu \ll \mu$, if

$$\mu(A) = 0 \implies \nu(A) = 0 \quad \forall A \in \Sigma$$

Two crucial theorems are the following:

Lebesgue decomposition theorem Given two σ-finite measures ν and μ on a measurable space X,Σ, there exist a unique decomposition for which

$$\nu = \nu_c + \nu_s$$

with $\nu_c \ll \mu$ and $\nu_s \perp \mu$. Moreover, $\nu_c \perp \nu_s$.

Radon-Nikodym theorem Given the σ-finite absolutely continuous measures $\nu_c \ll \mu$ on a measurable space X,Σ and with $A \in X$, there exist a measurable function $f : X \implies [0,+\infty)$ such that:

$$\nu_c(A) = \int_A f d\mu$$

Conventionally, f is written as $\frac{d\nu_c}{d\mu}$ and it is called the *Radon-Nikodym derivative*.

1.2 Convex Sets, Algebra and Cones

The formulation of the contact model is based on second-order cones that are here presented. In this section, also the fundamental concepts of convex sets and algebra are introduced.

Given S a vector space over the real numbers, then P set is said to be:

Convex if $x,y \in P$ and $t \in (0,1)$ then also the points $x(1-t)+yt$ still belongs to P;

Closed such that it contains its limit points;

Compact if, for every collection C of open subsets of P such that $P = \bigcup_{x \in C} x$, there is a *finite* subset F of C such that $P = \bigcup_{x \in F} x$;

Among the others, we are interested in *cones* for which the most generic definition is the following. A set $\mathscr{K} \in$ is said to be a *cone* if:

$$\forall \boldsymbol{x} \in \mathscr{K}, \forall \beta \in \mathbb{R}^+ \implies \beta \boldsymbol{x} \in \mathscr{K} \tag{1.1}$$

it is said to be:

Full if $int(\mathscr{K}) \neq \{\emptyset\}$;

Pointed if $\mathscr{K} \cap -\mathscr{K} = \emptyset$

Proper if it is close, convex and full

We describe convex cones with apex angle $2\arctan(\mu)$ along the x axis, also as:

$$\mathscr{K}_\mu = \left\{ \begin{bmatrix} \boldsymbol{x}_{[0]} \\ \boldsymbol{x}_{[1]} \end{bmatrix} \in \mathbb{R} \times \mathbb{R}^{p-1} \mid \mu \boldsymbol{x}_{[0]} \geq \|\boldsymbol{x}_{[1]}\| \right\} \tag{1.2}$$

Another equivalent expression is the following:

$$\mathscr{K}_\mu = \left\{ \begin{bmatrix} \boldsymbol{x}_{[0]} \\ \boldsymbol{x}_{[1]} \end{bmatrix} \in \mathbb{R} \times \mathbb{R}^{p-1} \mid \boldsymbol{x}^T J \boldsymbol{x} \geq 0,\, x_{[0]} \geq 0 \right\} \quad \text{with} \quad \begin{bmatrix} \mu^2 & 0 \\ 0 & -I_{p-1} \end{bmatrix} \tag{1.3}$$

where the subscript $_{[0]}$ picks the first component of the vector and $_{[1]}$ the remaining.

Given a cone $\mathscr{K}$ we use the symbols of *generalized inequality* $\succeq$ and $\preceq$ in the following notation:

$$\begin{aligned} \boldsymbol{x} \succeq_{\mathscr{K}} 0 &\iff \boldsymbol{x} \in \mathscr{K} \\ \boldsymbol{x} \succeq_{\mathscr{K}} \boldsymbol{y} &\iff \boldsymbol{x} - \boldsymbol{y} \in \mathscr{K} \\ \boldsymbol{x} \succ_{\mathscr{K}} \boldsymbol{y} &\iff \boldsymbol{x} - \boldsymbol{y} \in \operatorname{int}(\mathscr{K}) \end{aligned}$$

We now describe some attributes that relate to cones:

Dual Cone Given a set $\mathscr{K}$ in a generic real vector space equipped with an inner product, a *dual cone* $\mathscr{K}^*$ is referred as:

$$\mathscr{K}^* = \{\boldsymbol{y} \in \mathbb{R}^n : \langle \boldsymbol{y}, \boldsymbol{x} \rangle \geq 0 \quad \forall \boldsymbol{x} \in \mathscr{K}\}. \tag{1.4}$$

Dual cones are convex regardless the convexity of the initial space P. Moreover, $\mathscr{K}$ is not required to be a cone.

Polar Cone it is specular to the dual, respect to the hyperplane orthogonal to the cone axis. It is defined as:

$$\mathscr{K}^\circ = \{\boldsymbol{y} \in \mathbb{R}^n : \langle \boldsymbol{y}, \boldsymbol{x} \rangle \leq 0 \quad \forall \boldsymbol{x} \in \mathscr{K}\} = -\mathscr{K}^*. \tag{1.5}$$

Normal Cone Given a a closed convex set $\mathscr{K}$ and a point $\boldsymbol{x} \in \mathscr{K}$, the *normal* cone is closed and convex. It is defined as:

$$\mathscr{N}_{\mathscr{K}}(\boldsymbol{x}) = \{\boldsymbol{y} \in \mathbb{R}^n : \langle \boldsymbol{y}, \boldsymbol{x} - \boldsymbol{z} \rangle \geq 0, \forall \boldsymbol{z} \in \mathscr{K}\} \tag{1.6}$$

Note that if $\boldsymbol{x}$ is an interior point of $\mathscr{K}$, it is always $\mathscr{N}_{\mathscr{K}}(\boldsymbol{x}) = \{\emptyset\}$.

Tangent Cone Given a closed convex set $\mathscr{K}$ and a point $\boldsymbol{x} \in \mathscr{K}$, the *tangent* cone is closed and convex. It is defined as:

$$\mathscr{T}_{\mathscr{K}}(\boldsymbol{x}) = \mathrm{cl}\{\beta(\boldsymbol{y} - \boldsymbol{x}) : \boldsymbol{y} \in \mathscr{K}, \beta \in \mathbb{R}^+\} = \mathscr{N}_{\mathscr{K}}(\boldsymbol{x})^\circ \tag{1.7}$$

Second-Order Cone or *Lorentz Cone* is a self-dual, self-scaled, symmetric cone defined as

$$\mathscr{K} = \left\{(x_0, \boldsymbol{x}_1) \in \mathbb{R} \times \mathbb{R}^{p-1} : ||\boldsymbol{x}_1||_2 \leq x_0\right\} \tag{1.8}$$

1.3 Variational Inequalities

Expressing contacts by means of *measures* (section 1.1) is just one of the required steps in order to effectively describe this phenomena in a mathematical form.

We now introduce the concept of Variational Inequalities (VI), first introduced in the effort of solving the *problem with ambiguous boundary conditions* also known as the Signorini problem.

Given a Banach space E, a closed and convex subset $\mathscr{K}$ of E and a continuous functional $F : \mathscr{K} \mapsto E^*$ (i.e. to the dual space of E), a *Variational Inequality* problem is the problem of finding the solution $\boldsymbol{x} \in \mathscr{K}$ of described as:

$$\boldsymbol{x} \in \mathscr{K} \quad : \quad \langle \boldsymbol{F}(\boldsymbol{x}), \boldsymbol{y} - \boldsymbol{x} \rangle \geq 0 \quad \forall \boldsymbol{y} \in \mathscr{K}, \tag{1.9}$$

We will refer to the solution of this problem VI(F, $\mathscr{K}$) as $\mathsf{SOL}(\mathscr{K}, F)$.

Starting from the definition 1.6, one can note that the VI problem states that the functional F has to stay in the normal cone of the subset $\mathscr{K}$, that is:

$$\boldsymbol{x} \in \mathscr{K} \quad : \quad F(\boldsymbol{x}) \in \mathscr{N}_{\mathscr{K}}(\boldsymbol{x})$$

We then consider the fact that the normal cone $\mathcal{N}_{\mathcal{K}}$ is not trivial only if the point $\boldsymbol{x}$ considered lies on the boundary $\partial\mathcal{K}$.

The discussion regarding existence and uniqueness of the solution is more easily developed when restricting ourselves to specific cases. A solution to the problem 1.9:

- exists if $\mathcal{K}$ is *compact* (and convex).
- exists if $F(\cdot)$ is *cohercive*, i.e. if:

$$\frac{\langle F(\boldsymbol{x}) - F(\boldsymbol{x}_0), \boldsymbol{x} - \boldsymbol{x}_0 \rangle}{|\boldsymbol{x} - \boldsymbol{x}_0|} \implies \infty \quad \text{as } |\boldsymbol{x}| \implies \infty$$

- is unique if $F(\cdot)$ is *monotone*, i.e. if:

$$\langle F(\boldsymbol{x}) - F(\boldsymbol{x}_0), \boldsymbol{x} - \boldsymbol{x}_0 \rangle > 0 \quad \forall \boldsymbol{x}, \boldsymbol{x}_0 \in \mathcal{K}$$

Introducing VIs is the key to describing many problems arising not only from multibody dynamics, but more in general in mathematical programming.

From generic VIs, we restrict ourselves to some problems that are of specific interest, since they are the most effective to describe the contact problem:

Referring to problem 1.9:

- Choosing the subset $\mathcal{K} = \mathbb{R}^n_+$ the problem can be restated as a *Nonlinear Complementarity Problem* (NCP), that is the problem of finding a $\boldsymbol{x}$ that satisfies:

$$\boldsymbol{x} \geq \boldsymbol{0}, \quad \boldsymbol{F}(\boldsymbol{x}) \geq \boldsymbol{0}, \quad \langle \boldsymbol{F}(\boldsymbol{x}), \boldsymbol{x} \rangle = 0 \tag{1.10}$$

  also written in a the more compact notation:

$$\boldsymbol{x} \geq \boldsymbol{0} \quad \perp \quad \boldsymbol{F}(\boldsymbol{x}) \geq \boldsymbol{0}$$

- If $\boldsymbol{F}$ is an *affine* function, we could write it as $\boldsymbol{F}(\boldsymbol{x}) = A\boldsymbol{x} - \boldsymbol{b}$. The VI problem with $\mathcal{K} = \mathbb{R}^n_+$ and affine F is called *Linear Complementarity Problem* (LCP) and is the problem of finding a $\boldsymbol{x}$ that satisfies

$$\boldsymbol{x} \geq \boldsymbol{0}, \quad A\boldsymbol{x} - \boldsymbol{b} \geq \boldsymbol{0}, \quad \langle A\boldsymbol{x} - \boldsymbol{b}, \boldsymbol{x} \rangle = 0$$

also written in the more compact notation:

$$\mathbf{0} \leq \boldsymbol{x} \quad \perp \quad A\boldsymbol{x} - \boldsymbol{b} \geq \mathbf{0}$$

the LCP is equivalent to a VI where $\mathscr{K} = \mathbb{R}^n_+$ and with affine $\boldsymbol{F}$:

$$\boldsymbol{x} \in \mathbb{R}^n_+ \quad : \quad \langle A\boldsymbol{x} - \boldsymbol{b}, \boldsymbol{y} - \boldsymbol{x} \rangle \geq 0 \quad \forall \boldsymbol{y} \in \mathbb{R}^n_+$$

- In the previous Complementarity Problems we see that the subset $\mathscr{K}$ was restricted to be a non-negative orthant. In a *Cone Complementarity Problem* the subset $\mathscr{K}$ is now chosen to be a cone Γ. The problem is to find a $\boldsymbol{x}$ that satisfies:

$$\boldsymbol{x} \in \Upsilon, \quad A\boldsymbol{x} - \boldsymbol{b} \in -\Upsilon^o, \quad \langle A\boldsymbol{x} - \boldsymbol{b}, \boldsymbol{x} \rangle = 0 \tag{1.11}$$

also written in the more compact notation:

$$\Upsilon \ni \boldsymbol{x} \quad \perp \quad A\boldsymbol{x} - \boldsymbol{b} \in -\Upsilon^o$$

where Υ is a (convex) cone, and if it is a second-order Lorentz cone, one has a CCP. The CCP is equivalent to a VI where $\mathscr{K} = \Upsilon$ and with affine $\boldsymbol{F}$:

$$\boldsymbol{x} \in \Upsilon \quad : \quad \langle A\boldsymbol{x} - \boldsymbol{b}, \boldsymbol{y} - \boldsymbol{x} \rangle \geq 0 \quad \forall \boldsymbol{y} \in \Upsilon$$

1.4 Differential Problems

In the following we present some definitions about ODEs, DAEs, DIs, DVIs, etc. Differential inclusions can be interpreted as a generalization of ODEs to the case of discontinuous right hand side.

Ordinary Differential Equation (ODE)

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x}, t)$$

with given initial value $\boldsymbol{x}(t_0) = \boldsymbol{x}_0$. Existence and uniqueness of the solution is guaranteed, in case of Lipschitz continuity of $\boldsymbol{f}(\boldsymbol{x}, t)$, by Cauchy-Lipschitz and Picard-Lindelhof theorems;

Differential Algebraic Equation (DAE) can be expressed in an implicit form

$$\boldsymbol{F}\left(\frac{d\boldsymbol{x}}{dt},\boldsymbol{x},t\right)=0$$

or in the more common *semi-implicit* form,

$$\begin{aligned}\frac{d\boldsymbol{x}}{dt}&=\boldsymbol{f}(\boldsymbol{x},t)\\ \boldsymbol{g}(\boldsymbol{x},t)&=\boldsymbol{0}\end{aligned}$$

with the initial value of $\boldsymbol{x}(t_0)=\boldsymbol{x}_0$.

Differential Inclusion (DI)

$$\frac{d\boldsymbol{x}}{dt}\in\mathscr{F}(\boldsymbol{x},t)$$

with $\mathscr{F}(\boldsymbol{x},t)$ with closed graph.

Differential Variational Inequality (DVI)

$$\begin{aligned}\frac{d\boldsymbol{x}}{dt}&=\boldsymbol{f}(\boldsymbol{x},\boldsymbol{u},t)\\ \boldsymbol{u}&\in\mathsf{SOL}(\boldsymbol{F},\mathscr{K})\end{aligned}$$

where the solution of a VI problem $(\boldsymbol{F},\mathscr{K})$ is put together to a differential equation.

Measure Differential Inclusion (MDI)

$$\frac{d^2\boldsymbol{q}}{dt^2}\in\mathscr{K}(\boldsymbol{q},t) \tag{1.12}$$

where $d\boldsymbol{q}/dt$ is a function of bounded variation and $\mathscr{K}(\boldsymbol{q},t)$ is a set-valued function with closed graph and closed convex values. It can accommodate impulsive events.

Chapter 2

Optimization

The intrinsic behavior of nature is strictly bonded to optimization: the *most* fit survives, the light always takes the *shortest* path, a body always lies where the potential energy is at its *minimum*. We ourselves, as human beings, we are in a constant strive to obtain the *most* from our lives and our jobs. We aim to improve efficiency of our cars, harvests, power plants, always getting more spending less.

Optimization is crucial for life as well for engineering and science. Because bodies are but optimizing creatures, we, as engineers and scientists, should mimic their behavior in order to capture the laws that dictate their motion and interactions.

Three main ingredients are required to formulate an optimization problem:

Objective is a function that maps to a single scalar value that quantifies the performance of the given system configuration;

Variables are the parameters that uniquely define a given system configuration and on which the objective function depends;

Constraints are the bounds that the variables must respect in order to be considered actual solution of the problem. A solution that respects the constraints is said to be *feasible*.

The task of optimization is clear: to provide a set of values that minimizes (or maximizes) the objective function while respecting the constraints. However, the recipe

is not trivial and definitely not universal. A whole set of algorithms has been developed over the years, each of which has been found to give better results when applied to a specific type of problems. In fact, the complexity and variety of tasks that nowadays require an optimization process are always increasing and the demand for more efficient solving techniques is compelling.

Two questions arises. The first: how can I assess the *optimality* of a set of variables? It is clear if a tentative solution is *better* of another — we have the objective function to decree this — but surely it is not as trivial to assess if a set of variables is the *best* possible. Fortunately, for a wide range of problems, there exist *optimality conditions* that partially provide the answer and, moreover, that could give also a deep insight into how the approximate solution scores compared to the optimal one. We will focus on this in section 2.8.

The second question is how to judge the performance of an algorithm: unlike the previous question, we do not even have an easy way to assert if an algorithm is strictly *better* than another. So the judgment cannot infer anything about the performance of the algorithm itself, but it should always be related to the pair algorithm – problem-being-solved. An useful way to accomplish this is to feed a given optimization tool with a widely-recognized problem, such as those provided, for the multibody contacts field, by the *fclib* library [28], and compare the results.

Usually the performance are evaluated on the base of:

Robustness an algorithm could perform well for a limited set of problems while being less effective, or even completely unusable, for others; moreover, many optimization procedures are also sensitive to the starting point, or they could fail when asked to converge to extreme degrees of accuracy;

Accuracy the ability to provide a solution that approximates the optimality with the desired tolerance; in particular, also the code implementation could affect the precision, especially when the order of magnitude of rounding errors is comparable to the scale of the variables;

Efficiency the amount of resources, expressed both in terms of time and storage, that are required by the algorithm.

In the following sections we investigate optimality conditions and then we will provide a description of a peculiars set of problems, namely QP (section 2.10) and Cone QP (section 2.11), that will be at the base of our Interior-Point algorithm.

2.1 Global and Local Minimizer

Discarding for the moment the constraints, the problem of finding the minimum of a function f could be stated in this form:

$$\boldsymbol{x}^* = \operatorname{argmin} f(\boldsymbol{x})$$

(or argmax in case we are looking for the maximum).

However, even though a solution respects the optimality conditions — and that are mathematical expressions that tells if a set of variables is optimal for a given problem — we cannot expect to have found *the* optimal point. Being feasible and optimal is not sufficient in general: we have also to say in which neighborhood this optimality applies. It could be possible that *multiple local* extremal points exist and no general rule holds to decree if *an* extremal point is the only one. However, for specific problems some interesting deductions can be done, like the case of *convex* problems (see section 2.3).

A point x^* is said to be minimizer if

$$f(\boldsymbol{x}^*) \leq f(\boldsymbol{x}) \quad \forall \boldsymbol{x} \in S$$

then, if this statement applies:

- for $S = \mathbb{R}^n$ or for the whole space of interest, then the minimizer is *global*;
- only for a neighborhood of $\boldsymbol{x}$, then the minimizer is *local*.

2.2 Necessary and Sufficient Conditions

Checking every point in the neighborhood, looking for lower values of the objective function can be an idea, but in the case of *smooth* functions there is a better choice.

The *necessary optimality conditions* in fact, can be easily inferred from the Taylor's theorem [29, p.274]: assuming that x^* is a local minimizer we claim some properties on the derivatives.

Theorem 2.2.0.1 (Necessary Conditions)

$$\boldsymbol{x}^* \textit{ is a local minimizer} \implies \begin{cases} \nabla f(\boldsymbol{x}^*) = 0 & \textit{First-Order N.C.} \\ \nabla^2 f(\boldsymbol{x}^*) \geq 0 & \textit{Second-Order N.C.} \end{cases}$$

assumed the existence and continuity of the required derivatives in an open neighborhood of $\boldsymbol{x}^*$.

For the proof [29, pag.15]

Theorem 2.2.0.2 (Sufficient Conditions)

$$\begin{cases} \nabla f(\boldsymbol{x}^*) = 0 \\ \nabla^2 f(\boldsymbol{x}^*) > 0 \end{cases} \implies \boldsymbol{x}^* \textit{ is a (strict) local minimizer}$$

given the existence and continuity of the Hessian in an open neighborhood of $\boldsymbol{x}^*$.

For the proof [29, p.16]

As we can see, the role of the gradient is fundamental in order to catch stationarity points that luckily are also minimizers of the function. Optimization algorithms usually leverage this kind of information, especially the first derivative.

As the reader may have noticed, most of the theorems require *smooth* functions, not always easy to get in a multibody environment involving contacts. Luckily, we will find that a sort of smoothness is kept while analyzing our dynamic system that will allow to fit the problem into this description.

2.3 Convex Programming

As we have previously seen in section 1.2, a set in $\mathbb{R}^n$ is said to be convex if, considering any two points contained in it, a straight line lies entirely in the set itself.

However, also functions have a convex attribute: in order for a function f to be convex:

- its domain has to be convex;
- for any pair of points $\boldsymbol{x}_1, \boldsymbol{x}_2$ in the domain of f, it must hold

$$f(\boldsymbol{x}_1) + \alpha(f(\boldsymbol{x}_2) - f(\boldsymbol{x}_1)) \geq f(\boldsymbol{x}_1 + \alpha(\boldsymbol{x}_2 - \boldsymbol{x}_1)) \quad \forall \alpha \in [0,1]$$

A *concave* function is nothing but a function for which $-f$ is convex.

Even though convex functions cover only a limited set of problems, they hold a relevant role in the optimization world, so we will suggest some eminent examples such as:

Linear Functions $f(\boldsymbol{x}) = \boldsymbol{a}^T\boldsymbol{x} + b$ where, as the symbols suggests, $\boldsymbol{a}$ is a constant vector and b is a scalar;

Convex Quadratic Functions $f(\boldsymbol{x}) = \boldsymbol{x}^T A \boldsymbol{x}$ where the matrix A is symmetric and positive semidefinite.

Similarly, frequently encountered convex sets are:

Polyhedrons $\boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} = \boldsymbol{b},\ C\boldsymbol{x} \leq \boldsymbol{d}$

Unit Ball $\boldsymbol{x} \in \mathbb{R}^n \mid \|\boldsymbol{x}\|_2 \leq 1$

Since the definitions for sets and functions are given, we eventually introduce the definition of the *convex programming problem* that is an optimization problem in which:

- the objective function is convex;
- the equality constraints, if present, are linear;
- the inequality constraints, if present, are concave.

For such problems we are assured that *local* solutions are also *global* [29].

Theorem 2.3.0.1
If f is convex and either

$$\boldsymbol{x}^* \textit{ is a local minimizer} \quad \vee \quad \begin{cases} f \textit{ is differentiable} \\ \boldsymbol{x}^* \textit{ is a stationary point} \end{cases}$$

then $\boldsymbol{x}^*$ is a global *minimizer.*

For the proof [29, p.16]

2.4 The Line Search Intuition

We reserve a section to discuss briefly the *line search* technique for unconstrained optimization because of the conceptual importance that is going to have even in the Interior-Point method.

As any other optimization algorithm, line search is iterative: starting from a given point $\boldsymbol{x}_0$ it moves towards a second point that minimizes the objective function, given the direction $\boldsymbol{p}$. So, as in Interior Point, the degrees of freedom to tune are:

- the direction $\boldsymbol{p}$
- the step length α

that is

$$\alpha^* = \underset{\alpha>0}{\operatorname{argmin}} f(\boldsymbol{x}_k + \alpha \boldsymbol{p}_k) \tag{2.1}$$

If we want to have $f(\boldsymbol{x}_k + \alpha \boldsymbol{p}_k) < f(\boldsymbol{x}_k)$, recalling the Taylor's Theorem,

$$\begin{aligned} f(\boldsymbol{x}_k + \alpha \boldsymbol{p}_k) = f(\boldsymbol{x}_k + \boldsymbol{h}_k) = f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k + t\boldsymbol{h}_k)^T \boldsymbol{h}_k < f(\boldsymbol{x}_k) \\ \nabla f(\boldsymbol{x}_k + t\boldsymbol{h}_k)^T \boldsymbol{h}_k < 0 \end{aligned}$$

so, it is clear to see that, in order to reduce the objective function, one should follow the opposite direction provided by the gradient $\nabla f(\boldsymbol{x}_k)$. Taking exactly the direction $\boldsymbol{p}_k$ along $-\nabla f(\boldsymbol{x}_k)$ is called *steepest descent* method.

2.5 Constrained Optimization Problem

Dealing with minima and maxima of the objective is just one of the tasks that we are going to deal with. Together with accomplishing the reduction of this function, we now add the constraints to the problem. This introduces two main factors: first, the feasible set through which the solution has to be searched is narrower; second, the optimization algorithm should be able to take into account the introduced barriers.

The task is to find $\boldsymbol{x}^*$ that

$$\boldsymbol{x}^* = \operatorname{argmin} f(\boldsymbol{x}) \quad s.t. \quad \begin{cases} C_i(\boldsymbol{x}) = 0 & i \in \mathscr{E} \\ C_i(\boldsymbol{x}) \geq 0 & i \in \mathscr{I} \end{cases} \tag{2.2}$$

where the $i \in \mathscr{E}$ are the index of the equality constraints and $i \in \mathscr{I}$ are the inequalities constraints. Still the objective function and the constraints are smooth and on a subset of $\mathbb{R}^n$.

The set that is considered *feasible* is given by

$$\Omega = \left\{ \boldsymbol{x} \in \mathbb{R}^n : \begin{cases} C_i(\boldsymbol{x}) = 0 & i \in \mathscr{E} \\ C_i(\boldsymbol{x}) \geq 0 & i \in \mathscr{I} \end{cases} \right\} \tag{2.3}$$

so, eq. (2.2) can be rewritten as

$$\boldsymbol{x}^* = \operatorname*{argmin}_{\boldsymbol{x} \in \Omega} f(\boldsymbol{x}) \tag{2.4}$$

2.6 Dealing with Constraints

In section 2.2 we already presented the optimality conditions to which we will now add some additional requirement in order to extend this considerations to constrained optimization.

Suppose to have a point $\boldsymbol{x}$ for which we argue its optimality: in order to verify this statement, we would try to make a *step* $\boldsymbol{s}$ — that is, to move to another point close to $\boldsymbol{x}$ — and look if there is, in the neighborhood, a better approximation of the solution. *If not*, then it is *likely* that the current point $\boldsymbol{x}$ is an optimal solution. It is

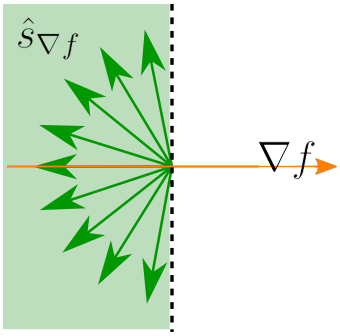


Figure 2.1: *Allowed directions that decrease the* objective function*; in orange, the gradient of the objective function, in green the allowed directions of the step.*

clear that this new point $\boldsymbol{x}+\boldsymbol{s}$, to be actually a *better* solution, must respect these two conditions:

1. Stationarity: the objective function must not have a higher value;
2. Feasibility: the constraints must be respected.

So, the next question is: for which point it is not possible to make any step? This point will (probably) be a local optimal solution, indeed.

For the first condition — the only that we have in the unconstrained case — it is asked that the step $\boldsymbol{s}$ must respect the condition $f(\boldsymbol{x}+\boldsymbol{s}) < f(\boldsymbol{x})$ or, linearizing the function f in the neighborhood of the approximated solution $\boldsymbol{x}$:

$$f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^T \boldsymbol{s} < f(\boldsymbol{x}) \quad \Longrightarrow \quad \nabla f(\boldsymbol{x})^T \boldsymbol{s} < 0 \tag{2.5}$$

The sub-space in which the vector $\boldsymbol{s}$ must lie is then an *open* half-space, delimited by the (hyper)plane orthogonal to $\nabla f(\boldsymbol{x})$ i.e. $\nabla f(\boldsymbol{x})^T \boldsymbol{s} = 0$, as we can see in fig. 2.1.

For the second condition, we have to face different situations, depending on the type of the constraint equation.

- Equality Constraints $\mathscr{E}$;
- Inequality *Active* Constraints $\mathscr{I}_A \subset \mathscr{I}$;

- Inequality *Inactive* Constraints $\mathscr{I}_I \subset \mathscr{I}$;

Given a constraint $C_i(\boldsymbol{x})$ we say that it is

Active if $C_i(\boldsymbol{x}) = 0$ i.e. is on the border of the feasible set;

Inactive if $C_i(\boldsymbol{x}) > 0$ i.e. is strictly *inside* the feasible set.

We then introduce:

Definition 2.6.0.1
The active set $\mathscr{A}(\boldsymbol{x})$ *is the set of constraint equation indexes for which* $C_i(\boldsymbol{x}) = 0$

$$\mathscr{A}(\boldsymbol{x}) = \mathscr{E} \cup \mathscr{I}_A$$

Under the assumption that the current approximated solution is feasible, it must hold that also the new point $x+s$ is within the constraint boundaries.

For the equality case, this can be summarized with the expression $C_i(\boldsymbol{x}) = C_i(\boldsymbol{x}+\boldsymbol{s}) = 0$ where i is picked from the equality index set $\mathscr{E}$. Linearizing C_i in the neighborhood of $\boldsymbol{x}$ we get

$$\begin{cases} C_i(\boldsymbol{x}+\boldsymbol{s}) = 0 \\ C_i(\boldsymbol{x}+\boldsymbol{s}) \approx \underbrace{C_i(\boldsymbol{x})}_{C_i(\boldsymbol{x})=0} + \nabla C_i(\boldsymbol{x})^T \boldsymbol{s} \end{cases} \implies \nabla C_i(\boldsymbol{x})^T \boldsymbol{s} \approx 0 \tag{2.6}$$

This condition imposes the step to be orthogonal to the gradient of the constraint function i.e. to stay, at least in first-order approximation, along the boundary, as we can see in the left-most figure in fig. 2.2.

As we already mentioned, if both the stationarity (eq. (2.5)) and feasibility conditions *cannot* be met at the same time, then it is likely to have an optimal solution. For the equality constrained case the feasibility conditions is represented by eq. (2.6) so that, considering eq. (2.5), no step exists only when the gradients of f and C_i are parallel. In fact, eq. (2.5) allows steps in an entire open half-space. The condition expressed by eq. (2.6) on the contrary, allow only steps on the plane $\nabla C_i(\boldsymbol{x})^T \boldsymbol{s} = 0$. The intersection of this two conditions is empty only when this latter plane is the border

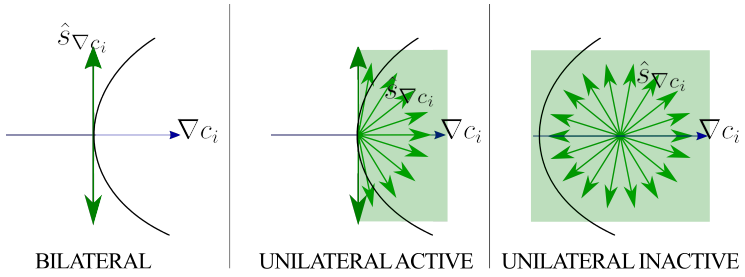
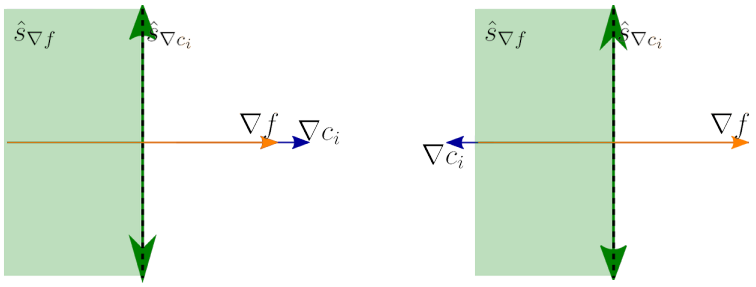


Figure 2.2: *Allowed directions that respect the* constraints equations*; in blue, the gradient of an arbitrary constraint function, in green the allowed directions of the step.*



(a) Concurrent gradients direction. **(b)** Opposite gradients direction.

Figure 2.3: *Bilateral constraints. Conditions for which no feasible steps exist.*

of the open half-space i.e. when $\{\nabla f(\boldsymbol{x})^T \boldsymbol{s} = 0\} = \{\nabla C_i(\boldsymbol{x})^T \boldsymbol{s} = 0\}$. In order for these planes to overlap, it is necessary that their orthogonal vectors are parallel (fig. 2.3). We write this condition in a more general form, that will be useful to describe also the other cases:

$$\nabla f(\boldsymbol{x}) = \lambda_i \nabla C_i(\boldsymbol{x}) \tag{2.7}$$

We must stress that, in this case, there is no restriction on the sign of λ_i since it is not necessary that the two gradients share the same orientation, but only the same direction. See fig. 2.3.

For the inequality case the feasibility condition is slightly different from the

equality case because of the presence of the $\geq$ operator:

$$\begin{cases} C_i(\boldsymbol{x}+\boldsymbol{s}) \geq 0 \\ C_i(\boldsymbol{x}+\boldsymbol{s}) \approx C_i(\boldsymbol{x}) + \nabla C_i(\boldsymbol{x})^T \boldsymbol{s} \end{cases} \tag{2.8}$$

At this point, we should make a distinction about the *current* value of $C_i(\boldsymbol{x})$. If the constraint is:

- inactive, then $C_i(\boldsymbol{x}) > 0$, so no statement can be made about the gradient $\nabla C_i(\boldsymbol{x})$;
- active, so that $C_i(\boldsymbol{x}) = 0$, so we will have the requirement $\nabla C_i(\boldsymbol{x})^T \boldsymbol{s} \geq 0$.

In the case of *inactive* constraint the point $\boldsymbol{x}$ lies *strictly inside* the feasible region i.e. it is not on the boundary. This means that we are allowed to move in any direction without hitting the constraint, so it is clear that we can *always* make a step, as we can see in the right-most figure in fig. 2.2; for example, in the direction suggested by the Line Search Method 2.4. The only exception is when $\nabla f(\boldsymbol{x}) = 0$; in this case we already have a stationary point. Thus, the only condition for which the point is an optimal solution is:

$$\begin{cases} \nabla f(\boldsymbol{x}) = \lambda_i \nabla C_i(\boldsymbol{x}) \\ \lambda_i = 0 \end{cases} \implies \nabla f(\boldsymbol{x}) = 0 \tag{2.9}$$

In the last case, the *active* inequality constraint has $C_i(\boldsymbol{x}) = 0$, thus the condition on the gradient $\nabla C_i(\boldsymbol{x})^T \boldsymbol{s} \geq 0$ implies that the step is within a *closed* half-space delimited by the plane orthogonal to $\nabla C_i(\boldsymbol{x})$, as we can see in the center figure in fig. 2.2. In order to have a (possibly) optimal solution we should have:

$$\begin{cases} \nabla f(\boldsymbol{x}) = \lambda_i \nabla C_i(\boldsymbol{x}) \\ \lambda_i \geq 0 \end{cases} \tag{2.10}$$

It has to be noticed that, in this case, in order to *not* have any feasible step, not only the gradient of the constraint should lie on the same line of the gradient of the objective function, but the two must share also the same direction, as we can see in fig. 2.5

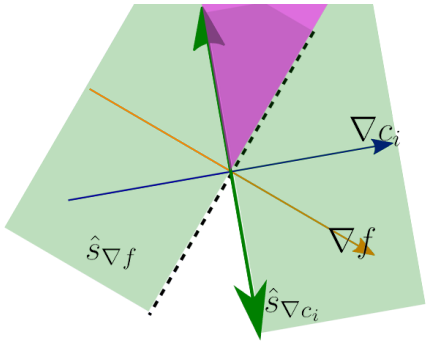


Figure 2.4: *Allowed directions that minimize the objective function* and *respect the constraint (unilateral, in this case); in blue, the gradient of an arbitrary constraint function, in orange the gradient of the objective function, in purple the allowed directions of the step.*

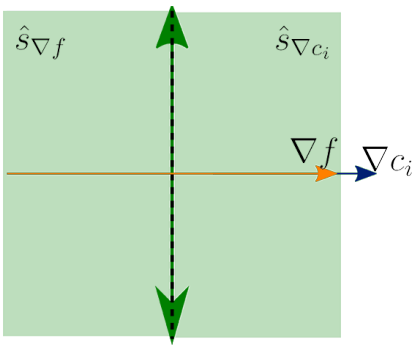


Figure 2.5: *Unilateral constraints. Conditions for which no feasible steps exist.*

In general, equations 2.7, 2.9, 2.10 come from observing the relative directions of the gradients of f and C_i and of the possible step directions. We summarize here these conditions and we recall once more that they describe the case for which *no steps* are allowed and thus, for which it is likely to have reached an optimal solution; these conditions are described by means of the general equation $\nabla\mathscr{L}(\boldsymbol{x},\boldsymbol{\lambda})$.

$$\nabla\mathscr{L}(\boldsymbol{x},\boldsymbol{\lambda}) = \nabla f(\boldsymbol{x}) - \sum_{i}^{\{\mathscr{E}\cup\mathscr{I}\}} \lambda_i \nabla C_i(\boldsymbol{x}) = 0 \tag{2.11a}$$

$$\begin{cases} \lambda_i \in \mathbb{R} & i \in \mathscr{E} \qquad \text{equality constraints} \\ \lambda_i = 0 & i \in \mathscr{I}_I \qquad \text{inequality inactive constraints} \\ \lambda_i \geq 0 & i \in \mathscr{I}_A \qquad \text{inequality active constraints} \end{cases} \tag{2.11b}$$

Or, in another more compact formulation, we can rewrite the conditions on the inequality constraints, observing the symmetry:

$$\left.\begin{array}{lll} \lambda_i \in \mathbb{R} & C_i(\boldsymbol{x}) = 0 & i \in \mathscr{E} \\ \lambda_i = 0 & C_i(\boldsymbol{x}) > 0 & i \in \mathscr{I}_I \\ \lambda_i \geq 0 & C_i(\boldsymbol{x}) = 0 & i \in \mathscr{I}_A \end{array}\right\} \implies \begin{cases} \lambda_i \geq 0,\ C_i(\boldsymbol{x}) \geq 0 & i \in \mathscr{I} \\ \lambda_i C_i(\boldsymbol{x}) = 0 & i \in \mathscr{E}\cup\mathscr{I} \end{cases} \tag{2.12a}$$

This new form introduces the *complementarity condition* $\lambda_i C_i(\boldsymbol{x}) = 0$. Moreover, the complementarity condition is said to be *strict* if, for each i, only one between λ_i and $C_i(\boldsymbol{x})$ is zero.

Aside from this, we have implicitly introduced a function $\mathscr{L}(\boldsymbol{x},\boldsymbol{\lambda})$, that is called *Lagrangian function*, of which we used its derivative. The Lagrangian function plays a crucial role in describing constrained problems also in the mechanics field.

Since we have now defined the situations in which no possible steps are allowed, it would be interesting to express analytically also the *allowed* directions.

Definition 2.6.0.2

Given a feasible point $\boldsymbol{x}$*, the set of* linearized feasible directions *is*

$$\mathscr{F}(\boldsymbol{x}) = \left\{ \boldsymbol{s} \;\middle|\; \begin{array}{ll} \nabla C_i(\boldsymbol{x})^T \boldsymbol{s} = 0 & i \in \mathscr{E} \\ \nabla C_i(\boldsymbol{x})^T \boldsymbol{s} \geq 0 & i \in \mathscr{I}_A \end{array} \right\} \tag{2.13}$$

We clearly see that this set is actually a cone, since the scaled vector $\alpha \boldsymbol{s}$ is still contained in $\mathscr{F}(\boldsymbol{x})$.

2.7 Feasible Directions

In the previous section, we aimed to find an optimal point — or at least a point that is likely to be optimal — making a subtle substitution: the constraints equations and the objective function were approximated, within a certain neighborhood, by their first-order approximations. However, this cannot be taken for granted. In fact, we may have functions (namely, f and/or C_i) whose first-order approximations only *poorly* reflects the original shape of the function. We can say that its first-order approximation does not *qualify* to describe the original function and, insisting to use it may lead to consider *optimal* points that may be not actually feasible, or maybe not stationary. Hence, before proceeding, we have to make sure that the approximation $g(\boldsymbol{x}+\Delta\boldsymbol{x}) \approx g(\boldsymbol{x}) + \nabla g(\boldsymbol{x})^T \Delta\boldsymbol{x}$ holds, at least up to a point.

We then take two sets:

- the linearized set of feasible directions; used in the previous section;
- the actual set of feasible directions.

If the linearized set is contained in the actual set, or is in some way similar, then we can be confident that the assumptions of the previous chapter hold.

Hence, let us take what can be considered a subset of the actual set:

Definition 2.7.0.1
A vector $\boldsymbol{d}$ is said to be tangent to the set Ω at a point $\boldsymbol{x}$ if there exist:

- *a feasible sequence $\{\boldsymbol{z}_k\}$ with $\boldsymbol{z}_k \implies \boldsymbol{x}$*
- *a positive scalar sequence $\{t_k\}$ with $t_k \implies 0$*

such that

$$\lim_{k \implies +\infty} (\boldsymbol{z}_k - \boldsymbol{x}) - t_k \boldsymbol{d} = 0$$

The set of all the vectors $\boldsymbol{d}$ tangent in $\boldsymbol{x}$ is a cone $T_\Omega(\boldsymbol{x})$

As we can see, with few modifications, the definition of tangent vector comes close to the definition of gradient.

From the tangent cone, it stems another definition:

Definition 2.7.0.2
$N_\Omega(\boldsymbol{x})$ is the normal cone to the Ω set at the point $\boldsymbol{x} \in \Omega$ and it is defined as

$$N_\Omega(\boldsymbol{x}) = \left\{ \boldsymbol{v} \mid \boldsymbol{v}^T \boldsymbol{w} \leq 0 \quad \forall \boldsymbol{w} \in T_\Omega(\boldsymbol{x}) \right\}$$

The question that now arises is whether the tangent cone – that is the set of limiting directions of a feasible sequence — can be compared to the set of *linearized* feasible directions of definition 2.6.0.2. If this applies, then the procedure presented in section 2.6 can be used likewise.

Unfortunately, this is not always true: some assumptions on the constraints need to be make. Although there could be a huge variety, one of the most used is the LICQ.

Definition 2.7.0.3
The Linear Independence Constraint Qualification (LICQ) *for the given point $\boldsymbol{x}$ holds if the set of active constraint gradients $\{\nabla C_i(\boldsymbol{x})\}$, with $i \in \mathscr{A}(\boldsymbol{x})$ is linearly independent.*

If this applies, it can be proved that

Lemma 2.7.0.1

- *if $\boldsymbol{x}^*$ is a feasible point, then $T_\Omega(\boldsymbol{x}^*) \subset \mathscr{F}(\boldsymbol{x}^*)$*
- *if $\boldsymbol{x}^*$ is a feasible point and LICQ holds, then $T_\Omega(\boldsymbol{x}^*) = \mathscr{F}(\boldsymbol{x}^*)$*
- *if $\boldsymbol{x}^*$ is a feasible point and the active constraints are linear functions, then $T_\Omega(\boldsymbol{x}^*) = \mathscr{F}(\boldsymbol{x}^*)$*

2.8 Optimality Conditions for Constrained Problems

The previous sections have been presented in order to allow the reader to have a deeper understanding of one of the crucial proposition on the optimality conditions of constrained problems.

Definition 2.8.0.1 (First-Order Necessary Conditions)
Given:

- $\boldsymbol{x}^*$ *a local solution of eq.* (2.2)*;*
- f *and* C_i *are continuously differentiable;*
- *LICQ holds at* $\boldsymbol{x}^*$

then there is a Lagrange multipliers vector $\boldsymbol{\lambda}^*$ *such that*

$$\nabla_x \mathscr{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) = 0 \tag{2.14a}$$
$$C_i(\boldsymbol{x}^*) = 0 \qquad i \in \mathscr{E} \tag{2.14b}$$
$$C_i(\boldsymbol{x}^*) \geq 0 \qquad i \in \mathscr{I} \tag{2.14c}$$
$$\lambda_i^* \geq 0 \qquad i \in \mathscr{I} \tag{2.14d}$$
$$\lambda_i^* C_i(\boldsymbol{x}^*) = 0 \qquad i \in \mathscr{E} \cup \mathscr{I} \tag{2.14e}$$

The optimality conditions above are also called the *Karush-Kuhn-Tucker conditions* or, shortly *KKT*.

There is also a geometric point of view that could come in hand. In fact, assembling a cone $N(\boldsymbol{x}^*)$ from the vectors of the active set at the solution point

$$N(\boldsymbol{x}^*) = \left\{ -\sum_{i \in \mathscr{A}(\boldsymbol{x})^*} \nabla C_i(\boldsymbol{x}) \quad \lambda_i \geq 0,\ i \in \mathscr{I}_A \right\}$$

it can be proven that, by means of the *Farkas lemma* [29, pag.327], either

$$-\nabla f(\boldsymbol{x}^*) \in N(\boldsymbol{x}^*) \quad \Longrightarrow \quad \boldsymbol{d}^T \nabla f(\boldsymbol{x}^*) \geq 0 \quad \forall \boldsymbol{d} \in \mathscr{F}(\boldsymbol{x}^*) \tag{2.15}$$

that means that

$$\nabla f(\boldsymbol{x}^*) = \sum_{i \in \mathscr{A}(\boldsymbol{x}^*)} \lambda_i \nabla C_i(\boldsymbol{x}^*) \quad \lambda_i \geq 0,\ i \in \mathscr{I}_A$$

This is quite similar to what we developed in eq. (2.11).

However, we can also affirm, under some assumptions, that $-\nabla f(\boldsymbol{x}^*) \in N_\Omega(\boldsymbol{x}^*)$. In order to do so, we leverage the following theorem:

Theorem 2.8.0.1
If $\boldsymbol{x}^*$ is a local solution, then

$$\nabla f(\boldsymbol{x}^*)^T \boldsymbol{d} \geq 0 \quad \forall \boldsymbol{d} \in T_\Omega(\boldsymbol{x}^*) \tag{2.16}$$

and we make sure that constraint equations are linearly independent, asking to LICQ to hold. In fact, from lemma 2.7.0.1, it stems that $T_\Omega(\boldsymbol{x}^*) = \mathscr{F}(\boldsymbol{x}^*)$ allowing us to join the results from eq. (2.15) and theorem 2.8.0.1. In these circumstances we can infer that $N(\boldsymbol{x}^*) = N_\Omega(\boldsymbol{x}^*)$ and so that

$$-\nabla f(\boldsymbol{x}^*) \in N_\Omega(\boldsymbol{x}^*)$$

In any case, the conditions on the first derivative are not sufficient to prove the optimality of a solution. For this, also the second derivative should be taken into account. Its role is critical as it was for the unconstrained case: it assesses the increase or decrease of the objective function in the neighborhood of a given point. Since our problem is in quadratic form, interesting considerations can be extrapolated from the following discussion.

Assume to have a pair of variables $\boldsymbol{x}^*$ and $\boldsymbol{\lambda}^*$ for which the KKT conditions apply, we now define a set $\mathscr{C}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) \in \mathscr{F}(\boldsymbol{x}^*)$ such that:

$$\boldsymbol{w} \in \mathscr{C}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) \quad \Longleftrightarrow \quad \begin{cases} \nabla C_i(\boldsymbol{x}^*)^T \boldsymbol{w} = 0 & \forall i \in \mathscr{E} \\ \nabla C_i(\boldsymbol{x}^*)^T \boldsymbol{w} = 0 & \forall i \in \mathscr{I}_A \text{ with } \lambda_i^* > 0 \\ \nabla C_i(\boldsymbol{x}^*)^T \boldsymbol{w} \geq 0 & \forall i \in \mathscr{I}_A \text{ with } \lambda_i^* = 0 \end{cases}$$

This set is called *critical cone* in $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)$ and, from the first and third requirements above, we can state that is contained in the linearized feasible directions set $\mathscr{F}(\boldsymbol{x}^*)$.

As we can see the particular case of *active* constraints (i.e. with $C_i(x^*) = 0$) with $\lambda_i^* = 0$ has been excluded: this requirement is called *strict complementarity*. The critical cone lists all the directions for which the constraints will *preserve* their state. In another form, we can rewrite the previous equation in a more compact form:

$$\lambda_i^* \nabla C_i(\boldsymbol{x}^*)^T \boldsymbol{w} = 0 \quad \forall i \in \mathscr{I} \cup \mathscr{E} \tag{2.17}$$

Since it is $\lambda_i^* = 0$ for inactive constraints.

The second-order conditions are fundamental since they assure the connection between the Karush-Kuhn-Tucker conditions and the optimal solution.

Definition 2.8.0.2 (Second-Order Necessary Conditions)
Given:

- $\boldsymbol{x}^*$ *solution of the constrained problem 2.2;*
- $\boldsymbol{\lambda}^*$ *the Lagrangian multipliers for which the KKT conditions hold;*
- *that LICQ holds.*

Then:

$$\boldsymbol{w}^T \nabla^2_{xx} \mathscr{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) \boldsymbol{w} \geq 0 \quad \forall \boldsymbol{w} \in \mathscr{C}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) \tag{2.18}$$

That is: the curvature along the $\boldsymbol{w}$ *direction is non-negative.*

However, the most interesting considerations are made on the sufficient conditions; based on them, given the hypothesis, we could check if the given value of the variables is solution of the problem.

Definition 2.8.0.3 (Second-Order Sufficient Conditions)
Given $\boldsymbol{x}^*$ *solution of the constrained problem and* $\boldsymbol{\lambda}^*$ *the Lagrangian multipliers for which the KKT conditions hold and*

$$\boldsymbol{w}^T \nabla^2_{xx} \mathscr{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) \boldsymbol{w} > 0 \quad \forall \boldsymbol{w} \in \mathscr{C}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) \setminus 0 \tag{2.19}$$

Then $\boldsymbol{x}^*$ *is a* strict *local solution for the constrained problem 2.2.*

This will come in hand when we will observe the quadratic programming problem, where $\nabla^2_{xx}\mathscr{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)$ will be particularly easy to find.

As we can see, in the sufficient conditions there is no requirement on the constraint qualification. However, they could be useful in order to reformulate the theorem in a more useful form. Let denote $\{\nabla C_i\}_{i\in\mathscr{A}(\boldsymbol{x}^*)}$ with $A(\boldsymbol{x}^*)$ and assume that:

- the KKT solution must be unique;
- strict complementarity holds, so $A(\boldsymbol{x}^*)\boldsymbol{w} = 0 \Leftrightarrow \boldsymbol{w} = 0$.

In this case the critical cone becomes:

$$\mathscr{C}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) = \text{null}\,(A(\boldsymbol{x}^*)) \tag{2.20}$$

so that there will be a square matrix Z with the same dimension of the rows of $A(\boldsymbol{x}^*)$ that will span the critical cone. In this case, each vector $\boldsymbol{w} \in \mathscr{C}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)$ can be rewritten as $Z\boldsymbol{u}$ and eq. (2.18) as

$$Z^T \nabla^2_{xx}\mathscr{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)Z \geq 0$$

and eq. (2.19) as

$$Z^T \nabla^2_{xx}\mathscr{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)Z > 0$$

The $\geq$ and $>$ operator, when applied to matrices, are intended to express their (semi)positive definiteness.

2.9 Duality

The duality theory propose a specular problem to the one proposed in eq. (2.2) — the latter called *primal* problem — that could be used with the purpose of finding a lower bound to the objective function at the solution or, as in this case, to design a solving algorithm.

The duality theory starts with the definition of the *Lagrangian Dual Function*

$$q(\boldsymbol{\lambda}) \stackrel{\text{def}}{=} \inf_{\boldsymbol{x}} \mathscr{L}(\boldsymbol{x}, \boldsymbol{\lambda})$$

It can be proven that the function $q(\boldsymbol{\lambda})$ is concave and its domain, that is $\{\boldsymbol{\lambda} \mid q(\boldsymbol{\lambda}) > -\infty\}$ is convex. However, finding the infimum is still as difficult as finding the minimum of the objective function of the primal problem. The convexity of the Lagrangian function is only partially helpful.

Expanding the terms of the previous equation it is clear that, being $\lambda_i \geq 0$ when $C_i(\boldsymbol{x}) \geq 0$, $i \in \mathscr{I}$ and being $\lambda_i = 0$ when $i \in \mathscr{E}$, the Lagrangian function has the following property

$$q(\boldsymbol{\lambda}) = \inf f(\boldsymbol{x}) - \lambda^T C(\boldsymbol{x}) \leq f(\boldsymbol{x}) - \lambda^T C(\boldsymbol{x}) \leq f(\boldsymbol{x})$$

whenever $\boldsymbol{x}$ a feasible point. Thus, it can be said that the Lagrangian Dual Function represents a lower bound to the objective function, for any feasible point. This result, called *weak duality* always holds, even for non-convex f and C.

Since the objective function of the primal problem is *lower*-bounded by the Lagrangian Dual Function, it is clear that we are interested in searching for a value $\hat{\boldsymbol{\lambda}}$ for which q attains its maximum, in order to push this bound on the primal objective as high as possible: this is called the *dual* problem.

$$\begin{aligned} \max \quad & q(\boldsymbol{\lambda}) && && \text{(2.21a)} \\ \text{s.t.} \quad & \lambda_i = 0 && i \in \mathscr{E} && \text{(2.21b)} \\ & \lambda_i \geq 0 && i \in \mathscr{I} && \text{(2.21c)} \end{aligned}$$

In general, it has been said that $q(\boldsymbol{\lambda}) \leq f(\boldsymbol{x})$: this means that we may identify a *gap* between the two functions, primal and dual. Moreover, the difference between such functions at the optimal points $\hat{\boldsymbol{\lambda}}$ for q and $\bar{\boldsymbol{x}}$ for f is named *duality gap*. Surely, the inequality still holds:

$$q(\hat{\boldsymbol{\lambda}}) \leq f(\bar{\boldsymbol{x}}) \tag{2.22}$$

However, in particular cases,

Theorem 2.9.0.1
Given that:

- *f and $-C$ are convex and continuously differentiable on $\mathbb{R}^n$;*

- $\bar{\boldsymbol{x}}$ *is solution of primal problem 2.2;*
- *LICQ holds at* $\bar{\boldsymbol{x}}$
- $\hat{\boldsymbol{\lambda}}$ *is solution of the dual problem 2.21 at the point* $\hat{\boldsymbol{x}}$*;*
- $\mathscr{L}(\cdot, \hat{\boldsymbol{\lambda}})$ *is strictly convex.*

then $\bar{\boldsymbol{x}} = \hat{\boldsymbol{x}}$ *and* $f(\bar{\boldsymbol{x}}) = \mathscr{L}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{\lambda}}) = q(\hat{\boldsymbol{\lambda}})$

2.10 Quadratic Programming

While, in the previous chapter, we faced generic problems with constraints, we will now restrict ourselves to a specific subset of problems with quadratic objective functions and linear constraints. We will see how the multibody problems faced in chapter 3 can be fit into this form.

In general, we can build the following problem:

$$\begin{aligned} \min \quad & \frac{1}{2}\boldsymbol{v}^T H \boldsymbol{v} + \boldsymbol{v}^T \boldsymbol{c} && \text{(2.23a)} \\ \text{s.t.} \quad & C\boldsymbol{v} - \boldsymbol{b}_\gamma = \boldsymbol{0} && \text{(2.23b)} \\ & D\boldsymbol{v} - \boldsymbol{b}_\lambda \geq \boldsymbol{0} && \text{(2.23c)} \end{aligned}$$

where both equality and inequalities are present. Or again, moving the non-negative constraint on a single variable, we can write equivalently

$$\begin{aligned} \min \quad & \frac{1}{2}\boldsymbol{v}^T H \boldsymbol{v} + \boldsymbol{v}^T \boldsymbol{c} && \text{(2.24a)} \\ \text{s.t.} \quad & C\boldsymbol{v} - \boldsymbol{b}_\gamma = \boldsymbol{0} && \text{(2.24b)} \\ & D\boldsymbol{v} - \boldsymbol{b}_\lambda - \boldsymbol{y} = \boldsymbol{0} && \text{(2.24c)} \\ & \boldsymbol{y} \geq \boldsymbol{0} && \text{(2.24d)} \end{aligned}$$

The dimensions of matrices can be deduced by those of the vectors $\boldsymbol{v} \in \mathbb{R}^{n_v}$, $\boldsymbol{b}_\gamma \in \mathbb{R}^{n_\gamma}$ and $\boldsymbol{b}_\lambda \in \mathbb{R}^{n_\lambda}$. As we can see, the objective function is quadratic and the constraints are linear, even though they contain both equalities and inequalities. Being linear

means also that they are at the same time convex and concave: this means that they are eligible to be part of a convex programming problem (2.3). However, in order to declare the whole problem as convex also the objective function should be so. In this case, its convexity depends on the positive definiteness of the the matrix H. In case $H \geq 0$ then the Quadratic Program (QP) can also be said *convex*, and all the considerations of the section 2.3 and following apply. In particular, there is the assurance that the local solution is also global, even without any further constraint qualification [29, pag.465]. It is also unique if H is positive definite. Second-order sufficient conditions 2.19 can be also be specialized for the QP case, simply considering that $\mathscr{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) = H$.

Moreover, if we set VI problem with $\boldsymbol{F}(\boldsymbol{v}) = N\boldsymbol{\lambda} + \boldsymbol{r}$:

$$\boldsymbol{\lambda} \in \mathbb{R}^{n_\lambda}_+ \quad : \quad \langle N\boldsymbol{\lambda} + \boldsymbol{r}, \boldsymbol{\lambda} - \boldsymbol{y} \rangle \geq 0 \quad \forall \boldsymbol{y} \in \mathbb{R}^{n_\lambda}_+ \tag{2.25a}$$

$$N = DHD^T \tag{2.25b}$$

$$\boldsymbol{r} = -DH^{-1}\boldsymbol{c} - \boldsymbol{b}_\lambda \tag{2.25c}$$

$$\boldsymbol{v} = H^{-1}(D^T\boldsymbol{\lambda} - \boldsymbol{c}) \tag{2.25d}$$

the resulting VI problem is exactly a QP problem and thus, also equivalent to a LCP:

$$\mathbf{0} \leq N\boldsymbol{\lambda} + \boldsymbol{r} \quad \perp \quad \boldsymbol{\lambda} \geq \mathbf{0}, \tag{2.26}$$

2.11 Cone Quadratic Programming

The standard QP formulation allows to model a wide range of problems, such as for our interest, *frictionless* contacts.

However, it may be interesting to expand the problem in order to take into account also friction and, in particular, the Coulomb Friction Law. Without going into details (we will in section 3.2.2) we can say that the Coulomb Law introduces a different constraint respect to the classic Quadratic Programming problems: instead of belonging to an half-space, the reaction (friction) force is now forced to lie into a

friction cone with apex angle $2\arctan(\mu)$. For a single contact, this can be written as:

$$\boldsymbol{\lambda} \in \mathscr{K}_\mu, \quad \mathscr{K}_\mu = \left\{ \begin{bmatrix} \boldsymbol{\lambda}_{[0]} \\ \boldsymbol{\lambda}_{[1]} \end{bmatrix} \in \mathbb{R} \times \mathbb{R}^{p-1} \mid \mu \boldsymbol{\lambda}_{[0]} \geq \|\boldsymbol{\lambda}_{[1]}\| \right\}$$

We clearly see a significant difference with the QP case in which the constraint, for a single contact, affected just one scalar value (e.g. $\lambda_i \geq 0$) while here, supposing to work in the $\mathbb{R}^3$ space, each individual contact is modeled by a triplet of equations and variables, and translates into the normal and tangential reaction forces.

The Cone QP will be written as

$$\begin{aligned} \min \quad & \frac{1}{2}\boldsymbol{v}^T H \boldsymbol{v} + \boldsymbol{v}^T \boldsymbol{c} && \text{(2.27a)} \\ \text{s.t.} \quad & C\boldsymbol{v} - \boldsymbol{b}_\gamma = \boldsymbol{0} && \text{(2.27b)} \\ & D\boldsymbol{v} - \boldsymbol{b}_\lambda - \boldsymbol{y} = \boldsymbol{0} && \text{(2.27c)} \\ & \boldsymbol{y} \succeq \boldsymbol{0} && \text{(2.27d)} \end{aligned}$$

Chapter 3

Multibody Systems

The ability to simulate mechanical systems before their realization deeply affected the way designers and engineers work. Not only this, but many machines and structures that are nowadays considered possible, only few decades ago were relegated to the inventors' imagination, because of the uncertainties about their behavior or resistance. The scientific discoveries that permitted this incredible accomplishments were possible thanks to the interested exerted by military (in the first part) and industries (in the second half of the last century) on the scientific knowledge, feeding the research with greater tasks and greater funds.

From the 70s, after many centuries of pen and paper, a new tool became in use in the scientific and technical world: the computer. Thanks to this new equipment, the computational power suddenly available opened the doors to new investments and new possibilities. The great explorations of space and sky, among the others. Since then, the interest on mechanical simulation never dropped, and together with it, the race for more powerful computing tools.

In this context, the current research is run in order to provide new and better solutions for the simulation of multibody systems; a branch that aspire to encompass structural analysis, finite-element analysis, collision-detection into a unique engineering tool.

One of the most critical topic in multibody dynamics is the non-smoothness of the

events that evolve during the simulations, mainly contacts. These phenomena break the continuous and smooth nature of the motion of the bodies and ask for a different modeling, in order to be taken into account and fit into the dynamic simulation framework.

One of the most used technique to solve this class of problems is the regularization approach, through which the discontinuities undergo a regularization treatment that reckon on the introduction of an artificial stiffness between contacting bodies. Although frequently in use in many Discrete and sometimes Finite Element softwares it brings few undesired consequences. First of all, rigid contacts cannot be pursued: the regularization relies on artificial stiffness between the point in contact and its value cannot clearly be infinite. Moreover, even if an arbitrary high value is chosen, the simulation has to be tuned accordingly: in fact, the increased stiffness of some components — i.e. the contact links — can generate impulsive events with consequential generation of high-valued forces and acceleration that need to be caught and handled inside the dynamical motion system. This can be possible only if shorter time steps are reached, since otherwise instabilities can halt the simulation. Second, it is clear that the artificial stiffness is a parameter that is only weakly linked to the surfaces' material. Most of the time an empirical coefficient must be adopted, causing the simulation to heavily depend on this choice.

Because of this, a different numerical scheme has been adopted, in order to be able to enclose contacts and joints into an unified formulation together with the body dynamics, going further the classical ODE/DAE systems, that would have required multiple piecewise integrations for each change of active constraints.

The variational approach seemed the most appropriate to this extent, and clearly forms a perfect pair for the application of our Interior-Point algorithm. We will show how this two different topics and formulations can be seen as one. By mean of Differential Variational Inequalities we are going to express the motion of the bodies together with contact formulations in the same framework, assembling what is referred to as Cone Complementarity Problem. However, in order to accomplish this we are going to first introduce LCPs and their application to contacts with no friction.

3.1 System State

In the absolute coordinate system the rigid bodies are represented by their barycentric position via the translational vector $\boldsymbol{x}_i \in \mathbb{R}^3$ with i the body index; its orientation is expressed by unit quaternions $\boldsymbol{\rho}_i(t) \in \mathbb{H}^1$ that introduce an exceeding parameter, so that the configuration of a single body in the $\boldsymbol{SO}(\mathbb{R},3)$ space becomes $\boldsymbol{q}_i(t) = \{\boldsymbol{x}_i, \boldsymbol{\rho}_i\} \in \mathbb{R}^7$. The fourth component of the quaternion is clearly redundant, but the overall number of degrees of freedom is actually still equal to six, because of the unity-norm constraint $\|\boldsymbol{\rho}_i(t)\| = 1$.

Given the particular composition of the *position* vector $\boldsymbol{q}_i(t)$, the *velocity* vector $\boldsymbol{v}_i$ has a different size: the translational speed is the classical derivative $\dot{\boldsymbol{x}}_i \in \mathbb{R}^3$, while the angular velocity is expressed, in the local reference frame, by the vector $\boldsymbol{\omega}_i \in \mathbb{R}^3$. Then the the velocity vector is $\boldsymbol{v}_i = \{\dot{\boldsymbol{x}}_i, \boldsymbol{\omega}_i\}$.

We provide here two main operations on the position vector that come in hand for the future development: derivation and integration. The position vector $\boldsymbol{x}_i$ is directly derived into the component $\dot{\boldsymbol{x}}_i$ of $\boldsymbol{v}_i$, but a different perspective should be taken for $\dot{\boldsymbol{\rho}}_i$ to use the angular velocity $\boldsymbol{\omega}_i$. The quaternion multiplication comes in hand:

$$\dot{\boldsymbol{\rho}}_i = \frac{1}{2}\boldsymbol{\rho}_i \begin{bmatrix} 0 \\ \boldsymbol{\omega}_i \end{bmatrix}$$

where the (Hamiltonian) multiplication of such numbers is considered as the following:

$$\boldsymbol{qp} = \begin{bmatrix} q_0 p_0 - \boldsymbol{q}_{1:3}^T \boldsymbol{p}_{1:3} \\ q_0 \boldsymbol{p}_{1:3} + p_0 \boldsymbol{q}_{1:3} - \boldsymbol{q}_{1:3} \times \boldsymbol{p}_{1:3} \end{bmatrix}$$

In general, the derivative of the position vector is obtained through a linear map

$$\dot{\boldsymbol{\rho}}_i = \Gamma(\boldsymbol{q}_i, \boldsymbol{v}_i) = \begin{bmatrix} \dot{\boldsymbol{x}}_i \\ \frac{1}{2}\boldsymbol{\rho}_i \begin{bmatrix} 0 \\ \boldsymbol{\omega}_i \end{bmatrix} \end{bmatrix}$$

The integration of the position vector could be quite evaluated through different integration schemes. We may point out that, even with the simplest explicit Euler integration, a particular caution should be taken in order to preserve the unity-norm

constraint over the quaternions. In fact, if the update of $\boldsymbol{x}_i(t+\Delta t) = \boldsymbol{x}_i(t) + \dot{\boldsymbol{x}}_i(t)\Delta t$ is quite straightforward, more attention should be payed for $\boldsymbol{\rho}_i(t+\Delta t) = \boldsymbol{\rho}_i(t) + \dot{\boldsymbol{\rho}}_i(t)\Delta t$. This operation would take the new vector $\|\boldsymbol{\rho}_i(t+\Delta t)\| \neq 1$ and, even with a proper re-scaling to restore the unitarian norm, the updated direction would be not exactly the one expected.

The best option is to leverage the intrinsic properties of quaternions. Given a unit vector axis $\boldsymbol{n}$ and an angle α we can describe the rotation of the local/rotated frame respect to the fixed frame [30] using an equivalent quaternion

$$\boldsymbol{\rho} = \begin{bmatrix} \cos(\frac{\alpha}{2}) \\ \boldsymbol{n}\sin(\frac{\alpha}{2}) \end{bmatrix}$$

Considering the angular velocity $\boldsymbol{\omega}_i$ it is clear that in a time step Δt the angular displacement is $\alpha = \|\boldsymbol{\omega}_i\|\Delta t$ around the axis $\boldsymbol{n} = \frac{\boldsymbol{\omega}_i}{\|\boldsymbol{\omega}_i\|}$, so the new position of the local frame can be computed via quaternions multiplications

$$\boldsymbol{\rho}_i(t+\Delta t) = \boldsymbol{\rho}_i(t)\Delta\boldsymbol{\rho}_i(t) = \boldsymbol{\rho}_i(t) \begin{bmatrix} \cos(\frac{1}{2}\|\boldsymbol{\omega}_i\|\Delta t) \\ \frac{\boldsymbol{\omega}_i}{\|\boldsymbol{\omega}_i\|}\sin(\frac{1}{2}\|\boldsymbol{\omega}_i\|\Delta t) \end{bmatrix} \tag{3.1}$$

In some cases multiple updates are needed at the single time step (e.g. in Runge-Kutta integrators). Fortunately, we can concatenate the equation.

The properties of each body are considered to be the mass matrix M with diagonal elements equal to m_i and the inertia's tensor $I_i \in \mathbb{R}^{3x3}$.

3.2 Constraints

The relation between bodies involved in the simulation is expressed by means of constraint equations (that we already saw in eq. (2.2) with the generic symbol $C_i(\boldsymbol{x})$)

Equalities from bilateral joints that are expressed by plain equations. In this category we find holonomic constraints that describe revolute and prismatic joints, glyphs, etc... written using relative body position;

Inequalities from contacts within bodies.

Each joint may be mapped by multiple (in)equalities depending on its type e.g. spherical joints require three equalities.

For sake of readability and compactness of this chapter, we will now diversify the generic constraint function symbol C_i for bilateral and unilateral constraints.

For the bilateral case, the equations come in the form

$$\Psi_i(\boldsymbol{q},t) = 0 \quad (i \in \mathscr{E}) \tag{3.2}$$

where the time argument is included in order to take into account rheonomic constraints that may also describe trajectory or motion impositions, such those applied by actuators/motors. The set $\mathscr{E}$ lists the bilateral constraints present in the system at the current time of the simulation.

The equation for the contacts are in the form

$$\Phi_i(\boldsymbol{q}) \geq 0 \quad (i \in \mathscr{I}) \tag{3.3}$$

Rigorously, these functions do not depend only on the position of the bodies, but also on their shapes. In fact, the equations that describe the contacts should relate two points belonging to different points (or event the same body if flexible) that are prone to collide in the next time step and these depend on the conformation of their surface.

Clearly, Φ_j can be just a signed distance function [31] between the contact points measured along a normal to the surfaces, as long as it takes into account also penetration (negative values). This task conceals the challenging objective [11, 32] to find such pair of points that are more likely to come in contact. For our purpose, the open-source library Chrono::Engine, used in the simulation, goes through a three-step collision detection routine

Broad-Phase using Sweep And Prune (SAP) algorithm [33]; avoids to check collision between shapes that are far away;

Middle-Phase using AABB in order to check concave shapes collisions;

Narrow-Phase using Gilbert-Johnson-Keerthi (GJK) algorithm [34];

The shapes are surrounded by an offset surface obtained by Minkowski sums in order to allow an acceptable inter-penetration.

3.2.1 Jacobian of the Constraint Equations

In a constrained systems, links exchange reactions with the bodies to which are connected, influencing their motions and even their shape, if flexible. Constraints equations are the key elements that allow this connection, but, as one can notice, they does not translate immediately to forces; they would rather describe position relations. The Lagrangian function is then used to relate the constraints to the dynamic of the bodies. One may notice that also in the Karush-Kuhn-Tucker conditions (eq. (2.14)) the same equation came in hand.

In order to write the KKT conditions or, alternatively, the equations of constrained motion, we are asked to provide the Jacobian of the Constraints Equations. Using following the differentiation rule:

$$\begin{aligned}\frac{dC_j(\boldsymbol{q},t)}{dt} &= \frac{\partial C_j}{\partial \boldsymbol{q}}\frac{d\boldsymbol{q}}{dt} + \frac{\partial C_j}{\partial t} \\ &= \frac{\partial C_j}{\partial \boldsymbol{q}}\Gamma(\boldsymbol{q})\boldsymbol{v} + \frac{\partial C_j}{\partial t}\end{aligned}$$

We recall that

- we assumed the differentiability of the function $C_j(\boldsymbol{q},t)$;
- the $\Gamma(\boldsymbol{q})$ linear map is used to translate from from angular velocities to quaternion derivatives, while keeping untouched the translational degrees of freedom, thus the matrix is just a composition of diagonal blocks matrices plus 4x3 blocks associated to the quaternion rates transformations;
- since their main role, we will declare two specific symbols for the gradient/Jacobian of bilateral and unilateral constraints:

$$\begin{aligned}C &\equiv \begin{bmatrix}\dots & \frac{\partial \Psi_j}{\partial \boldsymbol{q}}\Gamma(\boldsymbol{q}) & \dots\end{bmatrix} \\ D &\equiv \begin{bmatrix}\dots & \frac{\partial \Phi_j}{\partial \boldsymbol{q}}\Gamma(\boldsymbol{q}) & \dots\end{bmatrix}\end{aligned}$$

Each set of constraints is strictly related to a Lagrangian multiplier ($\boldsymbol{\gamma}$ or $\boldsymbol{\lambda}$) that expresses the intensity of the reaction force that the joint exchange. Depending on the constraint type certain assumption on the sign of the Lagrangian multipliers has to be made (see section 2.5).

3.2.2 Contacts

The collision detection algorithm is the very first tile of the contact simulation procedure. Its task is to catch those pairs of bodies that may come in contact in the very next time steps, informing in advance the dynamic model so that it can promptly introduce the required contact forces.

Thus, only between pair of bodies that are close enough, depending on a distance threshold, the unilateral constraint is added, reducing the overall computational effort. It is now clear that the set of constraints and their relative Jacobian are in continuous change.

The equation that is added, is in the known form $\Phi_i(\boldsymbol{q}) \geq 0$. However, this equation presents some critical features, like the non-differentiability for sharp edges or the occasional negativity that occurs simply because rounding errors and numerical inaccuracies. This are only some of the issues that arise in the collision detection process and that can be handled effectively.

Each body of the pair holds a nearest point to the other body shape, for which a local coordinate system may be introduced $\{\boldsymbol{t}_{n,i}, \boldsymbol{t}_{u,i}, \boldsymbol{t}_{v,i}\} \in \mathbb{R}^3$ that are aligned with the normal to the surface ($\boldsymbol{t}_{n,i}$) and along the tangent plane ($\{\boldsymbol{t}_{u,i}, \boldsymbol{t}_{v,i}\}$). Along these axes the relevant contact informations are described.

Axis	*Forces*	*Velocities*
$\boldsymbol{t}_{n,i}$	$\boldsymbol{F}_{n,i}$	$\hat{\boldsymbol{v}}_{n,i}$
$\boldsymbol{t}_{u,i}$	$\boldsymbol{F}_{u,i}$	$\hat{\boldsymbol{v}}_{u,i}$
$\boldsymbol{t}_{v,i}$	$\boldsymbol{F}_{v,i}$	$\hat{\boldsymbol{v}}_{v,i}$

in particular we have the contact forces:

$$\begin{aligned} \boldsymbol{F}_i &= \boldsymbol{F}_{n,i} + \boldsymbol{F}_{\|,i} && (3.4)\\ &= \boldsymbol{F}_{n,i} + \boldsymbol{F}_{u,i} + \boldsymbol{F}_{v,i} && (3.5)\\ &= \widehat{\lambda}_{n,i}\boldsymbol{t}_{n,i} + \widehat{\lambda}_{u,i}\boldsymbol{t}_{u,i} + \widehat{\lambda}_{v,i}\boldsymbol{t}_{v,i} && (3.6) \end{aligned}$$

where $\boldsymbol{F}_{\|,i}$ is given by the friction reactions. And the consequents velocities at the contact point:

$$\begin{aligned} \hat{\boldsymbol{v}}_i &= \hat{\boldsymbol{v}}_{n,i} + \hat{\boldsymbol{v}}_{\|,i} && (3.7)\\ &= \hat{\boldsymbol{v}}_{n,i} + \hat{\boldsymbol{v}}_{u,i} + \hat{\boldsymbol{v}}_{v,i} && (3.8)\\ &= u_{n,i}\boldsymbol{t}_{n,i} + u_{u,i}\boldsymbol{t}_{u,i} + u_{v,i}\boldsymbol{t}_{v,i} && (3.9)\\ &= (\boldsymbol{D}_{n,i}^T\boldsymbol{v})\boldsymbol{t}_{n,i} + (\boldsymbol{D}_{u,i}^T\boldsymbol{v})\boldsymbol{t}_{u,i} + (\boldsymbol{D}_{v,i}^T\boldsymbol{v})\boldsymbol{t}_{v,i} && (3.10) \end{aligned}$$

and $D_i = \begin{bmatrix} \boldsymbol{D}_{n,i} & \boldsymbol{D}_{u,i} & \boldsymbol{D}_{v,i} \end{bmatrix}$ is the Jacobian that relates them to the generalized velocities $\boldsymbol{v}$.

The Coulomb Friction Law imposes now two conditions

- Restriction on the tangential friction force

$$\mu\widehat{\lambda}_{n,i} \geq \sqrt{\widehat{\lambda}_{u,i}^2 + \widehat{\lambda}_{v,i}^2} \tag{3.11}$$

  where it can be clearly seen as a conic constraint $\boldsymbol{\lambda}_i \in \mathscr{K}_\mu$ where μ is the friction coefficient.

- Direction of tangential force $\boldsymbol{F}_{\|,i}$ opposite to the tangential velocity $\hat{\boldsymbol{v}}_{\|,i}$

$$\left\langle \boldsymbol{F}_{\|,i}, \hat{\boldsymbol{v}}_{\|,i} \right\rangle = -\|\boldsymbol{F}_{\|,i}\|\|\hat{\boldsymbol{v}}_{\|,i}\| \tag{3.12}$$

If we consider now eq. (3.11), eq. (3.12) together with the required condition for unilateral contacts

$$0 \leq \Phi_i(\boldsymbol{q}) \perp \widehat{\lambda}_{n,i} \geq 0$$

it is easy to identify an optimization problem

$$\min \quad \left(\widehat{\lambda}_{u,i}\,\boldsymbol{t}_{u,i}+\widehat{\lambda}_{v,i}\,\boldsymbol{t}_{v,i}\right)^T \boldsymbol{v}_{\|,i} \tag{3.13}$$

$$s.t. \quad \widehat{\boldsymbol{\lambda}}_i \in \Upsilon_{\mathscr{J},i} \tag{3.14}$$

$$0 \leq \widehat{\lambda}_{n,i} \perp \Phi_i(\boldsymbol{q}) \geq 0 \tag{3.15}$$

in which we recognize the maximum dissipation principle [13,19,35]. Remembering that

$$\dot{\Phi}_i(\boldsymbol{q}) = u_{n,i} = \boldsymbol{D}_{n,i}^T \boldsymbol{v} \tag{3.16}$$

one can restate the Signorini condition at the velocity level

$$0 \leq \widehat{\lambda}_{n,i} \perp \dot{\Phi}_i(\boldsymbol{q}) \geq 0$$

As a last step, one may recognize that with an additional change, the problem is also a Cone Complementarity Problem. In fact, while the $\widehat{\boldsymbol{\lambda}}$ vector is already fit to belong to a Lorentz cone ($\Upsilon_{\mathscr{J},i}$), the $\dot{\Phi}_i(\boldsymbol{q})$ vector, as it is, is not contained in its dual $\Upsilon^*_{\mathscr{J},i}$. Hence, we may add a corrected term $\bar{\boldsymbol{u}}_i$ formed as

$$\begin{aligned}
\bar{\boldsymbol{u}}_i &= \left\{ \begin{array}{l} u_{n,i} + \mu_i \sqrt{u_{u,i}^2 + u_{v,i}^2} = u_{n,i} + \mu_i \|\mathbf{v}_{\|,i}\| \\ u_{u,i} \\ u_{v,i} \end{array} \right\} \quad = \left\{ \begin{array}{l} u_{n,i} + \mu_i \|\mathbf{v}_{\|,i}\| \\ u_{u,i} \\ u_{v,i} \end{array} \right\} \\
&= D_i^T \boldsymbol{v} + \left\{ \begin{array}{l} \mu_i \|D_{\|,i}^T \boldsymbol{v}\| \\ 0 \\ 0 \end{array} \right\} \\
&= \boldsymbol{u}_i + \tilde{\boldsymbol{u}}_i
\end{aligned}$$

Unfortunately, the introduction of the term $\tilde{\boldsymbol{u}}_i$ is necessary in order to fit the velocity vector in the dual cone $\Upsilon^*_{\mathscr{J},i}$, but it comes with a price: it is not-linear and not-differentiable, and thus will be $\bar{\boldsymbol{u}}_i$.

3.3 DVI/MDI Dynamic Model

In previous sections we addressed the analytical models of the bodies (section 3.1) and the constraints (section 3.2), even in the case of contacts (section 3.2.2). The next task is to describe the law that describes their mutual influence and permits the combined simulation of objects together with their joints and contacts.

The classic dynamic system is the starting point of the formulation, but it has to undergo a long list of changes, in order to accommodate some particular features:

- we would like to summarize all the different types of constraints in a unique formulation:
  - the gradient of bilateral constraints can be put in the degenerate cone $\{0\}$ since no velocities are allowed for the constrained bodies and, on the opposite its dual cone i.e. where the Lagrangian multipliers reside, is the entire $\mathbb{R}$ space;
  - only active constraints should be put in the system, so that we take only $i \in \mathscr{I}_A \iff \{i \in \mathscr{I} \mid \Phi_i = 0\}$.
- we would like to describe impulsive events, like contacts, and the related reaction forces that follows; hence, the system has to deal with non-smooth formulations that permit discontinuities:

  Velocity during a collision, a body must be able to suddenly variate its velocity; e.g. an elastic bouncing ball: its speed switches sign instantaneously. In order to describe this phenomena *functions of Bounded Variations (BV)* are used;

  Acceleration since velocities have jumps, accelerations are described through *distributions of Radon measures* in order to be able to follow their discontinuity;

  Position given the time integration over the velocity, the position assumes *Absolute Continuity*;

Forces in order to provoke instantaneous change of speed, we should discuss about impulses and not only standard forces. To do this we are going to split the formulation in a continuous and impulsive part.

The following DVI for *smooth* systems

$$M\frac{d\boldsymbol{v}}{dt} = \boldsymbol{f}(\boldsymbol{q},\boldsymbol{v},t) + D^T\widehat{\boldsymbol{\lambda}}(t) + C^T\widehat{\boldsymbol{\gamma}}(t) \tag{3.17a}$$

$$\Upsilon_{\mathscr{I},i} \ni \widehat{\boldsymbol{\lambda}}_i \perp \bar{\boldsymbol{u}}_i \in \Upsilon^*_{\mathscr{I},i} \quad \forall i \in \mathscr{I}_A \tag{3.17b}$$

$$\widehat{\boldsymbol{\lambda}}_i = \boldsymbol{0} \quad \forall i \in \mathscr{I} \cap \mathscr{I}_A \tag{3.17c}$$

$$\Psi(\boldsymbol{q},t) = 0 \quad \forall i \in \mathscr{E} \tag{3.17d}$$

$$\dot{\boldsymbol{q}} = \Gamma(\boldsymbol{q})\boldsymbol{v} \tag{3.17e}$$

where the f term includes gravitational, gyroscopic and all external forces. We recall that $\mathscr{I}_A$ is the set of active unilateral constraints i.e. for which $\Phi_i > 0$. The matrices C and D contain the Jacobian of the respectively bilateral and unilateral constraints.

The system can be restated, leveraging the fact that all the constraints can be described as conic complementarities, into the more compact form [20]

$$M\frac{d\boldsymbol{v}}{dt} = \boldsymbol{f}(\boldsymbol{q},\boldsymbol{v},t) + D_{\mathscr{C}}\widehat{\boldsymbol{\lambda}}_{\mathscr{C}} \tag{3.18a}$$

$$\Upsilon_{\mathscr{C}} \ni \widehat{\boldsymbol{\lambda}}_{\mathscr{C}} \perp \bar{\boldsymbol{u}}_{\mathscr{C}} \in \Upsilon^*_{\mathscr{C}} \tag{3.18b}$$

$$\dot{\boldsymbol{q}} = \Gamma(\boldsymbol{q})\boldsymbol{v} \tag{3.18c}$$

where the $\Upsilon_{\mathscr{C}}$ merges all the cone constraints for both unilateral and bilateral cases and $\widehat{\boldsymbol{\lambda}}_{\mathscr{C}}$ considers also $\widehat{\boldsymbol{\gamma}}$. For a more detailed description of the required steps see [36].

The second step towards an MDI implementation, is motivated by the fact that in the previous model no impulsive events can be effectively handled: for this accomplishment is required to import *measures* into this framework.

Thanks to the *Lebesgue decomposition theorem* we can state that a measure ν can be decomposed into

$$\nu = \nu_c + \nu_d + \nu_{sc} \tag{3.19}$$

where

dt is the Lebesgue measure on whose base the statement regarding the continuity or singularity are referred;

ν_c is the absolutely continuous part; e.g. in our system, the speed variation $\boldsymbol{a}dt$ or the impulse caused by continuous forces $\widehat{\boldsymbol{\lambda}}dt$ i.e. all the forces with smooth variations within the Lebesgue measure dt;

ν_d is the discrete pure point; e.g. in our system, the velocity jump $\boldsymbol{j}$ given by a contact event or the discrete variation $\boldsymbol{\xi}$ regarding the impulsive reaction force.

so we can introduce the decomposition of velocities and forces into their measures:

$$d\boldsymbol{v} = \boldsymbol{a}dt + \boldsymbol{j}$$
$$d\boldsymbol{\lambda} = \widehat{\boldsymbol{\lambda}}dt + \boldsymbol{\xi}$$

Thanks to these changes we can reformulate the problem eq. (3.18) into an MDI

$$Md\boldsymbol{v} = \boldsymbol{f}(\boldsymbol{q},\boldsymbol{v},t)dt + D_{\mathscr{C}}d\boldsymbol{\lambda}_{\mathscr{C}} \tag{3.20a}$$
$$\Upsilon_{\mathscr{C}} \ni d\boldsymbol{\lambda}_{\mathscr{C}} \perp \bar{\boldsymbol{u}}_{\mathscr{C}} \in \Upsilon^*_{\mathscr{C}} \tag{3.20b}$$
$$\dot{\boldsymbol{q}} = \Gamma(\boldsymbol{q})\boldsymbol{v} \tag{3.20c}$$

This system can be solved with the unknown variations on a finite time step h that are:

- velocity variations: $\boldsymbol{v}^{(t+h)} - \boldsymbol{v}^{(t)}$;
- reaction impulses: $\boldsymbol{\lambda} = \int_{[t,t+h)} d\boldsymbol{\lambda}$.

On this latter system (eq. (3.20)), the time stepping schemes are applied.

During the time step integration some numerical inaccuracies may occur. Operating at the velocity level may cause, on the long run, a significant drift in the constraints that may show inter-penetration, or other undesired round-off and finite precision errors. The only way to circumvent the problem is to add a stabilization term for both unilateral and bilateral constraints:

$$\begin{cases} \nabla\Psi_i^T\boldsymbol{v}^{l+1} = 0 \\ \nabla\boldsymbol{\Phi}_i^T\boldsymbol{v}^{l+1} \geq 0 \end{cases} \longrightarrow \begin{cases} \nabla\Psi_i^T\boldsymbol{v}^{l+1} + \frac{1}{h}\Psi_i + \frac{\partial\Psi_i}{\partial t} = 0 \\ \nabla\boldsymbol{\Phi}_i^T\boldsymbol{v}^{l+1} + \frac{1}{h}\boldsymbol{\Phi}_i \geq 0 \end{cases} \tag{3.21}$$

with this additional term we will present now a basic, while complete, model for the time step integration

$$\begin{bmatrix} H & -C^T & -D^T \\ C^T & 0 & 0 \\ D^T & 0 & 0 \end{bmatrix} \begin{Bmatrix} \boldsymbol{v}^{(l+1)} \\ \boldsymbol{\gamma} \\ \boldsymbol{\lambda} \end{Bmatrix} - \begin{Bmatrix} M\boldsymbol{v}^{(l)} + h\boldsymbol{f}^{(l)} \\ -\frac{1}{h}\boldsymbol{\Psi}^{(l)} - \frac{\partial \boldsymbol{\Psi}}{\partial t}^{(l)} \\ -\frac{1}{h}\boldsymbol{\Phi}^{(l)} \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{0} \\ \boldsymbol{0} \\ \underline{\boldsymbol{u}}_{\mathscr{J}} \end{Bmatrix} \tag{3.22a}$$

$$\Upsilon_{\mathscr{J}} \ni \boldsymbol{\lambda} \perp \underline{\boldsymbol{u}}_{\mathscr{J}} \in \Upsilon^*_{\mathscr{J}} \tag{3.22b}$$

$$\boldsymbol{q}^{(l+1)} = \boldsymbol{q}^{(l)} + h\boldsymbol{v}^{(l+1)} \tag{3.22c}$$

where the $\underline{\boldsymbol{u}}_{\mathscr{J}}$ term includes the correction terms eq. (3.21). Where D is a diagonal composition of the Jacobians $\boldsymbol{D}_i$ of the contact constraints.

H is a modified version of the mass matrix M where

$$H = M - h^2\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}} - h\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{v}}$$
$$H = M - h^2\nabla_q \boldsymbol{f} - h\nabla_v \boldsymbol{f}$$

in order to take into account also the Jacobian of forces to allow the introduction, in the system, of forces that may arise from finite elements. In fact, the gradients on positions $\nabla_q \boldsymbol{f}$ and velocities $\nabla_q \boldsymbol{f}$ are, except for the sign, the translation of stiffness and damping matrix for finite elements.

We are now able to fit a mixed rigid-flexible bodies problem in the same simulation, allowing rigid contacts.

Clearly the problem (3.22) is not a simple linear system and requires specific tailored solvers that can effectively deal with complementarity constraints. Because of this, the whole chapter 4 is dedicated to an eligible solution for this specific scenario.

3.4 Dynamic Model from DAE Time Step Integration

The previous time stepping scheme is obtain directly in the DVI/MDI flavor, but we may arrive at the same conclusions also considering a classic DAE time stepping scheme in which unilateral constraints are added.

Let us start with an implicit Euler method for which, given a discrete time step h, the following time-stepping scheme applies

$$\hat{M}(\boldsymbol{v}^{l+1} - \boldsymbol{v}^{l}) - h\boldsymbol{f}^{l+1} - hD^{T}\boldsymbol{\lambda}^{l+1} - hC^{T}\boldsymbol{\gamma}^{l+1} = 0 \tag{3.23}$$

$$\boldsymbol{q}^{l+1} - \boldsymbol{q}^{l} - \boldsymbol{v}^{l+1}h = 0 \tag{3.24}$$

where $\hat{M} = (M^{l+1} + M^{l})/2$. However, the constraint equations must be enforced

$$\boldsymbol{\Psi}(\boldsymbol{q}^{l+1}, t^{l+1}) = 0 \tag{3.25}$$

$$\boldsymbol{\Phi}(\boldsymbol{q}^{l+1}, t^{l+1}) - \hat{\boldsymbol{y}}^{l+1} = 0 \tag{3.26}$$

moreover, it has to be noticed that the complementarity conditions should hold, thus

$$\hat{\boldsymbol{y}}^{l+1} \geq 0 \tag{3.27a}$$

$$\boldsymbol{\lambda}^{l+1} \geq 0 \tag{3.27b}$$

$$\hat{\boldsymbol{y}}_{i}^{l+1}\boldsymbol{\lambda}_{i}^{l+1} = 0 \quad i \in \mathscr{I} \tag{3.27c}$$

In order to solve eq. (3.24) and eq. (3.23) we can apply a Newton's iteration to

$$G(\boldsymbol{q}^{l+1}, \boldsymbol{v}^{l+1}, \boldsymbol{\gamma}^{l+1}, \boldsymbol{\lambda}^{l+1}) = \begin{bmatrix} \hat{M}(\boldsymbol{v}^{l+1} - \boldsymbol{v}^{l}) - h\boldsymbol{f}^{l+1} - hC^{T}\boldsymbol{\gamma}^{l+1} - hD^{T}\boldsymbol{\lambda}^{l+1} \\ \boldsymbol{q}^{l+1} - \boldsymbol{q}^{l} - h\boldsymbol{v}^{l+1} \end{bmatrix}$$

s

Now, deriving eq. (3.24) and eq. (3.23) respect to $\boldsymbol{q}^{l+1}$, $\boldsymbol{v}^{l+1}$, $\boldsymbol{\gamma}^{l+1}$ and $\boldsymbol{\lambda}^{l+1}$ and approximating the constraint equations as

$$\boldsymbol{\Psi}^{l+1} = \boldsymbol{\Psi}(\boldsymbol{q}^{l+1}, t^{l+1}) \approx \boldsymbol{\Psi}^{l} + C\Delta\boldsymbol{q} + h\frac{\partial \boldsymbol{\Psi}}{\partial t}$$

$$\boldsymbol{\Phi}^{l+1} = \boldsymbol{\Phi}(\boldsymbol{q}^{l+1}) \approx \boldsymbol{\Phi}^{l} + D\Delta\boldsymbol{q}$$

we get

$$\begin{bmatrix} -h\nabla_q \boldsymbol{f}^{l+1} & \hat{M} - h\nabla_v \boldsymbol{f}^{l+1} & -hC^T & -hD^T \\ I & -hI & 0 & 0 \\ C & 0 & 0 & 0 \\ D & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{q}^{l+1} \\ \Delta \boldsymbol{v}^{l+1} \\ \Delta \boldsymbol{\gamma}^{l+1} \\ \Delta \boldsymbol{\lambda}^{l+1} \end{bmatrix} = - \begin{bmatrix} \boldsymbol{q}^{l+1} - \boldsymbol{q}^l - \boldsymbol{v}^{l+1} h \\ \hat{M}(\boldsymbol{v}^{l+1} - \boldsymbol{v}^l) - h\boldsymbol{f}^{l+1} - hC^T \boldsymbol{\gamma}^{l+1} - hD^T \boldsymbol{\lambda}^{l+1} \\ \boldsymbol{\Psi}^l + h\frac{\partial \boldsymbol{\Psi}}{\partial t} \\ \boldsymbol{\Phi}^l \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \hat{\boldsymbol{y}}^{l+1} \end{bmatrix} \tag{3.28}$$

We may now want to add the following equivalence

$$\begin{aligned} \Delta \boldsymbol{q}^{l+1} &= h\Delta \boldsymbol{v}^{l+1} \\ \Delta \boldsymbol{v}^{l+1} &= \boldsymbol{v}^{l+1} - \boldsymbol{v}^l \end{aligned}$$

so we obtain, dividing the constraint equations by h, grouping h in the Lagrangian multipliers:

$$\begin{bmatrix} \hat{M} - h\nabla_v \boldsymbol{f}^{l+1} - h^2 \nabla_q \boldsymbol{f}^{l+1} & -C^T & -D^T \\ C & 0 & 0 \\ D & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{v}^{l+1} \\ h\Delta \boldsymbol{\gamma}^{l+1} \\ h\Delta \boldsymbol{\lambda}^{l+1} \end{bmatrix} = - \begin{bmatrix} \hat{M}(\boldsymbol{v}^{l+1} - \boldsymbol{v}^l) - h\boldsymbol{f}^{l+1} - hC^T \boldsymbol{\gamma}^{l+1} - hD^T \boldsymbol{\lambda}^{l+1} \\ \frac{\boldsymbol{\Psi}^l}{h} + \frac{\partial \boldsymbol{\Psi}}{\partial t} \\ \frac{\boldsymbol{\Phi}^l}{h} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \boldsymbol{y}^{l+1} \end{bmatrix} \tag{3.29}$$

where $\boldsymbol{y}^{l+1} = \hat{\boldsymbol{y}}^{l+1}/h$.

A further modification may occur in order to obtain the Euler semi-implicit integrator. In fact, one can consider the initial tentative values

$$\begin{aligned} \boldsymbol{v}_0^{l+1} = 0 \quad &\Longrightarrow \quad \Delta \boldsymbol{v}^{l+1} = \boldsymbol{v}^{l+1} \\ \boldsymbol{\gamma}_0^{l+1} = 0 \quad &\Longrightarrow \quad \Delta \boldsymbol{\gamma}^{l+1} = \boldsymbol{\gamma}^{l+1} \\ \boldsymbol{\lambda}_0^{l+1} = 0 \quad &\Longrightarrow \quad \Delta \boldsymbol{\lambda}^{l+1} = \boldsymbol{\lambda}^{l+1} \\ \boldsymbol{f}^{l+1} &\approx \boldsymbol{f}^l \end{aligned}$$

It has to be noticed that we are imposing, for example $\boldsymbol{v}_0^{l+1} = 0$, that does not mean that the velocity of the previous step is zero (that would have been $\boldsymbol{v}^l = 0$): we are just imposing the initial value of the current Newton iteration, not the initial value of the time integration. So the system becomes

$$\begin{bmatrix} H & -C^T & -D^T \\ C & 0 & 0 \\ D & 0 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{v}^{l+1} \\ h\boldsymbol{\gamma}^{l+1} \\ h\boldsymbol{\lambda}^{l+1} \end{bmatrix} = - \begin{bmatrix} -H\boldsymbol{v}^l - h\boldsymbol{f}^{l+1} \\ \frac{\boldsymbol{\Psi}^l}{h} + \frac{\partial \boldsymbol{\Psi}}{\partial t} \\ \frac{\boldsymbol{\Phi}^l}{h} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \boldsymbol{y}^{l+1} \end{bmatrix} \tag{3.30}$$

where $H = \hat{M} - h\nabla_v \boldsymbol{f}^{l+1} - h^2 \nabla_q \boldsymbol{f}^{l+1}$.

It is now evident the equivalence with eq. (3.22), once one considers the complementarity constraints (eq. (3.27)), and also with section 4.11.

Chapter 4

Interior-Point Method for QP

Interior-Point Methods are one of the most eminent families of algorithms widely used in optimization [37], especially for problems that arise from Linear and Quadratic Programming.

In the past, other techniques have being used, one for all the Simplex Method. This algorithms progress towards the solution, trying to guess the set of active constraints and, because of this process, they are better fit for large system with fewer constraints. However, the worst-case bound [38] could lead to convergence rates proportional to the exponent of the problem dimension and because of this, even if it is hardly hit in practice, it motivated the development of newer methods with hopefully polynomial with the problem size. Interior-Point algorithms sparkled thanks to this research effort [39–42].

On the opposite to simplex methods, IP follows a very different approach. The first difference it is the one that is just highlighted in the name: the iterative process follows a path that does not lie on the boundaries of the feasible set, as the simplex method would do, but instead it tries to stay as far as possible from them in order to allow for longer steps towards the solution, hitting the constraints to the limit only in the very last steps. Interior-Point Methods can also be distinguished as feasible or unfeasible depending on whether the process takes approximation that are part of the feasible or not-feasible domain.

Moreover, there is also a huge difference in terms of computational cost. While simplex methods are used to smaller, quicker steps, IP is more prone to longer steps at the cost of more involved and computational expensive computations. The processes, in fact, depends directly on the solution of the KKT system that is usually the bottleneck of the whole algorithm.

This chapter is dedicated to deepen the knowledge on the interior-point method itself and to support its implementation in the Chrono environment.

4.1 Newton's Iteration

The core of interior point methods is the solution of the KKT system 2.14 by means of Newton's iterations. So, the first step is to specialize the system 2.14 for the specific functions in use: in our case, those of eq. (2.23). The Lagrangian function and its derivative for this case (QP) are

$$\mathscr{L}(\boldsymbol{x},\boldsymbol{\gamma},\boldsymbol{\lambda}) = \frac{1}{2}\boldsymbol{v}^T H\boldsymbol{v} + \boldsymbol{c}^T\boldsymbol{v} - \boldsymbol{\gamma}^T\left(C\boldsymbol{v}-\boldsymbol{b}_\gamma\right) - \boldsymbol{\lambda}^T\left(D\boldsymbol{v}-\boldsymbol{b}_\lambda\right) \tag{4.1a}$$

$$\nabla_x\mathscr{L}(\boldsymbol{x},\boldsymbol{\gamma},\boldsymbol{\lambda}) = H\boldsymbol{v} + \boldsymbol{c} - C^T\boldsymbol{\gamma} - D^T\boldsymbol{\lambda} \tag{4.1b}$$

where the Lagrangian multipliers are separated in order to distinguish those that are related to inequality constraints ($\boldsymbol{\lambda} \geq 0$) and equality constraints ($\boldsymbol{\gamma}$) and where we leveraged the symmetry of H so that $(H+H^T)\boldsymbol{x} = 2H\boldsymbol{x}$.

Although the ∇ operator is usually considered as a row vector $\nabla\boldsymbol{g} \coloneqq \begin{bmatrix}\frac{\partial \boldsymbol{g}}{\partial x_1} & \cdots & \frac{\partial \boldsymbol{g}}{\partial x_n}\end{bmatrix}$, in order to have more common column-shaped vectors, we operated a transpose to the derivative of the Lagrangian function.

Introducing $\nabla_x\mathscr{L}$, the KKT will then be rewritten as

$$H\boldsymbol{v} + \boldsymbol{c} - C^T\boldsymbol{\gamma} - D^T\boldsymbol{\lambda} = 0 \tag{4.2a}$$

$$C\boldsymbol{v} - \boldsymbol{b}_\gamma = 0 \tag{4.2b}$$

$$D\boldsymbol{v} - \boldsymbol{b}_\lambda \geq 0 \tag{4.2c}$$

$$(D\boldsymbol{v} - \boldsymbol{b}_\lambda) \circ \boldsymbol{\lambda} = 0 \tag{4.2d}$$

$$\boldsymbol{\lambda} \geq 0 \tag{4.2e}$$

where the $\circ$ operator expresses the component-wise or Hadamard product.

Moreover, in order to move the non-negativity constraint to a single variable and simplify the complementarity condition, the *slack* variable $\boldsymbol{y}$ is introduced

$$\boldsymbol{r}_d := H\boldsymbol{v} + \boldsymbol{c} - C^T\boldsymbol{\gamma} - D^T\boldsymbol{\lambda} = 0 \tag{4.3a}$$

$$\boldsymbol{r}_{p_\gamma} := C\boldsymbol{v} - \boldsymbol{b}_\gamma = 0 \tag{4.3b}$$

$$\boldsymbol{r}_{p_\lambda} := D\boldsymbol{v} - \boldsymbol{b}_\lambda - \boldsymbol{y} = 0 \tag{4.3c}$$

$$\boldsymbol{y} \circ \boldsymbol{\lambda} = 0 \tag{4.3d}$$

$$\boldsymbol{\lambda} \geq 0 \tag{4.3e}$$

$$\boldsymbol{y} \geq 0 \tag{4.3f}$$

We highlight here the definition of the *residuals* $\boldsymbol{r}_d$, $\boldsymbol{r}_{p_\gamma}$ and $\boldsymbol{r}_{p_\lambda}$. In fact, the iterative process will identify a tentative solution $(\boldsymbol{x}_l, \boldsymbol{\gamma}_l, \boldsymbol{\lambda}_l, \boldsymbol{y}_l)$ that only approximates the actual solution, in particular respect to:

- $\boldsymbol{r}_d$: the stationarity of the Lagrangian function in the given point;
- $\boldsymbol{r}_{p_\lambda}$: the violation of the inequality constraints;
- $\boldsymbol{r}_{p_\gamma}$: the violation of the equality constraints.

We could also rewrite the complementarity condition in matrix form as:

$$Y\Lambda\boldsymbol{e} = 0 \tag{4.4}$$

where $Y = \operatorname{diag}(\boldsymbol{y})$, $\Lambda = \operatorname{diag}(\boldsymbol{\lambda})$ and $\boldsymbol{e} = [1, \ldots, 1] \in \mathbb{R}^{n_\lambda}$.

Or, again, using a scalar value that summarizes the overall violation of the complementarity

$$\mu = \frac{\boldsymbol{y}^T\boldsymbol{\lambda}}{n_\lambda} \tag{4.5}$$

Looking at the firsts four equations in 4.3, it is easy to notice that they should equal zero at the optimal point. Because of this, it would be possible to apply some suitable solving algorithm for multivariate functions, in order to get an approximated

solution. First of all, let us build the *residuals* function to be solved:

$$\boldsymbol{F}(\boldsymbol{v},\boldsymbol{\gamma},\boldsymbol{\lambda},\boldsymbol{y}) \coloneqq \begin{bmatrix} \boldsymbol{r}_d \\ \boldsymbol{r}_{p_\gamma} \\ \boldsymbol{r}_{p_\lambda} \\ Y\Lambda\boldsymbol{e} \end{bmatrix} = 0 \tag{4.6}$$

For the solution of $\boldsymbol{F}(\boldsymbol{v},\boldsymbol{y},\boldsymbol{\gamma},\boldsymbol{\lambda}) = 0$ it is now possible to apply Newton's Method. Since the components are linear, it is particularly easy to retrieve their derivatives. The intention is to move the function towards zero by making a step whose direction is suggested by a first-order approximation of the original function.

$$\boldsymbol{F}(\boldsymbol{v}+\Delta\boldsymbol{v},\boldsymbol{\gamma}+\Delta\boldsymbol{\gamma},\boldsymbol{\lambda}+\Delta\boldsymbol{\lambda},\boldsymbol{y}+\Delta\boldsymbol{y}) \approx \boldsymbol{F}(\boldsymbol{v},\boldsymbol{\gamma},\boldsymbol{\lambda},\boldsymbol{y}) + \nabla\boldsymbol{F}\Big|_{\boldsymbol{v},\boldsymbol{\gamma},\boldsymbol{\lambda},\boldsymbol{y}} \begin{bmatrix} \Delta\boldsymbol{v} \\ \Delta\boldsymbol{\gamma} \\ \Delta\boldsymbol{\lambda} \\ \Delta\boldsymbol{y} \end{bmatrix} \tag{4.7}$$

Recalling that we would like to have $\boldsymbol{F}(\boldsymbol{v}+\Delta\boldsymbol{v},\boldsymbol{\gamma}+\Delta\boldsymbol{\gamma},\boldsymbol{\lambda}+\Delta\boldsymbol{\lambda},\boldsymbol{y}+\Delta\boldsymbol{y}) = 0$ we aim to solve the following system in the variable $\begin{bmatrix}\Delta\boldsymbol{v} & \Delta\boldsymbol{\gamma} & \Delta\boldsymbol{\lambda} & \Delta\boldsymbol{y}\end{bmatrix}^T$

$$\nabla\boldsymbol{F}\Big|_{\boldsymbol{v},\boldsymbol{y},\boldsymbol{\gamma},\boldsymbol{\lambda}} \begin{bmatrix} \Delta\boldsymbol{v} \\ \Delta\boldsymbol{\gamma} \\ \Delta\boldsymbol{\lambda} \\ \Delta\boldsymbol{y} \end{bmatrix} = -\boldsymbol{F}(\boldsymbol{v},\boldsymbol{\gamma},\boldsymbol{\lambda},\boldsymbol{y}) = - \begin{bmatrix} \boldsymbol{r}_d \\ \boldsymbol{r}_{p_\gamma} \\ \boldsymbol{r}_{p_\lambda} \\ Y\Lambda\boldsymbol{e} \end{bmatrix} \tag{4.8}$$

This result applies in general, not only to QP problems. However, in this particular case, the gradient $\nabla\boldsymbol{F}$ is easily computed,

f	$\nabla_v f$	$\nabla_\gamma f$	$\nabla_\lambda f$	$\nabla_y f$
$\boldsymbol{r}_d$	H	$-C$	$-D$	0
$\boldsymbol{r}_{p_\gamma}$	C	0	0	0
$\boldsymbol{r}_{p_\lambda}$	D	0	0	$-I$
$\boldsymbol{y}\circ\boldsymbol{\lambda}$	0	0	$\mathrm{diag}(\boldsymbol{y})$	$\mathrm{diag}(\boldsymbol{\lambda})$

so eq. (4.8) becomes

$$H\Delta\boldsymbol{v} - C^T\Delta\boldsymbol{\gamma} - D^T\Delta\boldsymbol{\lambda} = -\boldsymbol{r}_d \tag{4.9a}$$

$$C\Delta\boldsymbol{v} = -\boldsymbol{r}_{p_\gamma} \tag{4.9b}$$

$$D\Delta\boldsymbol{v} - \Delta\boldsymbol{y} = -\boldsymbol{r}_{p_\lambda} \tag{4.9c}$$

$$Y\Delta\boldsymbol{\lambda} + \Lambda\Delta\boldsymbol{y} = -\boldsymbol{y}\circ\boldsymbol{\lambda} \tag{4.9d}$$

or in matrix form

$$\begin{bmatrix} H & -C^T & -D^T & 0 \\ C & 0 & 0 & 0 \\ D & 0 & 0 & -I \\ 0 & 0 & Y & \Lambda \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{v} \\ \Delta\boldsymbol{\gamma} \\ \Delta\boldsymbol{\lambda} \\ \Delta\boldsymbol{y} \end{bmatrix} = - \begin{bmatrix} \boldsymbol{r}_d \\ \boldsymbol{r}_{p_\gamma} \\ \boldsymbol{r}_{p_\lambda} \\ Y\Lambda\boldsymbol{e} \end{bmatrix} \tag{4.10}$$

4.2 Step Length

The bare Newton iteration approximate the function to the first-order in order to guess a direction that will reduce its value. However, there are three missing points:

1. There is no guarantee that the first-order approximation will hold for long steps; the $\boldsymbol{F}$ function may vary, indeed. So, multiple steps are required in order to get the actual solution;

2. There is no safe-guard about the non-negativity of $\boldsymbol{\lambda}$ and $\boldsymbol{y}$;

3. Different components of the $\boldsymbol{F}$ function may be reduced at different rates; this may allow more efficient step choices.

In order to take into account points 1 and 2 we operate on the step length. The direction suggested by the Newton's Method can be followed also only for a shorter distance $\alpha \in]0,1]$, in fact, and this length can be chosen to fit the other requirements. Moreover, given point 3, we could also choose different step length for different components of $\boldsymbol{F}$. In order to maximize the step length, while staying feasible to the

non-negativity constraints, we do the following

$$\begin{cases} \max\{\alpha_p > 0 \mid \boldsymbol{y} + \alpha_p \Delta \boldsymbol{y} \geq 0\} \\ \max\{\alpha_d > 0 \mid \boldsymbol{\lambda} + \alpha_d \Delta \boldsymbol{\lambda} \geq 0\} \end{cases} \implies \begin{cases} \alpha_P = \min\left\{1, \eta \min_{\Delta \boldsymbol{y}_i < 0} -\frac{\boldsymbol{y}_i}{\Delta \boldsymbol{y}_i}\right\} \\ \alpha_D = \min\left\{1, \eta \min_{\Delta \boldsymbol{\lambda}_i < 0} -\frac{\boldsymbol{\lambda}_i}{\Delta \boldsymbol{\lambda}_i}\right\} \end{cases} \tag{4.11}$$

where the variable $\eta \in (0,1)$ may be set to a value progressively closer to one as the iterations approach the optimal solution.

However, having two different step lengths for different components of the residual function may cause some divergence issues: a more robust, while less aggressive, approach is to use a precautionary value for α set as

$$\alpha = \min\{\alpha_P, \alpha_D\} \tag{4.12}$$

to be used in the update of the whole set of variables.

In any case, given the step length(s), the variables and the residual function can be updated with inexpensive operations, considering eq. (4.10), in the following way

$$\boldsymbol{v} \leftarrow \boldsymbol{v} + \alpha_P \Delta \boldsymbol{v} \tag{4.13a}$$
$$\boldsymbol{\gamma} \leftarrow \boldsymbol{\gamma} + \alpha_D \Delta \boldsymbol{\gamma} \tag{4.13b}$$
$$\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \alpha_D \Delta \boldsymbol{\lambda} \tag{4.13c}$$
$$\boldsymbol{y} \leftarrow \boldsymbol{y} + \alpha_P \Delta \boldsymbol{y} \tag{4.13d}$$

$$\boldsymbol{r}_d \leftarrow (1 - \alpha_P)\, \boldsymbol{r}_d + (\alpha_D - \alpha_P)\, H \Delta \boldsymbol{v} \tag{4.14a}$$
$$\boldsymbol{r}_{p_\lambda} \leftarrow (1 - \alpha_P)\, \boldsymbol{r}_{p_\lambda} \tag{4.14b}$$
$$\boldsymbol{r}_{p_\gamma} \leftarrow (1 - \alpha_D)\, \boldsymbol{r}_{p_\gamma} \tag{4.14c}$$

This will avoid many matrix-by-vector multiplications, substituted by simpler multiply-and-add.

4.3 Central Path

At this point the very basics of interior-point algorithms has already been defined. However, this does not suffice for an efficient scheme.

The first obstacle is represented by the step length. As we can see in eq. (4.11), it is required that each component y_i and λ_i, after the update, respects the non-negative clause $y_i \geq 0$ and $\lambda_i \geq 0$. Imagine that just one component is close to the boundaries, i.e. close to zero on the positive side, while the other are still far away: e.g. $y_1 \approx 0^+$. During the Newton's step, the gradient will provide the best descent direction that pushes y_1 towards zero, even if y_1 is already quite close. However, the step must be very short, in order to not exceed zero and to not go in the negative side ($\alpha \approx 0$). Unfortunately, since α is shared between all the components of $\boldsymbol{y}$, this will depress the step also for the other components even if they are still far away from the boundaries, causing a stall of the whole scheme.

Hence, in order to overcome this pathological condition, we have to make sure that all the components are decreased, during the iterations, at the same rate; this would not be critical, if there was not for the other constraints affecting $\boldsymbol{y}$ and $\boldsymbol{\lambda}$: namely the complementarity constraint $\boldsymbol{y} \circ \boldsymbol{\lambda} = 0$. Aiming to reduce this residual in fact, without any further safe-guard, will quickly degenerate to the undesirable condition above.

Before trying to solve this issue, we would like to depict the situation from a geometric point of view. The vector $\boldsymbol{s} := \boldsymbol{y} \circ \boldsymbol{\lambda}$, given the non-negative constraints, will always be $\boldsymbol{s} \geq 0$ since both $\boldsymbol{\lambda} \geq 0$ and $\boldsymbol{y} \geq 0$. Thus, $\boldsymbol{s}$ will always lie in the non-negative orthant and the hyperplanes that delimit the orthant represent the boundaries.

From this point of view, the intention to homogeneously reduce the components of $\boldsymbol{y}$ and $\boldsymbol{\lambda}$, is reflected into reducing the components of the $\boldsymbol{s}$ at the same rate, that geometrically means that $\boldsymbol{s}$ should stay close to the bisector of the non-negative orthant $\boldsymbol{e}$ also known as *central path*, up until its annihilation at the optimal point. This represents a little deviation from the original path: the approximated solutions, in order to avoid hitting the boundaries should move more close to the bisector, instead of moving directly towards the solution $\boldsymbol{s}^* = 0$

The questions that now arises are:

- how far from the boundaries/how close to the bisector one should stay in order to not penalize the convergence speed and, at the same time, to avoid hitting the boundaries?
- which point on the bisector we should target? and similarly
- how can we modify the Newton step so that the final solution to which the iteration converge is the same as the original KKT system?

A viable choice could be to relax, during the iterations, the complementarity condition in the following manner: $\boldsymbol{y} \circ \boldsymbol{\lambda} = \sigma\mu\boldsymbol{e}$. This change will allow to

- choose how close we pass to the bisector, by mean of the *centering parameter* σ;
- target a point that will move progressively toward the actual solution, by mean of the complementarity measure μ;
- it annihilates at the solution: in fact $\mu = 0$ at the solution.

The new KKT conditions will become

$$\boldsymbol{r}_d := H\boldsymbol{v} + \boldsymbol{c} - C^T\boldsymbol{\gamma} - D^T\boldsymbol{\lambda} = 0 \tag{4.15a}$$
$$\boldsymbol{r}_{p_\gamma} := C\boldsymbol{v} - \boldsymbol{b}_\gamma = 0 \tag{4.15b}$$
$$\boldsymbol{r}_{p_\lambda} := D\boldsymbol{v} - \boldsymbol{b}_\lambda - \boldsymbol{y} = 0 \tag{4.15c}$$
$$\boldsymbol{y} \circ \boldsymbol{\lambda} = \sigma\mu\boldsymbol{e} \tag{4.15d}$$
$$\boldsymbol{\lambda} \geq 0 \tag{4.15e}$$
$$\boldsymbol{y} \geq 0 \tag{4.15f}$$

so that the Newton's Method is asked to solve

$$\boldsymbol{F}(\boldsymbol{v}+\Delta\boldsymbol{v}, \boldsymbol{\gamma}+\Delta\boldsymbol{\gamma}, \boldsymbol{\lambda}+\Delta\boldsymbol{\lambda}, \boldsymbol{y}+\Delta\boldsymbol{y}) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \sigma\mu\boldsymbol{e} \end{bmatrix} \tag{4.16}$$

instead of $\boldsymbol{F}(\boldsymbol{v}+\Delta\boldsymbol{v},\boldsymbol{\gamma}+\Delta\boldsymbol{\gamma},\boldsymbol{\lambda}+\Delta\boldsymbol{\lambda},\boldsymbol{y}+\Delta\boldsymbol{y})=0$ and the Newton iteration scheme 4.10 will become

$$H\Delta\boldsymbol{v}-C^T\Delta\boldsymbol{\gamma}-D^T\Delta\boldsymbol{\lambda}=-\boldsymbol{r}_d \tag{4.17a}$$

$$C\Delta\boldsymbol{v}=-\boldsymbol{r}_{p_\gamma} \tag{4.17b}$$

$$D\Delta\boldsymbol{v}-\Delta\boldsymbol{y}=-\boldsymbol{r}_{p_\lambda} \tag{4.17c}$$

$$Y\Delta\boldsymbol{\lambda}+\Lambda\Delta\boldsymbol{y}=-\boldsymbol{y}\circ\boldsymbol{\lambda}+\sigma\mu\boldsymbol{e} \tag{4.17d}$$

or in matrix form

$$\begin{bmatrix} H & -C^T & -D^T & 0 \\ C & 0 & 0 & 0 \\ D & 0 & 0 & -I \\ 0 & 0 & Y & \Lambda \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{v} \\ \Delta\boldsymbol{\gamma} \\ \Delta\boldsymbol{\lambda} \\ \Delta\boldsymbol{y} \end{bmatrix} = - \begin{bmatrix} \boldsymbol{r}_d \\ \boldsymbol{r}_{p_\gamma} \\ \boldsymbol{r}_{p_\lambda} \\ Y\Lambda\boldsymbol{e} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \sigma\mu\boldsymbol{e} \end{bmatrix} \tag{4.18}$$

Thus, Newton's iterations are not solution-following, since they are not actually chasing the solution, but path-following, since they chase a point on a path.

An interesting result shows that method proposed hereby reduces the complementarity measure at the speed of $O(n_v\log(1/\varepsilon))$ [29].

Even though the system has slightly changed its aspect, the choice of the step length as in 4.2 still applies.

4.4 Logarithmic Barrier

Another way to obtain the same results is to reorder the minimization problem in order to fit the non-negativity constraints directly into the objective function through a penalty function ϕ that depresses any attempt to push the variable $\boldsymbol{y}=D\boldsymbol{v}-\boldsymbol{b}_\lambda$ to move in the negative orthant.

Clearly, we need to choose a function that, as the generic constraint $C_i(\boldsymbol{v})$ is violated, it increases the objective function, like for example the logarithm.

$$\phi(C(\boldsymbol{v}))=\sum_{i=1}^{n_\lambda}\phi_i(C_i(\boldsymbol{v})) \tag{4.19}$$

$$\phi(D\boldsymbol{v}-\boldsymbol{b}_\lambda)=-\boldsymbol{e}^T\log(D\boldsymbol{v}-\boldsymbol{b}_\lambda) \tag{4.20}$$

It is also clear that this function smooths the constraint: in fact, this alternated version depress the $\boldsymbol{y}$ vector also slightly before hitting the boundaries. This will be fair when the iterations are far from the boundaries, but we would like to have a sharper edge as far as the approximated solution comes close the the optimal point.

Because of this, to the objective function the logarithmic barrier is multiplied by a coefficient $1/t$ that should increase as soon as we come close to the optimal point:

$$\begin{aligned} \mathscr{L} &= \frac{1}{2}\boldsymbol{v}^T H\boldsymbol{v} + \boldsymbol{c}^T\boldsymbol{v} - \boldsymbol{\gamma}^T(C\boldsymbol{v} - \boldsymbol{b}_\gamma) - \frac{1}{t}\boldsymbol{e}^T \log(D\boldsymbol{v} - \boldsymbol{b}_\lambda) \\ \nabla_x\mathscr{L} &= H\boldsymbol{v} + \boldsymbol{c} - C^T\boldsymbol{\gamma} - \frac{1}{t}D^T \operatorname{diag}(D\boldsymbol{v} - \boldsymbol{b}_\lambda)^{-1}\boldsymbol{e} \\ \nabla_x\mathscr{L} &= H\boldsymbol{v} + \boldsymbol{c} - C^T\boldsymbol{\gamma} - \frac{1}{t}D^T \operatorname{diag}(\boldsymbol{y})^{-1}\boldsymbol{e} \end{aligned} \tag{4.21}$$

comparing eq. (4.21) to eq. (4.3a) the similarity is evident

$$\begin{cases} -D^T\left(\frac{1}{t}\operatorname{diag}\boldsymbol{y}^{-1}\right)\boldsymbol{e} \\ -D^T\boldsymbol{\lambda} \end{cases} \implies \boldsymbol{\lambda} \equiv \left(\frac{1}{t}\operatorname{diag}(\boldsymbol{y})^{-1}\right)\boldsymbol{e} \implies \boldsymbol{y}\circ\boldsymbol{\lambda} = \frac{1}{t}\boldsymbol{e} \tag{4.22}$$

It is now evident that, choosing $\frac{1}{t} = \sigma\mu$ we return to the KKT system in use. Hence, the Logarithmic Barrier is very close to the Path Following technique, except for very few details: the way the coefficient $\frac{1}{t}$ is chosen and a slightly different approach to the Newton's iteration (not underlined in the thesis).

4.5 Starting Point

The choice of the starting values for the set of variables $\begin{bmatrix}\boldsymbol{v} & \boldsymbol{\gamma} & \boldsymbol{\lambda} & \boldsymbol{y}\end{bmatrix}^T$ is one of the critical issues for Interior-Point algorithms, especially for *feasible* variants like the one proposed in this thesis.

The problem presents two different facets: the first, choosing the starting point so that it lies into the feasible set; the second, trying to pick a starting point already close to the solution.

Different approaches have been found in the literature [40, 43], but still struggle to find a procedure that is insensitive to the initial random guess.

We propose here the following compromise [29], that has been found to be someway effective, with the minimal computational effort.

$$\begin{aligned} \boldsymbol{v}_0 &= [1\dots1]^T \\ \boldsymbol{\gamma}_0 &= [1\dots1]^T \\ \boldsymbol{\lambda}_0 &= \max\{1, |\boldsymbol{\lambda} + \Delta\boldsymbol{\lambda}_p|\} \\ \boldsymbol{y}_0 &= \max\{1, |\boldsymbol{y} + \Delta\boldsymbol{y}_p|\} \end{aligned} \tag{4.23}$$

where $\max\{\}$ is applied component-wise. The variables $\Delta\boldsymbol{\lambda}_p$ and $\Delta\boldsymbol{y}_p$ are tentative solution of a single iteration of the system 4.10. This approach has the double effect of getting closer to the solution while staying away from the boundaries.

However, it should be noticed that the default values are chosen with a random value to which the algorithm is not completely insensitive. Changing this initial random guess by order of magnitude may still lead to instabilities.

4.6 Warm Start

The IP itself is intended to run in general only over a single problem, until it reaches the solution with the required accuracy.

However, our scenario is different: being in loop with a dynamical simulation means that the Interior-Point algorithm is asked to run at every time step a problem that could be in some way related to the previous one. The only difference is actually just a matter of the evolution of the dynamic system in that (usually short) time frame.

Because of this, some attempts have been made to recycle the solution of the previous problem to *warm start* the following; the task is, most of the time, a burden. Few problems arose, especially when the following elements undergo a modification:

- n_v may change; this happens when bodies are added to or excluded from the simulation;
- n_v may be constant, but H may still change; this happens mostly when Finite-Elements are considered; for rigid bodies the mass and inertia properties are constant and so is H;

- n_γ and C may change; (bilateral) links can be added or activated during the simulation or just simply change;

- n_λ and D may change; contacts are added and removed at each time step and deep changes may occur within two time steps.

Because of all these reasons warm starting is not always trivial to apply. We identified two possible warm start strategies:

Full Warm Start when both n_v, n_γ and n_λ are constant within different time-steps; the solution vector might be reused entirely;

Partial Warm Start when even only one from n_v, n_γ and n_λ changed; only parts of the solution can be reused.

Some results will be shown in section 5.1.

4.7 Prediction Compensation - Correction

Suppose to have found a step direction thanks to the solution of the system 4.18 and let us try to apply the updating step to the $\boldsymbol{y}$ and $\boldsymbol{\lambda}$ pair. The complementarity condition will then appear as:

$$(\boldsymbol{y}+\Delta\boldsymbol{y})\circ(\boldsymbol{\lambda}+\Delta\boldsymbol{\lambda}) = \boldsymbol{y}\circ\boldsymbol{\lambda}+\boldsymbol{y}\circ\Delta\boldsymbol{\lambda}+\boldsymbol{\lambda}\circ\Delta\boldsymbol{y}+\Delta\boldsymbol{y}\circ\Delta\boldsymbol{\lambda} \tag{4.24a}$$

$$\boldsymbol{y}\circ\Delta\boldsymbol{\lambda}+\boldsymbol{\lambda}\circ\Delta\boldsymbol{y} = -\boldsymbol{y}\circ\boldsymbol{\lambda}+\sigma\mu\boldsymbol{e} \tag{4.24b}$$

substituting eq. (4.24b) that comes from the Newton iteration scheme 4.17 into the updating 4.24a, we will come to the following result

$$(\boldsymbol{y}+\Delta\boldsymbol{y})\circ(\boldsymbol{\lambda}+\Delta\boldsymbol{\lambda}) = \Delta\boldsymbol{y}\circ\Delta\boldsymbol{\lambda}+\sigma\mu\boldsymbol{e} \neq \sigma\mu\boldsymbol{e}$$

The second-order term $\Delta\boldsymbol{y}\circ\Delta\boldsymbol{\lambda}$ is clearly an undesired term: it could be useful if the next iteration would take care of this.

For this purpose, an alternative version of eq. (4.18) is proposed, that will be used to *correct* the slight overshooting of the standard scheme.

$$H\Delta\boldsymbol{v} - C^T\Delta\boldsymbol{\gamma} - D^T\Delta\boldsymbol{\lambda} = -\boldsymbol{r}_d \tag{4.25a}$$

$$C\Delta\boldsymbol{v} = -\boldsymbol{r}_{p_\gamma} \tag{4.25b}$$

$$D\Delta\boldsymbol{v} - \Delta\boldsymbol{y} = -\boldsymbol{r}_{p_\lambda} \tag{4.25c}$$

$$Y\Delta\boldsymbol{\lambda} + \Lambda\Delta\boldsymbol{y} = -\boldsymbol{y}\circ\boldsymbol{\lambda} - \Delta\boldsymbol{y}_p\circ\Delta\boldsymbol{\lambda}_p + \sigma\mu_p\boldsymbol{e} \tag{4.25d}$$

or in matrix form

$$\begin{bmatrix} H & -C^T & -D^T & 0 \\ C & 0 & 0 & 0 \\ D & 0 & 0 & -I \\ 0 & 0 & Y & \Lambda \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{v} \\ \Delta\boldsymbol{\gamma} \\ \Delta\boldsymbol{\lambda} \\ \Delta\boldsymbol{y} \end{bmatrix} = - \begin{bmatrix} \boldsymbol{r}_d \\ \boldsymbol{r}_{p_\gamma} \\ \boldsymbol{r}_{p_\lambda} \\ Y\Lambda\boldsymbol{e} + \Delta\boldsymbol{y}_p\circ\Delta\boldsymbol{\lambda}_p \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \sigma\mu_p\boldsymbol{e} \end{bmatrix} \tag{4.26}$$

where $\Delta\boldsymbol{y}_p$, $\Delta\boldsymbol{\lambda}_p$ and μ_p are values that have been *predicted* by a previous run of the simpler scheme 4.10 and that we want to compensate for in the newly proposed scheme.

4.8 System Augmentation

The overall dimension of the system can be easily reduced by means of a simple substitution $\Delta\boldsymbol{y} = D\Delta\boldsymbol{v} + \boldsymbol{r}_{p_\lambda}$. Thanks to this modification the system 4.26 becomes

$$\begin{bmatrix} H & -C^T & -D^T \\ C & 0 & 0 \\ D & 0 & \Lambda^{-1}Y \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{v} \\ \Delta\boldsymbol{\gamma} \\ \Delta\boldsymbol{\lambda} \end{bmatrix} = - \begin{bmatrix} \boldsymbol{r}_d \\ \boldsymbol{r}_{p_\gamma} \\ \boldsymbol{r}_{p_\lambda} + \boldsymbol{y} - \Lambda^{-1}(\sigma\mu_p\boldsymbol{e} - \Delta\boldsymbol{y}_p\circ\Delta\boldsymbol{\lambda}_p) \end{bmatrix}$$

The appearance of the matrices inverse Λ^{-1} is not a concern: we remember that Λ is a diagonal matrix with λ as diagonal, so its inverse Λ^{-1} is trivial.

The reader may notice the undesirable effect of having a system matrix that is not symmetric. It is possible to group the minus sign on C^T and D^T and move it to the variables that become $-\Delta\boldsymbol{y}$ and $-\Delta\boldsymbol{\lambda}$. However, this operation corrupts the positive definiteness of the original matrix, if any. Depending on the linear solver available, the user may prefer one choice or the other.

4.9 Degenerate Case

The previous sections assume that the system is characterized by both unilateral and bilateral joints, as it may be for the most general case. However, with some trivial modifications, it is possible to fit also the unilateral-only and bilateral-only case.

The absence of bilateral constraints will bring to the elimination of the second row block in the whole system and of the second column of the system matrix.

$$\begin{bmatrix} H & -D^T \\ D & \Lambda^{-1}Y \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{v} \\ \Delta \boldsymbol{\lambda} \end{bmatrix} = - \begin{bmatrix} \boldsymbol{r}_d \\ \boldsymbol{r}_{p_\lambda} + \boldsymbol{y} - \Lambda^{-1}(\sigma \mu_p \boldsymbol{e} - \Delta \boldsymbol{y}_p \circ \Delta \boldsymbol{\lambda}_p) \end{bmatrix}$$

Still the above system refers to a QP problem.

On the opposite, if no unilateral/inequality constraints are present, there is no need for an iterative solution. The system to be solved is not given by the KKT conditions anymore, but by the one-shot solution of the dynamic equations system:

$$\begin{bmatrix} H & -C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{\gamma} \end{bmatrix} = \begin{bmatrix} -\boldsymbol{c} \\ \boldsymbol{b}_\gamma \end{bmatrix} \tag{4.27}$$

4.10 Redundant Constraints and Softened Contacts

The system matrix (4.11) is prone to rank deficiency whenever one of the Jacobian matrices C and/or D have linearly-dependent rows. This may happen in case of identical bilateral joints that, for example, are added twice by mistake, or when different joints lead to an over-constrained system. Fortunately, this is more common for the Jacobian of bilateral constraints C, rather than D.

In the cases above, the linear solver may be affected by the ill-posed matrix, throw errors and halt the iteration or, at best, can find odd solutions. Nowadays, solvers like MUMPS are able to detect and overcome this critical situations, providing one of the infinite solutions of the almost-singular system. Otherwise, one possible way is to purge any redundant row with SVD or QR revealing methods: a procedure that may be slow and complex and that will tear down the efficiency of the whole system.

However, one can notice that there is a block, namely $\Lambda^{-1}Y$, that may induce the reader to think that this may save the global matrix for any deficiency: if this may

come true for the very few initial steps, we have to remember that the component-wise product y_i/λ_i is intended to be close to zero, especially when the optimal solution is approached. And if the diagonal come close to zero, so are our hopes to restore the rank of the general matrix.

However, the $\Lambda^{-1}Y$ matrix plays an important role that may have a mechanical interpretation. As one can see, this block occupies the same position in the global matrix that would have been occupied by a *compliance matrix*, that is a matrix that makes the contacts act like if a stiffness or spring would have inserted between the contact point pairs. This is, by the way, a well-known technique used to *soften* the constraints allowing a slight inter-penetration in the bodies: in this case a diagonal compliance matrix E with the inverse of the contact stiffness on its diagonal is inserted in the same position of $\Lambda^{-1}Y$, allowing some inter-penetration. However, since in our case this block is progressively pushed towards zero, this means that the IP method tries to approach the solution beginning to solve an alternative problem with softer constraints and proceeding to the optimal solution progressively hardening the contacts stiffness.

In any case, adding a compliance matrix to $\Lambda^{-1}Y$ lets the matrix to be non-singular even with rank-deficient Jacobians D, especially while close to the optimal point

$$\begin{bmatrix} H & C^T & D^T \\ C & 0 & 0 \\ D & 0 & \Lambda^{-1}Y + E \end{bmatrix}$$

4.11 Interior-Point Implementation

We are going now into the implementation details of the Mehrotra [24] scheme proposed in the previous sections whose backbone is the iteration scheme already proposed in 4.26.

In the section 4.7 we considered a variant of the classic IP system in which a triplet of variables, namely $\Delta \boldsymbol{y}_p$ and $\Delta \boldsymbol{\lambda}_p$ and $\mu_p = \boldsymbol{y}_p^T \boldsymbol{\lambda}_p / n_\lambda$, from a previous run of the simpler system 4.10 are used in order to project an estimate and evaluate a correc-

tion term for the following iteration. This scheme is referred as *predictor-corrector*.

In this section the final procedure is summarized in order to provide a monolithic reference for the development of the algorithm. For the C++ implementation we suggest to read appendix B.

Variables and Matrices Initialization

The matrices H, C, D come from the dynamical simulation and are provided ready to use.

For starters, the variables $\boldsymbol{v}$, $\boldsymbol{\gamma}$ and $\boldsymbol{\lambda}$ are taken from the time integration; $\boldsymbol{y}$ is inferred from the eq. (4.15c) that is

$$\boldsymbol{y} = D\boldsymbol{v} - \boldsymbol{b}_\lambda$$

However, this warm start may fail so the algorithm falls back to the standard initialization shown in section 4.5. In order to do so, an early run of the prediction step must be done, with the current values of the variables.

All the elements are now set to update the residuals that can be are fit into the $-\boldsymbol{F}$ vector of eq. (4.8) or, that is the same, into eq. (4.10). However, this will be explicitly shown in the next section, where we will treat the system with the augmentation presented in section 4.8.

Prediction Step

The iterative loop starts with a Prediction Step.

Once the residuals and the variables have been set by the initialization step (section 4.11) or by the previous IP iteration, the following system can be solved to obtain the set of *predicted* variables, to which we will refer with the subscript $_p$. We underline that the system is exactly the same as eq. (4.10) with the augmentation described in section 4.8 or, that is the same, as the system 4.11 with no centering ($\sigma = 0$) and no prediction compensation i.e. without the terms $\Delta\boldsymbol{y}_p \circ \Delta\boldsymbol{\lambda}_p$. We remind that this step is made in order to get a first-attempt value for $\Delta\boldsymbol{y}_p$, $\Delta\boldsymbol{\lambda}_p$ and μ_p that will be later used by the correction step.

The system matrix is loaded with H, C and D according to the following scheme, where for Λ and Y the current values of $\boldsymbol{y}$ and $\boldsymbol{\lambda}$ are used.

So that we come to the formulation that permits to get Newton's iteration predicted direction (augmented form):

$$\begin{bmatrix} H & -C^T & -D^T \\ C & 0 & 0 \\ D & 0 & \Lambda^{-1}Y \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{v} \\ \Delta\boldsymbol{\gamma} \\ \Delta\boldsymbol{\lambda} \end{bmatrix}_p = - \begin{bmatrix} \boldsymbol{r}_d \\ \boldsymbol{r}_{p_\gamma} \\ \boldsymbol{r}_{p_\lambda} + \boldsymbol{y} \end{bmatrix} \tag{4.28}$$

together with

$$\Delta\boldsymbol{y}_p = D\Delta\boldsymbol{v}_p + \boldsymbol{r}_{p_\lambda}$$

We underline that the system matrix above is stored and factorized only once at this step. The correction step will make use just of the same factors, used to solve for a different known vector.

The predicted variables $\Delta\boldsymbol{v}_p$, $\Delta\boldsymbol{\gamma}_p$, $\Delta\boldsymbol{\lambda}_p$, $\Delta\boldsymbol{y}_p$ provide the direction to which the Newton's step will be made. However, the reader is reminded that it will be just a tentative step: the variables $\boldsymbol{v}$, $\boldsymbol{\gamma}$, $\boldsymbol{\lambda}$, $\boldsymbol{y}$ will not be updated yet. Indeed, temporary variables are used.

Given the new directions, the step length is evaluated according to section 4.2. At this point there is no need to take specific precautions in order to not hit the boundaries, thus is $\eta = 1$ for this case. The update of just $\boldsymbol{\lambda}$ and $\boldsymbol{y}$ will occur since they are the only that participate to the correction step.

$$\boldsymbol{\lambda}_p = \boldsymbol{\lambda} + \alpha_{D_p}\Delta\boldsymbol{\lambda}_p \tag{4.29a}$$

$$\boldsymbol{y}_p = \boldsymbol{y} + \alpha_{P_p}\Delta\boldsymbol{y}_p \tag{4.29b}$$

where $\boldsymbol{\lambda}$ and $\boldsymbol{y}$ are taken from the previous iteration or from the initialization step if it is the first iteration. The predicted complementarity measure is evaluated accordingly $\mu_p = \boldsymbol{y}_p^T\boldsymbol{\lambda}_p/n_\lambda$.

Correction Step

To set up the correction step the centering parameter σ must be set. A quite common practice is to use the empirical value of

$$\sigma = \left(\frac{\mu_p}{\mu}\right)^3$$

that we assure to be bounded in $(0,1)$ with proper safe-guards. However, this safeties are hit only in critical situation where an odd decrease speed of the complementarity measure strangely occurred.

Having set this parameter, it is possible to assemble the right-hand side for the augmented system that we rewrite here for sake of completeness

$$\begin{bmatrix} H & -C^T & -D^T \\ C & 0 & 0 \\ D & 0 & \Lambda^{-1}Y \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{v} \\ \Delta \boldsymbol{\gamma} \\ \Delta \boldsymbol{\lambda} \end{bmatrix} = - \begin{bmatrix} \boldsymbol{r}_d \\ \boldsymbol{r}_{p_\gamma} \\ \boldsymbol{r}_{p_\lambda} + \boldsymbol{y} - \Lambda^{-1}(\sigma\mu_p \boldsymbol{e} - \Delta \boldsymbol{y}_p \circ \Delta \boldsymbol{\lambda}_p) \end{bmatrix}$$

We would like to stress once more that the system above differs from the prediction step only for the $-\Lambda^{-1}(\sigma\mu_p \boldsymbol{e} - \Delta \boldsymbol{y}_p \circ \Delta \boldsymbol{\lambda}_p)$ part. In particular, the system matrix is kept untouched from the prediction step, so its factors if a direct solver has been used; this allows for a faster extraction of the new Newton's direction.

An update now follows. The step lengths are calculated according to section 4.2.

$$\alpha_P = \min\left\{1, \eta \min_{\Delta y_i < 0} -\frac{y_i}{\Delta y_i}\right\}$$
$$\alpha_D = \min\left\{1, \eta \min_{\Delta \lambda_i < 0} -\frac{\lambda_i}{\Delta \lambda_i}\right\}$$

and then used in the final update of the variables

$$\boldsymbol{v} \mathrel{+}= \alpha_P \Delta \boldsymbol{v}$$
$$\boldsymbol{\gamma} \mathrel{+}= \alpha_D \Delta \boldsymbol{\gamma}$$
$$\boldsymbol{\lambda} \mathrel{+}= \alpha_D \Delta \boldsymbol{\lambda}$$
$$\boldsymbol{y} \mathrel{+}= \alpha_P \Delta \boldsymbol{y}$$

and residuals

$$\boldsymbol{r}_d \leftarrow (1-\alpha_P)\,\boldsymbol{r}_d + (\alpha_D - \alpha_P)\,H\,\Delta\boldsymbol{v}$$
$$\boldsymbol{r}_{p_\lambda} \leftarrow (1-\alpha_P)\,\boldsymbol{r}_{p_\lambda}$$
$$\boldsymbol{r}_{p_\gamma} \leftarrow (1-\alpha_D)\,\boldsymbol{r}_{p_\gamma}$$

A this point the updated residuals (included the complementarity measure) are compared to a threshold value that will decide for the loop termination or, in case the achieved accuracy does not satisfy the criteria, will bring back to the prediction step (section 4.11) for another iteration.

4.11.1 Interior-Point Pseudo-Code

Algorithm 1 Interior-Point algorithm

```
1:  procedure SOLVE
2:      Assemble system matrix with H and eventually C and/or D;
3:      if D is empty then                          ▷ No unilateral constraints found;
4:          return the solution of the linear system (4.27);
5:      else
6:          set starting point as in section 4.5;
7:          repeat
8:              IP ITERATE
9:          until Reached: Tolerance or Maximum Iterations
10:     end if
11: end procedure
```

12: **procedure** IP ITERATE

Prediction

13: Solve eq. (4.28) to obtain $\begin{bmatrix}\Delta \boldsymbol{v}_p & \Delta \boldsymbol{\gamma}_p & \Delta \boldsymbol{\lambda}_p\end{bmatrix}^T$

14: Evaluate $\Delta \boldsymbol{y}_p$ as $\Delta \boldsymbol{y}_p = D\Delta \boldsymbol{v}_p + \boldsymbol{r}_{p_\lambda}$

15: Calculate step lengths α_P and α_D with eq. (4.11);

16: Evaluate $\boldsymbol{y}_p$ and $\boldsymbol{\lambda}_p$ with eq. (4.29) and μ_p

Correction

17: Set $\sigma = (\mu_p/\mu)^3$

18: Set $\eta \in (0,1]$

19: Solve section 4.11 to obtain $\begin{bmatrix}\Delta \boldsymbol{v} & \Delta \boldsymbol{\gamma} & \Delta \boldsymbol{\lambda}\end{bmatrix}^T$

20: Evaluate $\Delta \boldsymbol{y}$ as $\Delta \boldsymbol{y} = D\Delta \boldsymbol{v} + \boldsymbol{r}_{p_\lambda}$

21: Calculate step lengths α_P and α_D with eq. (4.11);

22: Update $\boldsymbol{v}$, $\boldsymbol{\gamma}$, $\boldsymbol{\lambda}$ and $\boldsymbol{y}$ with eq. (4.13a);

23: Update the complementarity measure μ;

24: Update residuals $\boldsymbol{r}_{p_\lambda}$, $\boldsymbol{r}_{p_\gamma}$ and $\boldsymbol{r}_d$ as in eq. (4.14)

25: **end procedure**

4.12 Common Issues

The iterating procedure of the Interior-Point Method may face some issues, mainly

- Unbalanced convergence of the residual; the centering parameter should provide a safe-guard that leads to a balanced convergence rate of the residual. However, it may happen that the choice of the centering parameter is not always optimal, leading to early reductions of the complementarity measure. Tuning the η parameter may improve the results. Empirically we found the following relation quite effective:

$$\eta = 0.1 \exp(\mu n_\lambda) + 0.9$$

- Penalized starting point; depending on the scale of the problem, the initial values of the variables may significantly affect the number of iterations required; a more effective starting point technique may prove better convergence.

4.13 Linear Solver

The most critical and demanding operation of the aforementioned algorithm is by far the solution of the linear system section 4.11. For this purpose, a great deal of effort has been invested into establishing interfacing classes to link the Interior-Point module to two well-known linear algebra libraries, the Intel Math Kernel Library and the MUltifrontal Massively Parallel Sparse direct Solver (MUMPS).

The first accomplishment regards the development of CSR [44] and COO matrix class that are used to feed the linear solvers. These classes are designed to leverage the internal scheme of Chrono and to provide, with the minimum expense, a compatible matrix description for the underlying linear solver interface. However, different versions of IP algorithm can be also designed to operate matrix-free [45].

However, since the complexity of the Chrono environment and the different storage classes, an additional intermediate layer has been in added in order to handle the data transfer and conversion between the simulation engine and the algebra routines.

For the test proposed in chapter 5, the combination of the MUMPS solver with the BLAS routines from Intel MKL has been used.

In order to minimize the storage requirements, no Schur complement has been used to solve the system. The system has been provided to the linear solver as is. The choice of a direct solver is in reply for a more accurate solution, compared to those offered by iterative solvers [46], and this is a requirement since the IP algorithm, especially in the last steps of the iterations may come to ill-conditioned matrices.

4.14 Interior-Point Method for Cone QP

In the previous section we analyzed the simpler case in which the complementarity condition was:

$$0 \leq \lambda_i \perp y_i \geq 0 \tag{4.30}$$

However, through this simple complementarity condition, we were able to model just a restricted set of problems, such as *frictionless* contacts. It may be useful if, introducing some changes in the standard QP formulation, we would be able to handle also the friction, and in particular the Coulomb friction model. Fortunately, this can be accomplished with few, yet significant, changes to both the QP problem and the Coulomb model, that will be translate into a Cone Complementarity Problem (eq. (1.11), eq. (3.22)). Given the brief description of section 2.11 we will try to extend the IP method to this kind of problems.

Suppose to have a mixed combination of constraints, hence the variable $\boldsymbol{y}$ is constituted by y_i or also by $\boldsymbol{y}_i$ components that may be subject to:

Non-Negativity Constraint the scalar y_i is asked to lie into the non-negative orthant $\mathbb{R}_+$ or, that is the same, $y_i \geq 0$;

Cone Constraint suppose that the cone is in the space $\mathbb{R} \times \mathbb{R}^{p-1}$, then the constraint involves a whole set of p equations and relative variables; thus, we have the form:

$$\boldsymbol{y}_i \succeq 0 \implies \begin{bmatrix} y_{i,0} \\ \boldsymbol{y}_{i,1} \end{bmatrix} \in \mathscr{K}_\mu \quad \left\{ \mu y_{i,0} \geq \|\boldsymbol{y}_{i,1}\| \right\} \tag{4.31}$$

For compactness, we will use the operator $\succeq$ for both the cases.

From the Cone QP formulation eq. (2.27), the KKT optimality conditions are

$$\boldsymbol{r}_d \coloneqq H\boldsymbol{v} + \boldsymbol{c} - C^T\boldsymbol{\gamma} - D^T\boldsymbol{\lambda} = 0 \tag{4.32a}$$

$$\boldsymbol{r}_{p_\gamma} \coloneqq C\boldsymbol{v} - \boldsymbol{b}_\gamma = 0 \tag{4.32b}$$

$$\boldsymbol{r}_{p_\lambda} \coloneqq D\boldsymbol{v} - \boldsymbol{b}_\lambda - \boldsymbol{y} = 0 \tag{4.32c}$$

$$\boldsymbol{y} \circ \boldsymbol{\lambda} = \sigma\mu\boldsymbol{e} \tag{4.32d}$$

$$\boldsymbol{\lambda} \succeq 0 \tag{4.32e}$$

$$\boldsymbol{y} \succeq 0 \tag{4.32f}$$

where the $\succeq$ operator refers to the cone $\mathscr{K}_\mu$, that is a Lorentz cone with apex angle $2\arctan(\mu)$.

We used the notation $\lambda_{i,0}$ and $\boldsymbol{\lambda}_{i,1}$ in the following way: for each cone constraint, we have p equations involved, thus also p components in $\boldsymbol{\lambda}$ and $\boldsymbol{y}$. For example, if the i-th constraint describes a conic constraint of order p, then

$$\boldsymbol{\lambda}_i = \begin{bmatrix} \lambda_{i,0} \\ \\ \boldsymbol{\lambda}_{i,1} \\ \\ \end{bmatrix} \tag{4.33}$$

where $\boldsymbol{\lambda}_{i,1}$ is of size $p-1$ and $\lambda_{i,0}$ is a scalar value. Notice the bold font for the first.

Also the operator $\circ$, used to formulate the complementarity condition, has to take into account the change in the constraints.

$$\boldsymbol{y} \circ \boldsymbol{\lambda} = \begin{cases} y_i\lambda_i & \text{for NNC} \\ \begin{bmatrix} \boldsymbol{y}_i^T\boldsymbol{\lambda}_i \\ y_{i,0}\boldsymbol{\lambda}_{i,1} + \lambda_{i,0}\boldsymbol{y}_{i,1]} \end{bmatrix} & \text{for CC} \end{cases} \tag{4.34}$$

And, in order to allow its inversion, we introduce the $\diamond$ operator that translates

into

$$\boldsymbol{\lambda} \diamond \boldsymbol{y} = \begin{cases} \lambda_i / y_i & \text{for NNC} \\ \begin{bmatrix} \lambda_{i,0} & \boldsymbol{\lambda}_{i,1}^T \\ \boldsymbol{\lambda}_{i,1} & \lambda_{i,0} I \end{bmatrix}^{-1} \begin{bmatrix} y_{i,0} \\ \boldsymbol{y}_{i,1} \end{bmatrix} & \text{for CC} \end{cases} \tag{4.35}$$

then we have $\boldsymbol{y} \circ (\boldsymbol{y} \diamond \boldsymbol{\lambda}) = \boldsymbol{y}$.

Also $\boldsymbol{e}$ is changed accordingly so that

$$\boldsymbol{e} = \begin{cases} 1 & \text{for NNC} \\ \begin{bmatrix} 1 \\ 0_{[p-1]} \end{bmatrix} & \text{for CC} \end{cases} \tag{4.36}$$

With these premises, we should be able to develop the Newton's steps for the solution of the KKT system, but few adjustments will be added in the next sections.

4.14.1 Logarithmic Barrier for Cone QP

In the section 4.4 we have shown the strict relation of the Central Path with the Logarithmic Barrier. We need now to expand the Logarithmic Barrier in order to fit also to the Cone QP formulation. For a generic constraint function $C(\boldsymbol{v}) \succeq 0$, we have

$$\phi(C(\boldsymbol{v})) = \sum_{i \in \mathscr{I}} \phi_i(C_i(\boldsymbol{v})) \quad \begin{cases} \phi_i(C_i) = -\log(C_i) & \text{for NNC} \\ \phi_i(C_i) = -\frac{1}{2} \log\left(C_{i,0}^2 - C_{i,1}^T C_{i,1}\right) & \text{for CC} \end{cases} \tag{4.37}$$

where, for $C_i \equiv C_i(\boldsymbol{v})$ we can substitute, for example $\boldsymbol{y}(\boldsymbol{v}) = D\boldsymbol{v} - \boldsymbol{b}_\lambda$.

The gradient of the barrier $\nabla\phi(\boldsymbol{w})$, required to build the KKT system, can be computed as

$$\begin{cases} \nabla\phi_i(w_i) = -w_i^{-1} & \text{for NNC} \\ \nabla\phi_i(\boldsymbol{w}_i) = -(\boldsymbol{w}_i^T J \boldsymbol{w}_i)^{-1} J \boldsymbol{w}_i & \text{for CC} \end{cases} \tag{4.38}$$

with

$$J = \begin{bmatrix} 1 & 0 \\ 0 & I_{[p-1]} \end{bmatrix}$$

Because of its self-scaled property [47, 48], the gradient of the barrier $\nabla\phi(\boldsymbol{w})$ can be of some use for the next section.

4.14.2 Nesterov-Todd Scaling

In order to obtain a better posed problem, various scaling systems have been developed, like the Jordan Algebra Approach [49] or the Nesterov-Todd Scaling that we we are going to briefly introduce here.

The scaling leaves the cones and the central path invariant, so that all the equation relating $\boldsymbol{y}$ and $\boldsymbol{\lambda}$ are still valid. In particular, we find a primal-dual scaling matrix W such that

$$\tilde{\boldsymbol{y}} = W\boldsymbol{y} \tag{4.39a}$$

$$\tilde{\boldsymbol{\lambda}} = W\boldsymbol{\lambda} \tag{4.39b}$$

We would like to find a matrix W such that

$$\boldsymbol{y} \succeq 0 \iff \tilde{\boldsymbol{y}} \succeq 0 \tag{4.40a}$$

$$\boldsymbol{\lambda} \succeq 0 \iff \tilde{\boldsymbol{\lambda}} \succeq 0 \tag{4.40b}$$

$$\boldsymbol{y} \circ \boldsymbol{\lambda} = \mu\boldsymbol{e} \iff \tilde{\boldsymbol{y}} \circ \tilde{\boldsymbol{\lambda}} = \mu\boldsymbol{e} \tag{4.40c}$$

The Nesterov-Todd scaling at the points $\boldsymbol{y}$ and $\boldsymbol{\lambda}$ is derived from the unique scaling point $\boldsymbol{w}$ that satisfies

$$\nabla\phi(\boldsymbol{w})\boldsymbol{y} = \boldsymbol{\lambda}$$

while the scaling matrix W derives from the factorization $\nabla\phi(\boldsymbol{w})^{-1} = W^T W$, for which a possible solution is $W = \nabla\phi(\boldsymbol{w})^{-1/2}$. From here, we are able to find also the scaled variable

$$\boldsymbol{z} = W^{-T}\boldsymbol{y} = W\boldsymbol{\lambda} \tag{4.41}$$

The scaling matrix W is now retrieved for the NNC and CC cases.

For Non-Negativity Constraints any positive diagonal matrix can be used. We

choose, according to [40], the scaling point and scaling matrix

$$w_i = \sqrt{\frac{y_i}{\lambda_i}} \tag{4.42a}$$

$$W_i = \operatorname{diag}(w_i) = \operatorname{diag}\left(\sqrt{\frac{y_i}{\lambda_i}}\right) \tag{4.42b}$$

$$z_i = W_i^{-1} y_i = W_i \lambda_i = \sqrt{y_i \lambda_i} \tag{4.42c}$$

For the Cone Constraint, the scaling point construction is more convoluted [40], so we report the results in compact form

$$\bar{\boldsymbol{\lambda}}_i = \frac{1}{\sqrt{(\boldsymbol{\lambda}_i^T J \boldsymbol{\lambda}_i)}} \boldsymbol{\lambda}_i$$

$$\bar{\boldsymbol{y}}_i = \frac{1}{\sqrt{(\boldsymbol{y}_i^T J \boldsymbol{y}_i)}} \boldsymbol{y}_i$$

$$\gamma = \sqrt{\frac{1 + \bar{\boldsymbol{\lambda}}_i \bar{\boldsymbol{y}}_i}{2}}$$

$$\bar{\boldsymbol{w}}_i = \frac{1}{2\gamma}(\bar{\boldsymbol{y}}_i + J\bar{\boldsymbol{\lambda}}_i)$$

$$\bar{W}_i = \begin{bmatrix} \bar{w}_{i,0} & \bar{\boldsymbol{w}}_{i,1}^T \\ \bar{\boldsymbol{w}}_{i,1} & I_{[p-1]} + \frac{\bar{\boldsymbol{w}}_{i,1}^T \bar{\boldsymbol{w}}_{i,1}}{\bar{w}_{i,0}+1} \end{bmatrix}$$

$$W_i = \left(\frac{\boldsymbol{y}_i^T J \boldsymbol{y}_i}{\boldsymbol{\lambda}_i^T J \boldsymbol{\lambda}_i}\right)^{1/4} \bar{W}_i$$

The overall scaling matrix W is now assembled from the different W_i, created depending on the specific constraint

$$W = \begin{bmatrix} W_1 & 0 & 0 \\ 0 & W_i & 0 \\ 0 & 0 & W_{\dots} \end{bmatrix}, \quad i \in \mathscr{I} \tag{4.43}$$

Based on this scaling, recalling that

$$\boldsymbol{y} = W^T \boldsymbol{z} \tag{4.44a}$$

$$\boldsymbol{\lambda} = W^{-1} \boldsymbol{z} \tag{4.44b}$$

we rewrite the complementarity condition

$$\boldsymbol{y} \circ \boldsymbol{\lambda} \equiv \boldsymbol{z} \circ \boldsymbol{z} = \sigma \mu \boldsymbol{e} \tag{4.45}$$

The Newton step is derived from the new KKT system and, in particular, regarding the derivation of the complementarity condition, given eq. (A.1a), we get

$$\begin{aligned} \frac{\partial \boldsymbol{r}_z}{\partial \lambda} \Delta \boldsymbol{\lambda} = \frac{\partial (\boldsymbol{z} \circ \boldsymbol{z} - \sigma \mu \boldsymbol{e})}{\partial \boldsymbol{\lambda}} \Delta z = \frac{\partial (W \boldsymbol{\lambda} \circ W^{-T} \boldsymbol{y} - \sigma \mu e)}{\partial \boldsymbol{\lambda}} \Delta \boldsymbol{\lambda} = \\ = W^{-T} \boldsymbol{y} \circ W \Delta \boldsymbol{\lambda} \end{aligned} \tag{4.46}$$

thus we obtain

$$\boldsymbol{r}_z \coloneqq \boldsymbol{z} \circ \boldsymbol{z} - \sigma \mu \boldsymbol{e} \quad \longrightarrow \quad \nabla \boldsymbol{r}_z \begin{bmatrix} \Delta \boldsymbol{v} \\ \Delta \boldsymbol{\gamma} \\ \Delta \boldsymbol{\lambda} \\ \Delta \boldsymbol{y} \end{bmatrix} = -\boldsymbol{r}_z \tag{4.47}$$

noting that $\nabla \boldsymbol{r}_z = \begin{bmatrix} 0 & 0 & \frac{\partial \boldsymbol{r}_z}{\partial \boldsymbol{\lambda}} & \frac{\partial \boldsymbol{r}_z}{\partial \boldsymbol{y}} \end{bmatrix}$ then

$$\begin{aligned} \nabla \boldsymbol{r}_z \begin{bmatrix} \Delta \boldsymbol{v} \\ \Delta \boldsymbol{\gamma} \\ \Delta \boldsymbol{\lambda} \\ \Delta \boldsymbol{y} \end{bmatrix} &\equiv \frac{\partial \boldsymbol{r}_z}{\partial \boldsymbol{\lambda}} \Delta \boldsymbol{\lambda} + \frac{\partial \boldsymbol{r}_z}{\partial \boldsymbol{y}} \Delta \boldsymbol{y} \\ &\equiv W^{-T} \boldsymbol{y} \circ W \Delta \boldsymbol{\lambda} + W \boldsymbol{\lambda} \circ W^{-T} \Delta \boldsymbol{y} \\ &\equiv \boldsymbol{z} \circ W \Delta \boldsymbol{\lambda} + \boldsymbol{z} \circ W^{-T} \Delta \boldsymbol{y} \\ &\equiv \boldsymbol{z} \circ (W \Delta \boldsymbol{\lambda} + W^{-T} \Delta \boldsymbol{y}) = -\boldsymbol{z} \circ \boldsymbol{z} \end{aligned}$$

From this latter equation, reorganizing the terms, we obtain

$$\Delta \boldsymbol{y} = -W^T (\boldsymbol{z} \diamond \boldsymbol{r}_z) - W^T W \Delta \boldsymbol{\lambda}$$

that can be used into the equation of $\boldsymbol{r}_{p_\lambda}$ to rewrite the KKT system without $\Delta \boldsymbol{y}$ in clear.

$$D \Delta \boldsymbol{v} + W^T W \Delta \boldsymbol{\lambda} = -\boldsymbol{r}_{p_\lambda} - W^T (\boldsymbol{z} \diamond \boldsymbol{r}_z)$$

However, for the simpler case where $\boldsymbol{r}_z \coloneqq \boldsymbol{z} \circ \boldsymbol{z}$ (i.e. in the prediction step), the right term turns out to be just

$$D\Delta\boldsymbol{v} + W^T W \Delta\boldsymbol{\lambda} = -\boldsymbol{r}_{p_\lambda} - \boldsymbol{y}$$

The considerations above are valid also in the case where the correction term are added

$$\boldsymbol{r}_z \coloneqq \boldsymbol{z} \circ \boldsymbol{z} - \sigma\mu\boldsymbol{e} + (W^{-T}\Delta\boldsymbol{y}_p \circ W\Delta\boldsymbol{\lambda}_p)$$

Finally, the matrix for the Newton iteration

$$H\Delta\boldsymbol{v} - C^T\Delta\boldsymbol{\gamma} - D^T\Delta\boldsymbol{\lambda} = -\boldsymbol{r}_d \tag{4.48a}$$

$$C\Delta\boldsymbol{v} = -\boldsymbol{r}_{p_\gamma} \tag{4.48b}$$

$$D\Delta\boldsymbol{v} + W^T W\Delta\boldsymbol{\lambda} = -\boldsymbol{r}_{p_\lambda} - W^T(\boldsymbol{z} \diamond \boldsymbol{r}_z) \tag{4.48c}$$

$$\begin{bmatrix} H & -C^T & -D^T \\ C & 0 & 0 \\ D & 0 & W^T W \end{bmatrix} = - \begin{bmatrix} \boldsymbol{r}_d \\ \boldsymbol{r}_{p_\gamma} \\ \boldsymbol{r}_{p_\lambda} + W^T(\boldsymbol{z} \diamond \boldsymbol{r}_z) \end{bmatrix} \tag{4.49}$$

where

$$\boldsymbol{r}_z \coloneqq \boldsymbol{z} \circ \boldsymbol{z} \qquad \text{for the prediction step} \tag{4.50}$$

$$\boldsymbol{r}_z \coloneqq \boldsymbol{z} \circ \boldsymbol{z} - \sigma\mu\boldsymbol{e} + (W^{-T}\Delta\boldsymbol{y}_p \circ W\Delta\boldsymbol{\lambda}_p) \qquad \text{for the correction step} \tag{4.51}$$

4.14.3 Step Length for Cone QP

Given a cone $\mathcal{K}_\mu$ described by eq. (1.2), a position p and a direction d in the $\mathbb{R}^p$ space, we want to know the infimum and supremum of the scalar value α such that the updated vector $\boldsymbol{p} + \alpha\boldsymbol{d} \in \mathcal{K}_\mu$ (i.e. belongs to the cone) or, in another form, $\boldsymbol{p} + \alpha\boldsymbol{d} \succeq 0$ where the $\succeq$ operator here implicitly refers to the cone $\mathcal{K}_\mu$.

Given that the updated vector should be contained in the cone, we can state that:

$$\mu(p_0 + \alpha d_0) \geq \|\boldsymbol{p}_1 + \alpha\boldsymbol{d}_1\| \tag{4.52}$$

That is equivalent to the following system, obtained quadrating both terms, making sure to hold the non-negativity of the left side:

$$\begin{cases} \alpha^2 \left(\mu^2 d_0^2 - \sum_{i=1}^{n} d_i^2 \right) + 2\alpha \left(\mu^2 p_0 d_0 - \sum_{i=1}^{n} p_i d_i \right) + \left(\mu^2 p_0^2 - \sum_{i=1}^{n} p_i^2 \right) \geq 0 \\ p_0 + \alpha d_0 \geq 0 \end{cases}$$

where the unknown variable α has been grouped in order to obtain

$$\begin{aligned} a_\alpha &= \mu^2 d_0^2 - \sum_{i=1}^{n} d_i^2 = \boldsymbol{d}^T J \boldsymbol{d} \\ k_\alpha &= \mu^2 p_0 d_0 - \sum_{i=1}^{n} p_i d_i = \boldsymbol{p}^T J \boldsymbol{d} \\ c_\alpha &= \mu^2 p_0^2 - \sum_{i=1}^{n} p_i^2 = \boldsymbol{p}^T J \boldsymbol{p} \end{aligned}$$

where, for this general case, we have

$$J = \begin{bmatrix} \mu^2 & 0 \\ 0 & -I_{[p-1]} \end{bmatrix}$$

The obvious solutions for the first equation of section 4.14.3 are

$$\alpha_{m,M} = \frac{-k_\alpha \pm \sqrt{k_\alpha^2 - a_\alpha c_\alpha}}{a_\alpha} \tag{4.53}$$

where $\alpha_m <= \alpha_M$. That gives the following conditions

$$\begin{cases} a_\alpha > 0 & \Longrightarrow \quad \alpha_m \leq \alpha \leq \alpha_M \\ a_\alpha < 0 & \Longrightarrow \quad \alpha \leq \alpha_m \vee \alpha \geq \alpha_M \end{cases}$$

while for second equation of section 4.14.3, naming $\alpha_b = -\frac{p_0}{d_0}$

$$\begin{cases} d_0 > 0 & \Longrightarrow \quad \alpha \geq \alpha_b \\ d_0 < 0 & \Longrightarrow \quad \alpha \leq \alpha_b \end{cases}$$

In table 4.1 and table 4.2 the different combinations are summarized, while in table 4.3 the existence conditions are stated.

Table 4.1: *Infimum step length* α

	$a_\alpha > 0$	$a_\alpha < 0$
$d_0 > 0$	$\max\{\alpha_M, \alpha_b\}$	$\max\{\alpha_m, \alpha_b\}$
$d_0 > 0$	$-\infty$	α_m

Table 4.2: *Supremum step length* α

	$a_\alpha > 0$	$a_\alpha < 0$
$d_0 > 0$	$+\infty$	α_M
$d_0 > 0$	$\min\{\alpha_m, \alpha_b\}$	$\min\{\alpha_M, \alpha_b\}$

These considerations are particularly useful in order to compute the minimum or the maximum length of the step that is necessary to reach the border of the cone or to not exit from it.

It is notable that, for the search of the supremum, only the solution of eq. (4.53) with the minus sign is required, and conversely the solution with the plus sign is the only one required for the infimum search. This is because the change of sign in a_α.

An interesting sub-case is when the direction of the step is along the axis of the cone i.e. $\{d_0 = 1, d_i = 0 \forall i \neq 0\}$. In that case

$$a_\alpha = \mu^2 > 0 \tag{4.54a}$$

$$d_0 = 1 > 0 \tag{4.54b}$$

$$\alpha_M = -p_0 + \frac{\|\boldsymbol{p}_1\|}{\mu} \tag{4.54c}$$

and, if $\mu = 1$ then the infimum limit is given by α_M since $\alpha_b \leq \alpha_M$. For our purpose

Table 4.3: *Existence conditions for* α

	$a_\alpha > 0$	$a_\alpha < 0$
$d_0 > 0$	-	$\alpha_M \geq \alpha_b$
$d_0 > 0$	-	$\alpha_b \geq \alpha_m$

the cones are all with $\mu = 1$, so we can simplify some of the operations above.

Another approach, is the one suggested by Vandenberghe [40]. Given $\Delta\tilde{\boldsymbol{y}} = W^{-T}\Delta\boldsymbol{y}$ and $\Delta\tilde{\boldsymbol{\lambda}} = W\Delta\boldsymbol{\lambda}$, requiring that

$$\boldsymbol{z} + \alpha\Delta\tilde{\boldsymbol{y}} \succeq 0 \qquad \boldsymbol{z} + \alpha\Delta\tilde{\boldsymbol{\lambda}} \succeq 0$$

Re-scaling the cones, we get the following criteria for the selection of the step length. First, we introduce the scaled directions

$$\boldsymbol{\rho}_i = \nabla\phi(\boldsymbol{z}_i)_i^{1/2}\Delta\tilde{\boldsymbol{y}}_i \qquad \boldsymbol{\sigma}_i = \nabla\phi(\boldsymbol{z}_i)_i^{1/2}\alpha W\Delta\tilde{\boldsymbol{\lambda}}_i$$

that are used to compute α

$$\alpha = \min\left(\sup\left(\alpha | \boldsymbol{e}_i + \alpha\boldsymbol{\rho}_i, \boldsymbol{e}_i + \alpha\boldsymbol{\sigma}_i\right)\right)$$

For the NNC we get

$$\boldsymbol{\rho}_i = \Delta\tilde{\boldsymbol{y}} \circ \boldsymbol{z}_i^{-1} \qquad \boldsymbol{\sigma}_i = \Delta\tilde{\boldsymbol{\lambda}} \circ \boldsymbol{z}_i^{-1} \tag{4.55a}$$

$$\alpha = \min\left\{\max\left\{0, -\min\left(\boldsymbol{\rho}_i\right), -\min\left(\boldsymbol{\sigma}_i\right)\right\}\right\} \tag{4.55b}$$

while, for the CC,

$$\boldsymbol{\rho}_i = \frac{1}{(\boldsymbol{z}_i^T J \boldsymbol{z}_i)^{1/2}} \begin{bmatrix} \bar{\boldsymbol{z}}_i^T J\Delta\tilde{\boldsymbol{y}}_i \\ \Delta\tilde{\boldsymbol{y}}_{i,1} - \frac{\bar{\boldsymbol{z}}_i^T J\Delta\tilde{\boldsymbol{y}}_i + \Delta\tilde{y}_{i,0}}{\bar{z}_{i,0}^T + 1}\bar{\boldsymbol{z}}_{i,1} \end{bmatrix} \tag{4.56a}$$

$$\boldsymbol{\sigma}_i = \frac{1}{(\boldsymbol{z}_i^T J \boldsymbol{z}_i)^{1/2}} \begin{bmatrix} \bar{\boldsymbol{z}}_i^T J\Delta\tilde{\boldsymbol{\lambda}}_i \\ \Delta\tilde{\boldsymbol{\lambda}}_{i,1} - \frac{\bar{\boldsymbol{z}}_i^T J\Delta\tilde{\boldsymbol{\lambda}}_i + \Delta\tilde{\lambda}_{i,0}}{\bar{z}_{i,0}^T + 1}\bar{\boldsymbol{z}}_{i,1} \end{bmatrix} \tag{4.56b}$$

$$\alpha = \min\left\{\max\left\{0, -\rho_{i,0} + \|\boldsymbol{\rho}_{i,1}\|, -\sigma_{i,0} + \|\boldsymbol{\sigma}_{i,1}\|\right\}\right\} \tag{4.56c}$$

with $\bar{\boldsymbol{z}}_i = \frac{\boldsymbol{z}_i}{(\boldsymbol{z}_i^T J \boldsymbol{z}_i)^{1/2}}$.

4.14.4 Starting Point for Cone QP

Even in the Cone variant of the Interior-Point solver we are asked to start the algorithm providing a starting point that is feasible. In order to accomplish this task a

two-step procedure is followed, partially taking a cue from the CVXOPT by Vandenberghe [40].

The first step is to obtain a raw estimate for the variables $\boldsymbol{v}$ $\boldsymbol{\gamma}$ e $\boldsymbol{\lambda}$ by means of the system

$$\begin{bmatrix} D & -C^T & -D^T \\ C & 0 & 0 \\ D & 0 & I \end{bmatrix} \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{\gamma} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{c} \\ \boldsymbol{b}_\gamma \\ \boldsymbol{b}_\lambda \end{bmatrix} \tag{4.57}$$

that gives the optimality conditions for

$$\begin{aligned} \min \quad & \frac{1}{2}\boldsymbol{v}^T H \boldsymbol{v} + \boldsymbol{v}^T \boldsymbol{c} + \frac{1}{2}\boldsymbol{y}^T \boldsymbol{y} \\ \text{s.t.} \quad & C\boldsymbol{v} - \boldsymbol{b}_\gamma = \boldsymbol{0} \\ & D\boldsymbol{v} - \boldsymbol{b}_\lambda - \boldsymbol{y} = \boldsymbol{0} \end{aligned}$$

then $\boldsymbol{y} = -\boldsymbol{\lambda}$ so that the two resides in their dual cones. However, this first step does not guarantee that neither $\boldsymbol{y} \succeq 0$ or that $\boldsymbol{\lambda} \succeq 0$, so the second step occurs.

In order to push the two inequality-constrained variables into their respective cones (generally speaking, hence also the non-negativity constraint has its cone that is degenerated in a half-space) a step has to be made. The choice is to make the step in the direction of the cone axis: for this purpose, we are looking for the *minimum* step that guarantees that the whole variable resides in the cone. The informations gathered in section 4.14.3 are here used to find this value. So, after having stored

$$\boldsymbol{y} \leftarrow -\boldsymbol{\lambda}$$

we make sure that both the variables are in their respective cones. Thus, we find

$$\begin{aligned} \alpha_P = \min\{\alpha | \boldsymbol{y} + \alpha \boldsymbol{e} \succeq 0\}, \qquad & \alpha_P > 0 \implies \boldsymbol{y} \mathrel{+}= (\eta + \alpha_P)\boldsymbol{e} \\ \alpha_D = \min\{\alpha | \boldsymbol{\lambda} + \alpha \boldsymbol{e} \succeq 0\}, \qquad & \alpha_D > 0 \implies \boldsymbol{\lambda} \mathrel{+}= (\eta + \alpha_D)\boldsymbol{e} \end{aligned}$$

where $\eta \in (0, 1]$ is a safe-guard value that pushes the variable to be not just on the border, but with a margin of $\eta \boldsymbol{e}$.

Chapter 5

Results

The Interior-Point for frictionless contact has been implemented in C++ language and plugged into the Chrono::Engine open-source software. Thanks to an already existent dynamic framework, the effort has been restricted to the design of the solver itself, leaving what concerns the dynamic of the bodies to the upper layers of the program.

The hardware on which the tests are performed is a notebook computer equipped with an Intel i7-6700HQ processor, 2x8GB DDR4 RAM, with SSD drive.

The iteration loop exits when *all* the following conditions are met

$$\frac{\|\boldsymbol{r}_{p_\gamma}\|}{n_\gamma} < 1 \times 10^{-10}$$
$$\frac{\|\boldsymbol{r}_{p_\lambda}\|}{n_\lambda} < 1 \times 10^{-10}$$
$$\frac{\|\boldsymbol{r}_d\|}{n_v} < 1 \times 10^{-10}$$
$$\mu < 1 \times 10^{-9}$$

5.1 Warm start

As previously mentioned in section 4.6, since this algorithm is planted into a dynamic simulation framework, we were able to try a *warm start*, that is: recycle the solution

from the previous time step to fed the first iteration of the following IP loop. Depending on the variations that the system underwent during the two different instants of time, two possible warm start techniques have been applied.

Full Warm Start the solution vector is used directly and entirely to feed the next IP iteration; this is a very rare situation, since different conditions should be met contemporaneously. See section 4.6;

Partial Warm Start only parts of the solution vector are reused, depending on which matrix between H, C and D preserved its size; however, matrices dimensions do not guarantee that the past solution is still a good starting point for the new time-step. In these tests only the $\boldsymbol{v}$ has been reused: this is normally a good choice since the bodies are not removed/added during the simulation and are not affected by deep changes.

Because of the rapid, continuous and deep modifications that affect the Jacobian matrices C and especially D, is quite difficult to find a real-case scenario in which *full* warm start may be used. In fact, at least the contact points may change quickly, wasting any opportunity to reuse the $\boldsymbol{\lambda}$ vector.

A first test (Balls in the Box section 5.3.1) has been run for this purpose, together with even more trivial examples in order to test the effectiveness of the warm start method. In table 5.1, for "Single Ball" and "Single Brick" we intend extremely basic benchmarks in which respectively a sphere and a brick has been put on a flat surface. We also note that, for the brick case, redundant constraints are present, since four reactions are added by the Chrono::Engine environment.

5.2 Softened Constraints

During the tests, the MUMPS linear solver has been trained to intervene if redundant rows would have been caught: in none of the benchmarks this has been experienced so it has not been necessary to manipulate the system matrix with more advanced SVD or QR rank-revealing methods. Also the contact softening technique has not

Test (IP calls)	None	Partial Warm Start	Full Warm Start
Balls in Box (50)	935	927	747
Single Ball (100)	720	515	90
Single Brick (100)	499	370	81

Table 5.1: *IP iterations for solving unilateral-only contact problems with different warm start techniques.*

been applied in the next set of tests since the matrices did not show any strong ill-conditioning.

5.3 Convergence and Speed Results

5.3.1 Balls in the Box

A set of tests has been performed in order to the test the performance of the method. In this case, a rigid fixed container is filled with smaller spheres whose number is varied from hundreds to thousands. The expectancy is to have a number of iterations that does not change as the size of the problem increases. No bilateral joints are used in this simulation. In table 5.2 we present the parameters of the simulation, where the number of spheres is kept variable and it is specified in table 5.3, together with the related number of iterations required.

In our experience, one of the most critical elements for an effective IP solver is the choice of the starting point and the criteria to select the step lengths. Both of them heavily affect the convergence speed and robustness.

5.3.2 Three Point Bending

As a comparison between smooth/penalty and rigid (MDI) contacts is proposed here, using a three point bending test. This benchmark offers also the occasion to introduce finite elements in the simulation, in this case tetrahedrons, together with rigid bodies. These flexible elements introduce sparse non-diagonal terms $h^2\nabla_q \boldsymbol{f}^{(l+1)}$ and $h\nabla_v \boldsymbol{f}^{(l+1)}$ in the H matrix, that restricts the choice of a viable solver. The FE solids are corotational tetrahedrons that accept large displacements with linear elastic material.

Two fixed cylinders are supporting a beam, made of flexible elements, while a

Data	Value
Container Diameter	1 m
Spheres Diameter	70 mm
Spheres Density	$1000\,\mathrm{kg/m^3}$

Table 5.2: *Balls in the Box Parameters.*

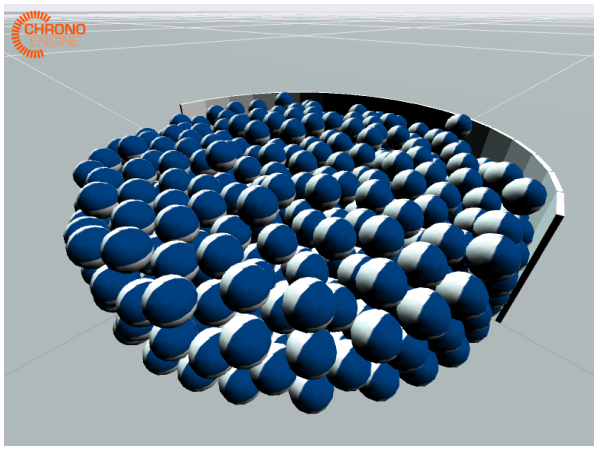


Figure 5.1: *Balls in the Box Test.*

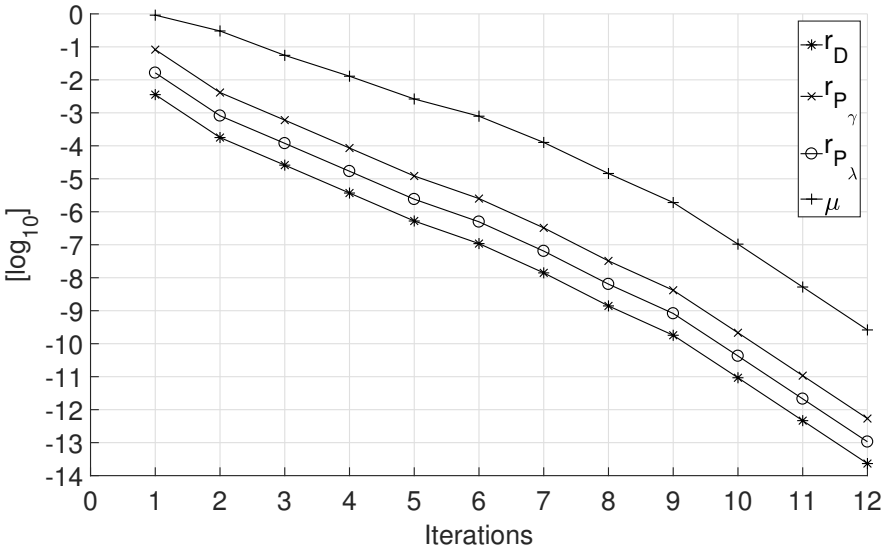


Figure 5.2: *Balls in the Box Test: convergence* ($\Delta t = 0.02\,\mathrm{s}$).

Total DOF	n_v	n_c	Average Iterations (50 steps)
23938	7140	16798	17.8
11871	3570	8301	17.7
5558	1680	3878	17.9
1268	384	884	17.2

Table 5.3: *Iterations needed at each time step for different matrix sizes (Balls in the Box example) ($\Delta t = 0.02\,\mathrm{s}$).*

Data	Value
FE Mesh Size	$26 \times 12 \times 3$
FE Young Modulus	0.01 GPa
FE Poisson	0.3
FE Density	$1000\,\mathrm{kg/m^3}$
Cylinder Diameter	0.05 m
Cylinder Horizontal Distance	1 m

Table 5.4: *Three Point Bending Parameters.*

third is moving down at a speed of $-1\,\mathrm{m/s}$, driven by a rheonomic constraint. The test is performed pushing the time step to its higher value, in compatibility with the given contact method. The extreme performances of the MDI approach are justified by a longer time step that, in particular, is not affected by the stiffness of the elements involved in the simulation, like the smooth/penalty method would have been. The stability of the MDI has been reported up to 0.06 s, while the penalty approach required steps shorter than 0.005 s even with implicit integrators.

In table 5.4 the list of parameters, and in fig. 5.4 the convergence results.

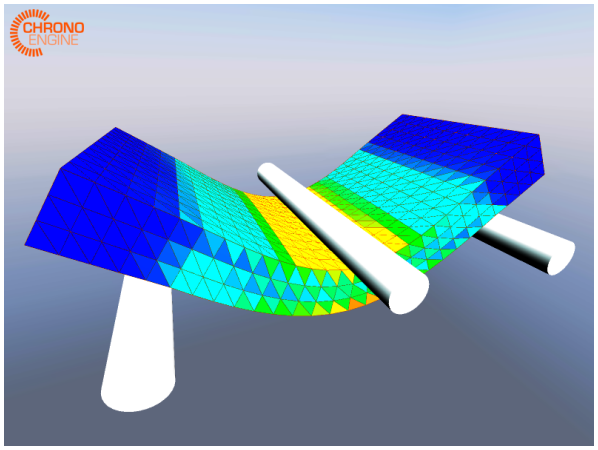


Figure 5.3: *Three Point Bending Test.*

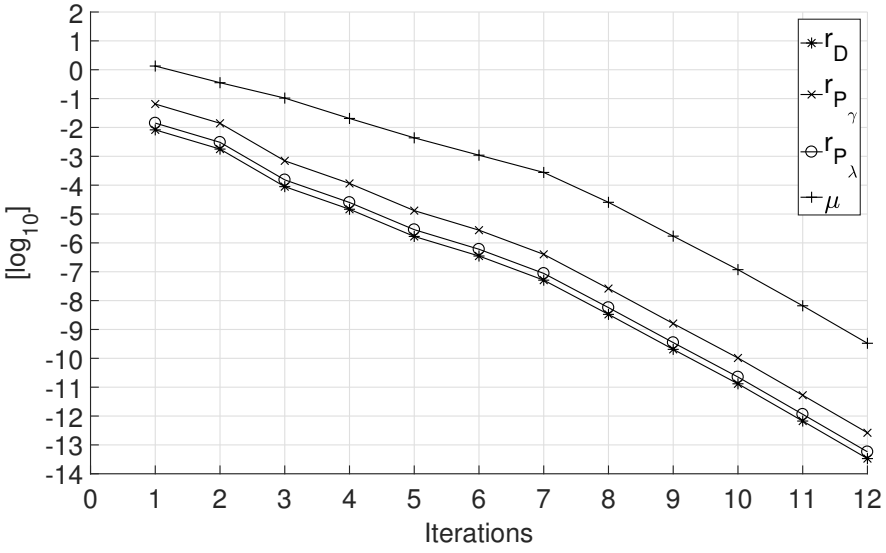


Figure 5.4: *Three Point Bending Test: convergence* ($\Delta t = 0.05\,\mathrm{s}$).

5.3.3 Stacked Sphere - Odd Mass Ratio

One of the critical issues that the solver should address is the case where bodies with very different masses and scale co-exists in the same simulation system. The odd ratio between masses goes to increase the already sensible difference between masses and inertia tensors that reside in the H matrix. In order to push this oddity to the extreme, we set up a critical scenario in which a stack of spheres with various masses are forced to stay one on the top of the others: this is accomplished by constraining each sphere with a bilateral joint that forces its center of mass to move along a vertical axis. In this situation, the Jacobian of the constraints C and D should be quite similar through the different test configurations. In fact each body should have a translational constraints that limits two degrees of freedom plus, at most, two unilateral constraints.

Four different configurations are investigated: the first three have sphere with masses of respectively 100 kg, 1×10^6 kg and 1×10^{12} kg. So, in each test, the spheres shares the same mass within the stack. The density of the material is kept constant, thus the increase of mass is obtained through the change of diameter. As we can see in fig. 5.6, the number of required iterations changes with the spheres weight, not with number of sphere stacked.

The fourth test sees a stack in which each spheres has a different mass compared to the others. In particular, each sphere has, on its top, a sphere that is ten times heavier than itself; so the ratio between each pair of spheres is always ten. However, one can easily see that there is a exponential progression: the j-th sphere is 10^j times heavier that the first one. This introduces a series of issues, not only linked with sensible difference in the diagonal of the H matrix, but also in the solution of the Lagrangian Multipliers regarding the unilateral joints $\boldsymbol{\lambda}$. Its components, in fact, may replicate the same mass oddity, having to describe reaction forces that are different by orders of magnitude. And remember: this tests share the same D and C. This pushes the algorithm and, most of all, the linear solver to the limit.

In this latter test, the number of iterations grows linearly with the exponent of the sphere mass: the increasing mass and reaction forces ratios ask the IP for more iterations. Moreover, we were able to stack a huge number of spheres for the equal-mass test, but only fifteen for the odd mass-ratio scenario before meeting instability.

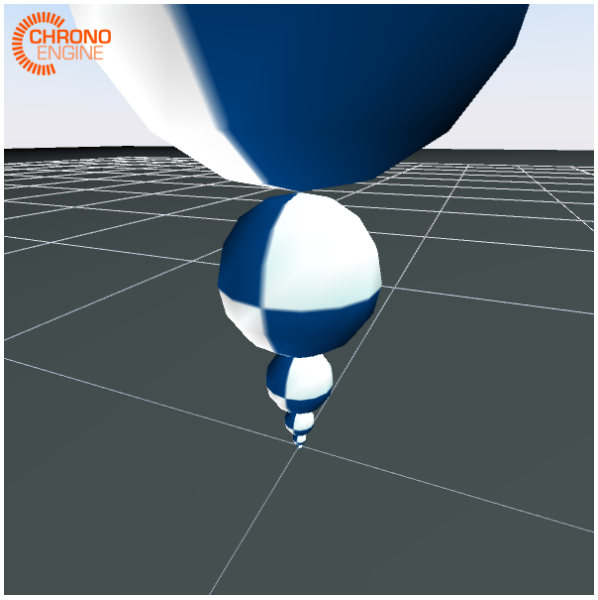


Figure 5.5: *Stacked Spheres Test.*

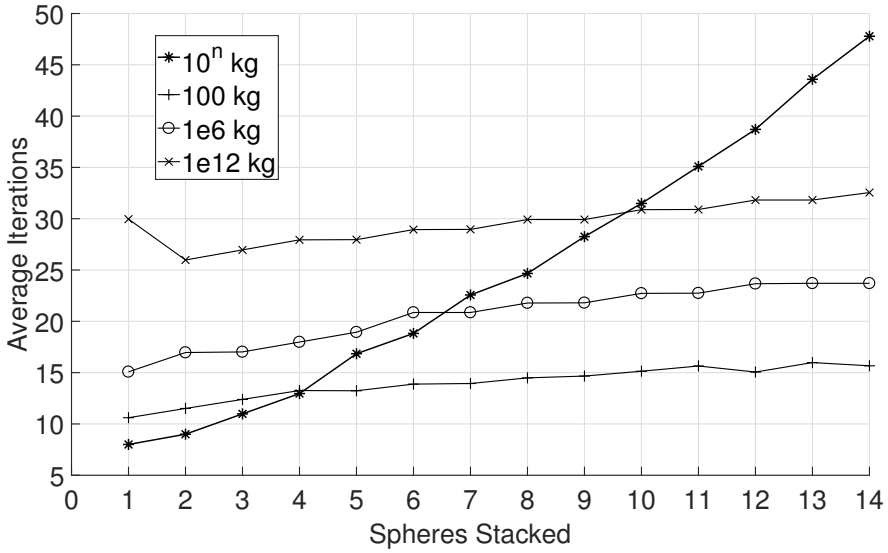


Figure 5.6: *Stacked Spheres Test: convergence ($\Delta t = 0.05\,\mathrm{s}$). The 10^n kg legend entry refers to the test in which the n-th sphere weights 10^n kg.*

Chapter 6

Conclusions

The thesis showed a possible and effective implementation of an Interior-Point Method specifically built upon an MDI framework applied to multibody non-smooth dynamic problems. The algorithm, developing the scheme suggested by Mehrotra [24], was tailored to work together with this particular mechanical problem, but it is suited to provide interesting results also in other fields in which similar problems structure may arise. The method showed good performances for the frictionless case and aims to the get the same remarkable results for the friction case.

The C++ implementation of the Interior-Point algorithm permitted to simulate concurrent rigid and flexible bodies simulations, using the Chrono::Engine open-source library, opening new perspectives and possibilities. Interface and matrix classes are developed too, in order to let the library communicate with third-party softwares like Intel MKL and MUMPS, in order to provide the required linear solvers.

The analytical description of the multibody system was rewritten, following the criteria adopted in [36], in order to accommodate the discontinuous nature of impulsive events, allowing the simulation of rigid contacts i.e. without the introduction of fictitious stiffness between contact points. This new formulation permitted not only to describe them in a more rigorous fashion, but also allowed to relax some unnecessary strict time steps, regaining the possibility to run bigger-scale problems in shorter time. This new formulation fits perfectly in the proposed Interior-Point scheme.

The required mathematical backgrounds are widely discussed, as well as the optimality conditions for the problems in use. In particular, the essay included the required tools for the novel multibody dynamic formulation, such as measures and the Differential Variational Inequalities theory and the optimality requirements that lead to the formulation of the Interior-Point system.

A brief introduction to the friction case is proposed both from the multibody perspective and from the solver side: the development of the Interior-Point for Cone QP, still in its early stages, is ongoing and some preliminary results might be available, but it has been chosen to not report them in this context, since their experimental nature. Nevertheless, the C++ code for Cone QP has been presented in appendix B.

The challenging task to provide an efficient solver for multibody non-smooth dynamics for contacts *with friction* is, for sure, the next objective of the research. Meanwhile, a well-performing algorithm for the frictionless case is already available and tested, and aspires to be distributed to end-users.

Appendix A

Mathematical Concepts

A.1 Gradient

Gradient (where not otherwise specified) should be intended as

$$\nabla \boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial \boldsymbol{f}}{\partial x_1} & \cdots & \frac{\partial \boldsymbol{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

$$\nabla(\boldsymbol{u}^T \boldsymbol{v}) = \begin{bmatrix} \frac{\partial \boldsymbol{u}^T \boldsymbol{v}}{\partial \boldsymbol{u}} & \frac{\partial \boldsymbol{u}^T \boldsymbol{v}}{\partial \boldsymbol{v}} \end{bmatrix}_{[1 \times 2n]} = \begin{bmatrix} \boldsymbol{v}^T & \boldsymbol{u}^T \end{bmatrix}$$

A.2 Hadamard Product

Property of the derivative of the Hadamard product (valid both for non-negative orthant and second order cones):

$$\frac{\partial(\boldsymbol{u} \circ \boldsymbol{v})}{\partial \boldsymbol{u}} \boldsymbol{x} = \boldsymbol{v} \circ \boldsymbol{x} \tag{A.1a}$$

$$\frac{\partial(A\boldsymbol{u} \circ \boldsymbol{v})}{\partial \boldsymbol{u}} \boldsymbol{x} = \boldsymbol{v} \circ A\boldsymbol{x} \tag{A.1b}$$

The Hadamard product $u \circ v$ can be rewritten as normal product operator:

$$\begin{aligned} \boldsymbol{u} \circ \boldsymbol{v} &= P_H(u)\, v \\ &= \operatorname{diag}(\boldsymbol{u})\, \boldsymbol{v} && \text{for } \mathbb{R}_+ \\ &= \begin{bmatrix} u_0 & \boldsymbol{u}_1^T \\ \boldsymbol{u}_1 & u_0 I \end{bmatrix} \boldsymbol{v} && \text{for } \mathscr{K} \end{aligned}$$

from which it is easy to derive their inverse operator $u \diamond v$ simply inverting the $P_H(u)$ matrix.

Appendix B

C++ Code

In this appendix a cut version of the C++ code used to run the tests of chapter 5. This implementation already includes the Cone QP framework: obviously this part has not been used in the examples; in fact, the software recognizes the presence of cone constraints.

For the full implementation, please checkout the Chrono fork at `https://github.com/dariomangoni/chrono.git`.

Interior-Point C++ code.

```
// =============================================================================
// PROJECT CHRONO - http://projectchrono.org
//
// Copyright (c) 2014 projectchrono.org
// All rights reserved.
//
// Use of this source code is governed by a BSD-style license that can be found
// in the LICENSE file at the top level of the distribution and at
// http:///projectchrono.org/license-chrono.txt.
//
// =============================================================================
// Authors: Dario Mangoni
// =============================================================================

#include "ChInteriorPoint.h"
#include "ChInteriorPointUtils.h"
#include "solver/ChConstraintTwoTuplesFrictionT.h"
#include <algorithm>
#include <iomanip>

#ifdef CHRONO_POSTPROCESS
#include "chrono_postprocess/ChGnuPlot.h"
#endif

#define DEBUG_TRAPS
#define DEBUG_CONSOLE
#define CORRECT_STARTING_POINTS
#define ADD_COMPLIANCE false
#define REUSE_OLD_SOLUTIONS false
#define USE_MY_MAX_STEPLENGTHS
//#define BYPASS_RESTITUTION_TERM
```

```
//TODO: drive iterations toward the term that has not reached the tolerated level yet

namespace chrono {


double ChInteriorPoint::Solve(ChSystemDescriptor& sysd) {
    solver_call++;

    ip_timer_solve_assembly.start();
    /********** Load system **********/
    // for optimization of the starting point

    // set offsets so that equality constraints come first
    sysd.SortActiveConstraints();

    // get matrices size and identify the constraint mode for inequality constraints
    auto constr_list = sysd.GetConstraintsList();
    n = sysd.CountActiveVariables();
    m_eq = 0;
    auto m_ineq_tot = 0;
    auto nnz_scaling = 0;
    ineq_mode.clear();
    for (auto it = 0; it<constr_list.size(); ++it)
    {
        switch(constr_list[it]->GetMode())
        {
        case eChConstraintMode::CONSTRAINT_FREE:
            ++m_eq; break;
        case eChConstraintMode::CONSTRAINT_LOCK:
            ++m_eq; break;
        case eChConstraintMode::CONSTRAINT_UNILATERAL:
            ineq_mode.push_back(eChConstraintModeMOD::CONSTRAINT_UNILATERAL);
            nnz_scaling += 1;
            break;
        case eChConstraintMode::CONSTRAINT_FRIC:
            ++m_ineq_tot;
            if (!dynamic_cast<ChConstraintTwoTuplesFrictionTall*>(constr_list[it]))
            {
                ineq_mode.push_back(eChConstraintModeMOD::CONSTRAINT_FRIC_N);
                nnz_scaling += 1;
            }
            else
            // we assume that CONSTRAINT_FRIC has only N and UV type of constraints, otherwise if (
                  ↪ enable_friction_cones && dynamic_cast<ChConstraintTwoTuplesFrictionTall*>(
                  ↪ constr_list[it]))
            if (!skip_contacts_uv)
            {
                ineq_mode.push_back(eChConstraintModeMOD::CONSTRAINT_FRIC_UV);
                nnz_scaling += 4;
            }
            break;
        default:
            assert(0);
        }
    }
    m_ineq = static_cast<int>(ineq_mode.size());

    // scale variables
    var.Reset(n, m_eq, m_ineq);
    res.Reset(n, m_eq, m_ineq);
    yl_scaled.Reset(m_ineq, 1);

    rhs.Resize(n, m_eq, m_ineq);


    // update mutables
    vectn.Resize(n, 1);
    vectm_eq.Resize(m_eq, 1);
    vectm_ineq.Resize(m_ineq, 1);
    sol_chrono.Resize(n + m_eq + m_ineq_tot, 1);

    // Let the matrix acquire the information about ChSystem
    if (m_force_sparsity_pattern_update)
    {
        m_force_sparsity_pattern_update = false;

        ChSparsityPatternLearner sparsity_learner(n + m_eq + m_ineq, n + m_eq + m_ineq, true);
        sysd.ConvertToMatrixForm(&sparsity_learner, nullptr);
        for (auto cs = 0; cs < ineq_mode.size(); ++cs)
        {
            if (ineq_mode[cs] == eChConstraintModeMOD::CONSTRAINT_UNILATERAL || ineq_mode[cs] ==
                  ↪ eChConstraintModeMOD::CONSTRAINT_FRIC_N && skip_contacts_uv)
            {
                sparsity_learner.SetElement(cs, cs, 0.0);
                continue;
            }
```

```
            if (ineq_mode[cs] == eChConstraintModeMOD::CONSTRAINT_FRIC_N)
            {
                sparsity_learner.SetElement(cs+0, cs+0, 0.0);
                sparsity_learner.SetElement(cs+1, cs+0, 0.0);
                sparsity_learner.SetElement(cs+2, cs+0, 0.0);
                sparsity_learner.SetElement(cs+0, cs+1, 0.0);
                sparsity_learner.SetElement(cs+1, cs+1, 0.0);
                sparsity_learner.SetElement(cs+2, cs+1, 0.0);
                sparsity_learner.SetElement(cs+0, cs+2, 0.0);
                sparsity_learner.SetElement(cs+1, cs+2, 0.0);
                sparsity_learner.SetElement(cs+2, cs+2, 0.0);
            }
        }

        BigMat.LoadSparsityPattern(sparsity_learner);
    }
    else
    {
        // If an NNZ value for the underlying matrix was specified, perform an initial resizing, *
            ↪ before*
        // a call to ChSystemDescriptor::ConvertToMatrixForm(), to allow for possible size
            ↪ optimizations.
        // Otherwise, do this only at the first call, using the default sparsity fill-in.
        if (ip_solver_call == 0) {
            BigMat.Reset(n + m_eq + m_ineq, n + m_eq + m_ineq, static_cast<int>((n + m_eq + m_ineq) *
                ↪ ((n + m_eq + m_ineq) * SPM_DEF_FULLNESS)));
        }
    }

    sysd.ConvertToMatrixForm(&BigMat, nullptr, false, skip_contacts_uv, ADD_COMPLIANCE);

    if (!leverage_symmetry)
        make_positive_definite();

    sysd.ConvertToMatrixForm(nullptr, nullptr, nullptr, &rhs.c, &rhs.b, nullptr, false,
        ↪ skip_contacts_uv);  // load f->c and b->b
    rhs.c.MatrScale(-1); // adapt to InteriorPoint convention
    rhs.b.MatrScale(-1); // adapt to InteriorPoint convention

    ip_timer_solve_assembly.stop();
#ifdef BYPASS_RESTITUTION_TERM
    rhs.b.FillElem(0);
#endif

    /********** Check if system has inequality constraints **********/
    if( m_ineq == 0 )  // if no inequality constraints
    {
        ChMatrixDynamic<double> mumps_rhs(n + m_eq, 1);

        // Fill 'mumps_rhs' with just Chrono's 'f' i.e. IP's '-c'
        for( auto row_sel = 0; row_sel < n; row_sel++ )
            mumps_rhs.SetElement(row_sel, 0, -rhs.c.GetElement(row_sel, 0));
        for (auto row_sel = 0; row_sel < m_eq; row_sel++)
            mumps_rhs.SetElement( n + row_sel, 0, rhs.b.GetElement(row_sel, 0));

        // Solve the KKT system
        BigMat.Compress();
        mumps_engine.SetProblem(BigMat, mumps_rhs);
        if( mumps_engine.MumpsCall(ChMumpsEngine::COMPLETE) )
            mumps_engine.PrintINFOG();

        if( verbose && mumps_engine.GetRINFOG(6) > 1e-6 )
            std::cout << "MUMPS scaled residual: " << mumps_engine.GetRINFOG(6) << std::endl;

        if (leverage_symmetry) // then the gamma have flipped sign
            for (auto row_sel = 0; row_sel < m_eq; row_sel++)
                mumps_rhs(n + row_sel,0) *= -1;

        sysd.FromVectorToUnknowns(mumps_rhs);

        // Export variable so that can be used in the next iteration as starting point
        var.v.Resize(n, 1);
        var.gamma.Resize(m_eq, 1);
        for (auto row_sel = 0; row_sel < n; row_sel++)
            var.v.SetElement(row_sel, 0, mumps_rhs.GetElement(row_sel, 0));
        for (auto row_sel = 0; row_sel < m_eq; row_sel++)
            var.gamma.SetElement(row_sel, 0, leverage_symmetry ? -mumps_rhs.GetElement(n + row_sel, 0)
                ↪  : mumps_rhs.GetElement( n + row_sel, 0));

        if( verbose )
            std::cout << "IP call: " << solver_call << "; No inequality constraints." << std::endl;

        return 0.0;
    }

    /********* The system DOES have constraints! Start Interior Point ********/
```

```
    ip_solver_call++;
    ip_timer_solver_solvercall.start();

    if (m_eq > 0)
    {
        rhs.b_eq.PasteClippedMatrix(rhs.b, 0, 0, m_eq, 1, 0, 0);
        rhs.b_ineq.PasteClippedMatrix(rhs.b, m_eq, 0, m_ineq, 1, 0, 0);
    }
    else
        rhs.b_ineq = rhs.b;

    PrintCVXOPTproblem(sysd, "cvxopt_problem");


    if( ADD_COMPLIANCE && m_ineq > 0 )
    {
        sysd.ConvertToMatrixForm(nullptr, nullptr, &E, nullptr, nullptr, nullptr, false,
            ↪ skip_contacts_uv);
        E *= -1;
    }

    set_feasible_starting_point();

    if (verbose)
    {
        std::cout << "IP " << solver_call << "| n: " << n << "; m_eq: " << m_eq << "; m_ineq: " <<
            ↪ m_ineq << std::endl;
        std::cout << "       || *** prediction *** ||      || correction  ||" << std::endl;
        std::cout << "  | it || a_pr | a_du |  mu  || tau  || a_pr | a_du ||  mu  |    resP    |
            ↪ res_eq   |  res_ineq  |" << std::endl;
    }

    for( iteration_count = 0; iteration_count < iteration_count_max; iteration_count++ )
    {

        iterate();
        iteration_count_tot++;


        if( res <= res_nnorm_tol ) // check for exit conditions (less OR EQUAL in order to catch zero
            ↪ residual (for example, no bilateral))
        {
            if (verbose)
                PrintIPStatus();
            break;
        }
    }

    ip_timer_solver_solvercall.stop();

    // Scatter the solution into the Chrono environment
    sysd.FromVectorToUnknowns(adapt_to_Chrono(sysd, sol_chrono));

    return 0.0;
}

// Iterating function
void ChInteriorPoint::iterate() {
    /*****************************************************************************/
    /***************************** Prediction Phase ******************************/
    /*****************************************************************************/

    //DumpProblem("pre");
    // Paste scaling matrix W^T * W
    computeNesterovToddScalingMatrix(var);
    scaling_matrix.MatrMultiplyClipped(scaling_matrix, BigMat, 0, m_ineq - 1, 0, m_ineq - 1, 0, n +
        ↪ m_eq, true, 0, 0, n + m_eq);
    if (ADD_COMPLIANCE)
        for (auto diag_sel = 0; diag_sel < m_ineq; ++diag_sel)
            BigMat.SetElement(n + m_eq + diag_sel, n + m_eq + diag_sel, leverage_symmetry ? -E.
                ↪ GetElement(diag_sel - n + m_eq, diag_sel - n + m_eq) : +E.GetElement(diag_sel - n
                ↪ + m_eq, diag_sel - n + m_eq), false);
    factorize_system_matrix();
    scaling_matrix.MatrMultiply(var.lambda, yl_scaled);

    DumpProblem("post");

#ifdef DEBUG_TRAPS
    assert(is_valid(yl_scaled, m_ineq));
    assert(is_valid(var.y, m_ineq));
    assert(is_valid(var.lambda, m_ineq));
#endif

    // WARNING: the residual structure 'res' must be already updated at this point!

    // fill 'mumps_rhs' with rhs [-res.rd;-res.rp_gamma;-res.rp_lambda-y]
    mumps_rhs.Resize(n + m_eq + m_ineq, 1);
```

```
    for( auto row_sel = 0; row_sel < n;      row_sel++ ) mumps_rhs(row_sel           ) = -res.rd(
        ↪ row_sel);
    for( auto row_sel = 0; row_sel < m_eq;   row_sel++ ) mumps_rhs(row_sel + n       ) = -res.rp_gamma
        ↪ (row_sel);
    for( auto row_sel = 0; row_sel < m_ineq; row_sel++ ) mumps_rhs(row_sel + n + m_eq) = -res.
        ↪ rp_lambda(row_sel) - var.y(row_sel);

    Dvar_pred.Resize(n, m_eq, m_ineq);
    get_Newton_direction(Dvar_pred, mumps_rhs, res);

#ifdef DEBUG_TRAPS
    assert(is_valid(Dvar_pred.y, m_ineq));
    assert(is_valid(Dvar_pred.lambda, m_ineq));
#endif

    /*** compute step lengths ***/
    // from 16.60 pag.482 from 14.32 pag.408 (remember that y>=0!)
#ifdef USE_MY_MAX_STEPLENGTHS
    auto alfa_pred_prim = get_Newton_steplength_MAX_mod(var.y, Dvar_pred.y);
    auto alfa_pred_dual = get_Newton_steplength_MAX_mod(var.lambda, Dvar_pred.lambda);
#else
    auto alfa_pred_prim = get_Newton_steplength_MAX(var.y, Dvar_pred.y);
    auto alfa_pred_dual = get_Newton_steplength_MAX(var.lambda, Dvar_pred.lambda);
#endif

#ifdef DEBUG_CONSOLE
    printf("  |%4d", iteration_count);
    printf("||%-6.3g|%-6.3g", alfa_pred_prim, alfa_pred_dual);
    //printf("|%-6d|%-6d|", find_out_of_cone(var.y, Dvar_pred.y, alfa_pred_prim), find_out_of_cone(var
        ↪ .lambda, Dvar_pred.lambda, alfa_pred_dual));
#endif

    if( equal_step_lengths )
    {
        auto alfa_pred = std::min(alfa_pred_prim, alfa_pred_dual);
        alfa_pred_prim = alfa_pred;
        alfa_pred_dual = alfa_pred;
    }


    /*** make the prediction step ***/
    // update only variables needed for complementarity measure
    var_pred.Resize(n, m_eq, m_ineq);

    var_pred.y = Dvar_pred.y;
    var_pred.lambda = Dvar_pred.lambda;

    var_pred.y.MatrScale(alfa_pred_prim);
    var_pred.lambda.MatrScale(alfa_pred_dual);

    var_pred.y += var.y;
    var_pred.lambda += var.lambda;

    /*** compute complementarity measure ***/
    auto mu_pred = var_pred.y.MatrDot(var_pred.y, var_pred.lambda) / m_ineq;  // from 16.56 pag.481


    if( only_predict)
    {
        // update remaining variables (v, gamma)
        var_pred.v = Dvar_pred.v;
        var_pred.v.MatrScale(alfa_pred_prim);
        var_pred.v += var.v;

        var_pred.gamma = Dvar_pred.gamma;
        var_pred.gamma.MatrScale(alfa_pred_dual);
        var_pred.gamma += var.gamma;

        // store in global variables
        var.v = var_pred.v;
        var.y = var_pred.y;
        var.gamma = var_pred.gamma;
        var.lambda = var_pred.lambda;

        // update residuals
        res.rp_lambda.MatrScale(1 - alfa_pred_prim);
        res.rp_gamma.MatrScale(1 - alfa_pred_prim);

        res.rd.MatrScale(1 - alfa_pred_dual);
        if (!equal_step_lengths)
        {
            multiplyH(Dvar_pred.v, vectn);                          // vectn = H * Dvar.v
            vectn.MatrScale(alfa_pred_prim - alfa_pred_dual);  // vectn = (alfa_pred_prim -
                ↪ alfa_pred_dual) * (H * Dvar.v)
            res.rd += vectn;
        }
```

```
        res.mu = mu_pred;

        return;
    }

    /*****************************************************************************/
    /***************************** Correction phase ******************************/
    /*****************************************************************************/

    /*** evaluate centering parameter ***/
    auto sigma = std::pow(std::max(0.0, std::min(1.0, mu_pred / res.mu)), 3.0); // from [5] pag. 12
    //auto sigma = std::pow(mu_pred / res.mu, 3.0); // from [5] pag. 12

    /*** step length correction ***/
    auto tau = adaptive_eta ? exp(-res.mu * m_ineq) * 0.1 + 0.9 : 0.95;  // exponential descent of tau

    auto rhs_corr = 0;
    //auto rhs_corr = sigma;

    for (auto cs = 0; cs < ineq_mode.size(); ++cs)
    {
        if (ineq_mode[cs] == eChConstraintModeMOD::CONSTRAINT_UNILATERAL || ineq_mode[cs] ==
            ↪ eChConstraintModeMOD::CONSTRAINT_FRIC_N)
        {
            vectm_ineq(cs) = sigma*res.mu - Dvar_pred.lambda(cs)*Dvar_pred.y(cs);
            continue;
        }

        if (ineq_mode[cs] == eChConstraintModeMOD::CONSTRAINT_FRIC_UV)
        {
            vectm_ineq(cs) = - Dvar_pred.lambda(cs)*Dvar_pred.y(cs);
            continue;
        }
    }

    inverse_Hadamard(yl_scaled, vectm_ineq, vectm_ineq);
    for (auto row_sel = 0; row_sel < m_ineq; row_sel++) mumps_rhs.SetElement(row_sel + n + m_eq, 0, -
         ↪ yl_scaled(row_sel) + vectm_ineq(row_sel));
    scaling_matrix.MatrMultiplyClipped(mumps_rhs, vectm_ineq, 0, m_ineq - 1, 0, m_ineq - 1, n + m_eq,
         ↪ 0, true, 0, 0, 0, true);


    /*** find directions ***/
    for (auto row_sel = 0; row_sel < n;      row_sel++) mumps_rhs(row_sel           ) =  -(1 -
         ↪ rhs_corr)*res.rd(row_sel);
    for (auto row_sel = 0; row_sel < m_eq;   row_sel++) mumps_rhs(row_sel + n       ) =  -(1 -
         ↪ rhs_corr)*res.rp_gamma(row_sel);
    for (auto row_sel = 0; row_sel < m_ineq; row_sel++) mumps_rhs(row_sel + n + m_eq) =  -(1 -
         ↪ rhs_corr)*res.rp_lambda(row_sel) + vectm_ineq(row_sel);

    Dvar.Resize(n, m_eq, m_ineq);
    get_Newton_direction(Dvar, mumps_rhs, res);


    /*** compute step lengths ***/
#ifdef USE_MY_MAX_STEPLENGTHS
    auto alfa_corr_prim = get_Newton_steplength_MAX_mod(var.y, Dvar.y);
    auto alfa_corr_dual = get_Newton_steplength_MAX_mod(var.lambda, Dvar.lambda);
#else
    auto alfa_corr_prim = get_Newton_steplength_MAX(var.y, Dvar.y);
    auto alfa_corr_dual = get_Newton_steplength_MAX(var.lambda, Dvar.lambda);
#endif


#ifdef DEBUG_CONSOLE
    printf("|%-6.0g", mu_pred);
    printf("||%-6.0g", tau);
    printf("||%-6.0g|%-6.0g", alfa_corr_prim, alfa_corr_dual);
    //printf("|%-6d|%-6d|", find_out_of_cone(var.y, Dvar.y, alfa_corr_prim), find_out_of_cone(var.
         ↪ lambda, Dvar.lambda, alfa_corr_dual));
#endif

    if( equal_step_lengths )
    {
        auto alfa_corr = std::min(alfa_corr_prim, alfa_corr_dual);
        alfa_corr_prim = alfa_corr;
        alfa_corr_dual = alfa_corr;
    }

    // apply correction
    alfa_corr_prim *= tau;
    alfa_corr_dual *= tau;


    /*** make the correction step ***/
    var_corr.Resize(n, m_eq, m_ineq);
```

```
    var_corr.v = Dvar.v;        var_corr.v.MatrScale(alfa_corr_prim);        var_corr.v += var.v;
        ↪          var.v = var_corr.v;
    var_corr.y = Dvar.y;        var_corr.y.MatrScale(alfa_corr_prim);        var_corr.y += var.y;
        ↪          var.y = var_corr.y;
    var_corr.gamma = Dvar.gamma;      var_corr.gamma.MatrScale(alfa_corr_dual);     var_corr.gamma +=
        ↪ var.gamma;        var.gamma = var_corr.gamma;
    var_corr.lambda = Dvar.lambda;    var_corr.lambda.MatrScale(alfa_corr_dual);    var_corr.lambda +=
        ↪  var.lambda;    var.lambda = var_corr.lambda;

    /********** Residuals update **********/
    res.mu = var.y.MatrDot(var.y, var.lambda) / m_ineq;  // from 14.6 pag.395
    res.rp_gamma.MatrScale(1 - alfa_corr_prim);
    res.rp_lambda.MatrScale(1 - alfa_corr_prim);

    res.rd.MatrScale(1 - alfa_corr_dual);
    if( !equal_step_lengths )
    {
        multiplyH(Dvar.v, vectn);                          // vectn = H*Dvar.v
        vectn.MatrScale(alfa_corr_prim - alfa_corr_dual);  // vectn = (alfa_pred_prim - alfa_pred_dual
            ↪ ) * (H * Dvar.v)
        res.rd += vectn;
    }


    if( print_history )
        output_log();

#ifdef DEBUG_CONSOLE
    printf("||%-6.0g", res.mu);
    printf("|%-12.0g|%-12.0g|%-12.0g|", res.rd.NormTwo(), res.rp_gamma.NormTwo(), res.rp_lambda.
        ↪ NormTwo());
    printf("\n");
#endif


}

void ChInteriorPoint::factorize_system_matrix() {

    BigMat.Compress();
    mumps_engine.SetMatrix(BigMat);
    for (auto loop_expand_workspace_size = 0; loop_expand_workspace_size < 5; ++
        ↪ loop_expand_workspace_size)
    {
        if (mumps_engine.MumpsCall(ChMumpsEngine::ANALYZE_FACTORIZE) == 9)
        {
            mumps_engine.SetICNTL(14, static_cast<int>(round(mumps_engine.GetICNTL(14)*1.5)));
            std::cout << "MUMPS work space will be allocated overestimating the estimate by " <<
                ↪ mumps_engine.GetICNTL(14) << "%" << std::endl;
        }
        else
            break;

    }

}

void ChInteriorPoint::get_Newton_direction(IPvariables_t& Dvar_unknown, ChMatrix<>& rhs, const
    ↪ IPresidual_t& residuals) {
    // Solve the KKT system
    mumps_engine.SetRhsVector(rhs);
    if( mumps_engine.MumpsCall(ChMumpsEngine::SOLVE) )
        mumps_engine.PrintINFOG();
    if( mumps_engine.GetRINFOG(6) > 1e-6 )
        std::cout << "Scaled residual norm of MUMPS call: " << mumps_engine.GetRINFOG(6) << std::endl;

    // MUMPS uses its 'rhs' vector to store the solution. In order to clarify that:
    const ChMatrixDynamic<double>& sol = rhs;

    // Extract 'v' and 'lambda' from 'sol'
    for (auto row_sel = 0; row_sel < n;      row_sel++ ) Dvar_unknown.v.SetElement(row_sel, 0, sol.
        ↪ GetElement(row_sel, 0));
    if (leverage_symmetry)
    {
        for (auto row_sel = 0; row_sel < m_eq; row_sel++) Dvar_unknown.gamma.SetElement(row_sel, 0, -
            ↪ sol.GetElement(row_sel + n, 0));
        for (auto row_sel = 0; row_sel < m_ineq; row_sel++) Dvar_unknown.lambda.SetElement(row_sel, 0,
            ↪  -sol.GetElement(row_sel + n + m_eq, 0));
    }
    else
    {
        for (auto row_sel = 0; row_sel < m_eq; row_sel++) Dvar_unknown.gamma.SetElement(row_sel, 0,
            ↪ sol.GetElement(row_sel + n, 0));
        for (auto row_sel = 0; row_sel < m_ineq; row_sel++) Dvar_unknown.lambda.SetElement(row_sel, 0,
            ↪  sol.GetElement(row_sel + n + m_eq, 0));
    }
```

```
    // Calc 'y' (it is also possible to evaluate y as (-lambda o y+sigma*res.mu*e-y o lambda)./lambda
        ↪ )
    multiplyIneqCon(Dvar_unknown.v, Dvar_unknown.y);  // Dy = IneqCon*Dv
    Dvar_unknown.y += residuals.rp_lambda; // Dy = (IneqCon*Dv) + rp_lambda
    if( ADD_COMPLIANCE )
    {
        E.MatrMultiply(Dvar_unknown.lambda, vectm_ineq);
        Dvar_unknown.y += vectm_ineq;
    }
}

double ChInteriorPoint::get_Newton_steplength_MAX(const ChMatrix<double>& pos, const ChMatrix<double>&
     ↪  dir) const
{
    // we are going to solve max{alfa | pos + alfa*dir >= 0 }
    double alfa_inv = 0.0;
    ChMatrixNM<double, 3, 1> a_pr;
    ChMatrixNM<double, 3, 1> pos_n; // normalized scaled variable
    for (auto cs = 0; cs < ineq_mode.size(); ++cs)
    {
        if (ineq_mode[cs] == eChConstraintModeMOD::CONSTRAINT_UNILATERAL || ineq_mode[cs] ==
            ↪ eChConstraintModeMOD::CONSTRAINT_FRIC_N && skip_contacts_uv)
        {
            if (-dir(cs) / pos(cs) > alfa_inv)
            {
                alfa_inv = -dir(cs) / pos(cs);
            }

            continue;
        }


        if (ineq_mode[cs] == eChConstraintModeMOD::CONSTRAINT_FRIC_N)
        {
            auto pos_p = projection_on_polar_cone(pos, cs);

            pos_n(0) = pos(cs + 0) / pos_p;
            pos_n(1) = pos(cs + 1) / pos_p;
            pos_n(2) = pos(cs + 2) / pos_p;

            auto pd_p = pos_n(0)*dir(cs + 0) - pos_n(1)*dir(cs + 1) - pos_n(2)*dir(cs + 2);
            a_pr(0) = pd_p / pos_p;
            a_pr(1) = dir(cs + 1) - (pd_p + dir(cs + 0)) / (pos_n(0) + 1)*pos_n(1);
            a_pr(2) = dir(cs + 2) - (pd_p + dir(cs + 0)) / (pos_n(0) + 1)*pos_n(2);

            auto alfa_inv_temp = -a_pr(0) + sqrt(a_pr(1)*a_pr(1) + a_pr(2)*a_pr(2));

            if (alfa_inv_temp > alfa_inv)
                alfa_inv = alfa_inv_temp;

            continue;
        }
    } // end FOR loop

    return std::min(1.0 / alfa_inv, 1.0);
}


double ChInteriorPoint::get_Newton_steplength_MIN_along_cone_axis(const ChMatrix<double>& pos) const
{
    // we are going to solve min{alfa | pos + alfa*[1;0;0] >= 0 i.e. pos + alfa*[1;0;0] is inside the
        ↪ cone with equation u(0)>=norm(u(1:end))}
    double alfa_lb = -1000;
    for (auto cs = 0; cs < ineq_mode.size(); ++cs)
    {
        if (ineq_mode[cs] == eChConstraintModeMOD::CONSTRAINT_UNILATERAL || ineq_mode[cs] ==
            ↪ eChConstraintModeMOD::CONSTRAINT_FRIC_N && skip_contacts_uv)
        {
            alfa_lb = std::max(alfa_lb, -pos(cs));
            continue;
        }

        if (ineq_mode[cs] == eChConstraintModeMOD::CONSTRAINT_FRIC_N)
        {
            alfa_lb = std::max(alfa_lb, -pos(cs) + sqrt(pos(cs + 1)*pos(cs + 1) + pos(cs + 2)*pos(cs +
                ↪  2)));
            continue;
        }
    } // end FOR loop

    return alfa_lb;
}
```

```
void ChInteriorPoint::set_feasible_starting_point()
{
    // fill SE corner with identity matrix
    for (auto diag_sel = 0; diag_sel < m_ineq; ++diag_sel)
        BigMat.SetElement(n + m_eq + diag_sel, n + m_eq + diag_sel, leverage_symmetry ? -1 : +1, true)
            ↪ ;

    mumps_rhs.Resize(n + m_eq + m_ineq, 1);
    for (auto row_sel = 0; row_sel < n; row_sel++) mumps_rhs(row_sel) = -rhs.c(row_sel, 0);
    for (auto row_sel = 0; row_sel < m_eq; row_sel++) mumps_rhs(row_sel + n) = +rhs.b_eq(row_sel, 0);
    for (auto row_sel = 0; row_sel < m_ineq; row_sel++) mumps_rhs(row_sel + n + m_eq) = +rhs.b_ineq(
        ↪ row_sel, 0);

    BigMat.Compress();
    mumps_engine.SetProblem(BigMat, mumps_rhs);
    if (mumps_engine.MumpsCall(ChMumpsEngine::COMPLETE))
        mumps_engine.PrintINFOG();

    var.v.Resize(n, 1);
    var.gamma.Resize(m_eq, 1);
    var.lambda.Resize(m_ineq, 1);
    for (auto row_sel = 0; row_sel < n; row_sel++) var.v(row_sel) = mumps_rhs(row_sel);
    for (auto row_sel = 0; row_sel < m_eq; row_sel++) var.gamma(row_sel) = leverage_symmetry ? -
        ↪ mumps_rhs(n + row_sel) : mumps_rhs(n + row_sel);
    for (auto row_sel = 0; row_sel < m_ineq; row_sel++) vectm_ineq(row_sel) = leverage_symmetry ? -
        ↪ mumps_rhs(n + m_eq + row_sel) : mumps_rhs(n + m_eq + row_sel);


    auto alfa_p = get_Newton_steplength_MIN_along_cone_axis(-vectm_ineq);
    auto alfa_d = get_Newton_steplength_MIN_along_cone_axis(vectm_ineq);

    if (alfa_p >= -1e-8)
        for (auto cs = 0; cs < ineq_mode.size(); ++cs)
            var.y(cs) = ineq_mode[cs] == eChConstraintModeMOD::CONSTRAINT_FRIC_UV ? -vectm_ineq(cs) :
                ↪ -vectm_ineq(cs) + (1 + alfa_p);
    else
        var.y = -vectm_ineq;

    if (alfa_d >= -1e-8)
        for (auto cs = 0; cs < ineq_mode.size(); ++cs)
            var.lambda(cs) = ineq_mode[cs] == eChConstraintModeMOD::CONSTRAINT_FRIC_UV ? vectm_ineq(cs
                ↪ ) : vectm_ineq(cs) + (1 + alfa_d);
    else
        var.lambda = vectm_ineq;


    residual_fullupdate(res, var);


#ifdef DEBUG_TRAPS
    if (find_out_of_cone(var.y) != -1)
        std::cout << "y out of cone: " << find_out_of_cone(var.y) << std::endl;
    if (find_out_of_cone(var.lambda) != -1)
        std::cout << "lambda out of cone: " << find_out_of_cone(var.lambda) << std::endl;
    assert(is_valid(var.y, m_ineq));
    assert(is_valid(var.lambda, m_ineq));
#endif

}

inline double ChInteriorPoint::uTJv(const ChMatrix<double>& u, const ChMatrix<double>& v, int offset)
{
    return u(offset)*v(offset) - u(offset + 1)*v(offset + 1) - u(offset + 2)*v(offset + 2);
}


void ChInteriorPoint::inverse_Hadamard(const ChMatrix<>& v1, const ChMatrix<>& v2, ChMatrix<>& v_out)
    ↪ const
{

    double sf = 0;
    for (auto cs = 0; cs < ineq_mode.size(); ++cs)
    {
        if (ineq_mode[cs] == eChConstraintModeMOD::CONSTRAINT_UNILATERAL || ineq_mode[cs] ==
            ↪ eChConstraintModeMOD::CONSTRAINT_FRIC_N && skip_contacts_uv)
        {
            v_out(cs) = v2(cs) / v1(cs);
            continue;
        }


        if (ineq_mode[cs] == eChConstraintModeMOD::CONSTRAINT_FRIC_N)
        {
            sf = 1 / std::pow(projection_on_polar_cone(v1, cs), 2.0);

            // backup v2 (needed if v_out == v2)
            auto v2_0 = v2(cs + 0);
```

```
            auto v2_1 = v2(cs + 1);
            auto v2_2 = v2(cs + 2);

            v_out(cs + 0) = ( v1(cs + 0) * v2_0 - v1(cs + 1) * v2_1 - v1(cs + 2) * v2_2) / sf;
            v_out(cs + 1) = (-v1(cs + 1) * v2_0 + (sf + std::pow(v1(cs + 1), 2.0)) / v1(cs + 0) * v2_1
                ↪  + v1(cs + 1)*v1(cs + 2) / v1(cs + 0) * v2_2) / sf;
            v_out(cs + 2) = (-v1(cs + 2) * v2_0 + v1(cs + 1)*v1(cs + 2) / v1(cs + 0) * v2_1 + (sf +
                ↪ std::pow(v1(cs + 2), 2.0)) / v1(cs + 0) * v2_2) / sf;
            cs += 2;
            continue;
        }
    }

}

void ChInteriorPoint::computeNesterovToddScalingMatrix(const IPvariables_t& var)
{
    // resize the scaling matrix
    auto nnz = 0;
    for (auto cs = 0; cs < ineq_mode.size(); ++cs)
    {
        if (ineq_mode[cs] == eChConstraintModeMOD::CONSTRAINT_UNILATERAL)
            nnz++;

        if (ineq_mode[cs] == eChConstraintModeMOD::CONSTRAINT_FRIC_N)
            skip_contacts_uv ? ++nnz : nnz += 9;

        // skip UV, they have already been considered in the previous 'if'
    }
    scaling_matrix.Reset(m_ineq, m_ineq, nnz);

    // evaluate the scaling matrix
    for (auto cs = 0; cs<ineq_mode.size(); ++cs)
    {
        if (ineq_mode[cs] == eChConstraintModeMOD::CONSTRAINT_UNILATERAL || ineq_mode[cs] ==
            ↪ eChConstraintModeMOD::CONSTRAINT_FRIC_N && skip_contacts_uv)
        {
            scaling_matrix.SetElement(cs, cs, sqrt(var.y(cs))/ sqrt(var.lambda(cs)));
            continue;
        }


        if (ineq_mode[cs] == eChConstraintModeMOD::CONSTRAINT_FRIC_N)
        {
            // grant that we are handling UV contact components
            assert(!skip_contacts_uv);
            // grant that the friction force N is followed by two UV
            assert(ineq_mode[cs + 1] == eChConstraintModeMOD::CONSTRAINT_FRIC_UV &&
                ineq_mode[cs + 2] == eChConstraintModeMOD::CONSTRAINT_FRIC_UV
                && "A friction reaction force along N is not followed by a couple of constraints on UV
                    ↪ ");

            // WARNING: assembling the scaling matrix is possible only if the vectors 'y' and 'lambda'
            // are STRICTLY inside the cone otherwise you will get NaN (if on the border) or complex
                ↪ numbers (if outside)

#ifdef CORRECT_STARTING_POINTS // there are multiple calls to uTJv that can be avoided
            auto alfa_y = -var.y(cs) + sqrt(var.y(cs + 1)*var.y(cs + 1) + var.y(cs + 2)*var.y(cs + 2))
                ↪ ;
            auto alfa_lam = -var.lambda(cs) + sqrt(var.lambda(cs + 1)*var.lambda(cs + 1) + var.lambda(
                ↪ cs + 2)*var.lambda(cs + 2));
            // if alfa is positive it means that we have to
            if (alfa_y >= -1e-8 || var.y(cs)<=0)
                const_cast<ChMatrixDynamic<>&>(var.y)(cs) += std::max(alfa_y*1.000001, 1e-6);
            if (alfa_lam >= -1e-8 || var.lambda(cs) <= 0)
                const_cast<ChMatrixDynamic<>&>(var.lambda)(cs) += std::max(alfa_lam*1.000001, 1e-6);

#endif
            auto y_proj = std::sqrt(uTJv(var.y, var.y, cs));
            auto lam_proj = std::sqrt(uTJv(var.lambda, var.lambda, cs));
            auto gamma = std::sqrt((1 + (var.lambda(cs)*var.y(cs) + var.lambda(cs + 1)*var.y(cs + 1) +
                ↪  var.lambda(cs + 2)*var.y(cs + 2)) / y_proj / lam_proj) / 2);


            ChVector<double> nsp; // Normalized Scaling Point  i.e. \bar{w}_k
            nsp[0] = (var.y(cs    ) / y_proj + var.lambda(cs    ) / lam_proj) / 2 / gamma;
            nsp[1] = (var.y(cs + 1) / y_proj - var.lambda(cs + 1) / lam_proj) / 2 / gamma;
            nsp[2] = (var.y(cs + 2) / y_proj - var.lambda(cs + 2) / lam_proj) / 2 / gamma;

            ChMatrix33<double> nsm; // Normalized Scaling SubMatrix i.e. \bar{W}_k
            nsm[0][0] = nsp[0];
            nsm[0][1] = nsp[1];
            nsm[0][2] = nsp[2];

            nsm[1][0] = nsp[1];
            nsm[2][0] = nsp[2];
```

```
            nsm[1][1] = 1 + nsp[1] * nsp[1] / (nsp[0] + 1);
            nsm[1][2] =     nsp[1] * nsp[2] / (nsp[0] + 1);
            nsm[2][1] =     nsp[2] * nsp[1] / (nsp[0] + 1);
            nsm[2][2] = 1 + nsp[2] * nsp[2] / (nsp[0] + 1);


#ifdef DEBUG_TRAPS
            // // from [5] pag.9
            // deb = 2*J*(nsp*nsp')*J - J
            ChMatrix33<double> deb;
            deb(0,0) = + 2 * nsp[0]*nsp[0] - 1;
            deb(0,1) = - 2 * nsp[0]*nsp[1];
            deb(0,2) = - 2 * nsp[0]*nsp[2];
            deb(1,0) = - 2 * nsp[1]*nsp[0];
            deb(1,1) = + 2 * nsp[1]*nsp[1] + 1;
            deb(1,2) = + 2 * nsp[1]*nsp[2];
            deb(2,0) = - 2 * nsp[2]*nsp[0];
            deb(2,1) = + 2 * nsp[2]*nsp[1];
            deb(2,2) = + 2 * nsp[2]*nsp[2] + 1;

            // deb = W * deb * W
            ChMatrix33<double> temp;
            temp.MatrMultiply(nsm, deb);
            deb.MatrMultiply(temp, nsm);

            // deb = identity
            for (auto i = 0; i < 3; ++i)
                for (auto j = 0; j < 3; ++j)
                    i == j ? assert(std::abs(deb(i,j) - 1) < 1e-4) : assert(std::abs(deb(i, j)) < 1e
                        ↪ -4);
#endif

            // De-normalized Scaling SubMatrix i.e. W_k
            nsm *= std::sqrt(y_proj / lam_proj);

            scaling_matrix.PasteClippedMatrix(nsm, 0, 0, 3, 3, cs, cs, true);

            cs += 2;
            continue;
        }

        // skip UV, they should have already been considered in the previous 'if'

#ifdef DEBUG_TRAPS
        if (ineq_mode[cs] == eChConstraintModeMOD::CONSTRAINT_FRIC_UV)
            continue;

        assert(false && "An unhandled constraint has been found.");
#endif


    } // end FOR loop

    scaling_matrix.Compress(); //TODO: it might be not needed

}


void ChInteriorPoint::residual_fullupdate(IPresidual_t& residuals, const IPvariables_t& variables)
     ↪ const {
    // Dual Residual
    // res.rd = H*v + c - EqCon^T*gamma - IneqCon^T*lambda
    multiplyH(variables.v, residuals.rd);  // res.rd = H*v
    residuals.rd += rhs.c; // res.rd = (H*v) + c
    if (m_eq > 0)
    {
        multiplyNegEqConT(variables.gamma, vectn);
        residuals.rd += vectn; // res.rd = (H*v + c) + (-EqCon^T*gamma)

        // res.rp_gamma = EqCon*v - b_eq
        multiplyEqCon(variables.v, residuals.rp_gamma);  // res.rp_gamma = EqCon*v
        residuals.rp_gamma -= rhs.b_eq; // res.rp_gamma = (EqCon*v) - b_eq

    }
    else
        residuals.rp_gamma.FillElem(0);


    if( m_ineq > 0 )
    {
        multiplyNegIneqConT(variables.lambda, vectn);  // vectn = (-IneqCon^T)*lambda
        residuals.rd += vectn;                // res.rd = (H*v + c - EqCon^T*gamma) + (-IneqCon^T*
            ↪ lambda)

        // Primal residual
        // res.rp_lambda = IneqCon*v - y - b
```

```
        multiplyIneqCon(variables.v, residuals.rp_lambda);  // res.rp_lambda = IneqCon*v
        residuals.rp_lambda -= variables.y; // res.rp_lambda = (IneqCon*v) - y
        residuals.rp_lambda -= rhs.b_ineq; // res.rp_lambda = (IneqCon*v - y) - b_ineq;
        if( ADD_COMPLIANCE )
        {
            E.MatrMultiply(variables.lambda, vectm_ineq);
            residuals.rp_lambda += vectm_ineq;
        }

        residuals.mu = variables.y.MatrDot(variables.y, variables.lambda) / m_ineq;
    }
    else
    {
        residuals.rp_lambda.FillElem(0);
        residuals.mu = 0;
    }
}

int ChInteriorPoint::find_out_of_cone(const ChMatrix<double>& pos)
{
    for (auto cs = 0; cs < ineq_mode.size(); ++cs)
    {
        if (ineq_mode[cs] == eChConstraintModeMOD::CONSTRAINT_UNILATERAL || ineq_mode[cs] ==
            ↪ eChConstraintModeMOD::CONSTRAINT_FRIC_N && skip_contacts_uv)
        {
            if (pos(cs) < 0)
                return cs;

            continue;
        }

        if (ineq_mode[cs] == eChConstraintModeMOD::CONSTRAINT_FRIC_N)
            if (pos(cs) < 0 || uTJv(pos, pos, cs) < 0)
                return cs;

    }

    return -1;
}

ChInteriorPoint::ChInteriorPoint() {
    mumps_engine.SetICNTL(11, 2);
    RecordHistory(true);
    BigMat.SetSparsityPatternLock(m_lock);
}

ChInteriorPoint::~ChInteriorPoint() {
    if( logfile_stream.is_open() )
        logfile_stream.close();
}


} // end namespace chrono
```

Bibliography

[1] Bruce R. Donald and Dinesh K. Pai. On the motion of compliantly connected rigid bodies in contact: A system for analyzing designs for assembly. In *Proceedings of the Conf. on Robotics and Automation*, pages 1756–1762. IEEE, 1990.

[2] Peng Song, P. Kraus, Vijay Kumar, and P. Dupont. Analysis of rigid-body dynamic models for simulation of systems with frictional contacts. *Journal of Applied Mechanics*, 68(1):118–128, 2001.

[3] Peng Song, Jong-Shi Pang, and Vijay Kumar. A semi-implicit time-stepping model for frictional compliant contact problems. *International Journal of Numerical Methods in Engineering*, 60(13):267–279, 2004.

[4] D. E. Stewart and J.C. Trinkle. An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction. *International J. Numer. Methods Engineering*, 39(15):281–287, 1996.

[5] P. Lotstedt. Mechanical systems of rigid bodies subject to unilateral constraints. *SIAM Journal of Applied Mathematics*, 42(2):281–296, 1982.

[6] M. D. P.Monteiro Marques. *Differential Inclusions in Nonsmooth Mechanical Problems: Shocks and Dry Friction*, volume 9 of *Progress in Nonlinear Differential Equations and Their Applications*. Birkhäuser Verlag, Basel, 1993.

[7] J. J. Moreau. Unilateral contact and dry friction in finite freedom dynamics. In J. J. Moreau and P. D. Panagiotopoulos, editors, *Nonsmooth Mechanics and Applications*, pages 1–82, Berlin, 1988. Springer-Verlag.

[8] David Baraff. Issues in computing contact forces for non-penetrating rigid bodies. *Algorithmica*, 10:292–352, 1993.

[9] Jong-Shi Pang and Jeffrey C. Trinkle. Complementarity formulations and existence of solutions of dynamic multi-rigid-body contact problems with coulomb friction. *Math. Program.*, 73(2):199–226, 1996.

[10] Jeffrey Trinkle, Jong-Shi Pang, Sandra Sudarsky, and Grace Lo. On dynamic multi-rigid-body contact problems with Coulomb friction. *Zeitschrift fur angewandte Mathematik und Mechanik*, 77:267–279, 1997.

[11] Mihai Anitescu, James F. Cremer, and Florian A. Potra. Formulating 3D contact dynamics problems. *Mechanics of Structures and Machines*, 24(4):405–437, 1996.

[12] Mihai Anitescu, Florian A. Potra, and David Stewart. Time-stepping for three-dimensional rigid-body dynamics. *Computer Methods in Applied Mechanics and Engineering*, 177:183–197, 1999.

[13] David E. Stewart. Rigid-body dynamics with friction and impact. *SIAM Review*, 42(1):3–39, 2000.

[14] D.E. Stewart. Convergence of a time-stepping scheme for rigid body dynamics and resolution of Painlevé's problems. *Archive Rational Mechanics and Analysis*, 145(3):215–260, 1998.

[15] F. Génot and B. Brogliato. New results on painlevé paradoxes. *European Journal of Mechanics - A/Solids*, 18(4):653 – 677, 1999. URL: http://www.sciencedirect.com/science/article/pii/S0997753899001448, doi:https://doi.org/10.1016/S0997-7538(99)00144-8.

[16] Vincent Acary and Bernard Brogliato. *Numerical methods for nonsmooth dynamical systems: applications in mechanics and electronics*, volume 35. Lect. N. App. Comput. Mech. 35 Springer Verlag, 2008.

[17] Bernard Brogliato. Nonsmooth Mechanics. Models, Dynamics and Control : Erratum/Addendum. Research report, INRIA Grenoble - Rhone-Alpes, February 2017. URL: `https://hal.inria.fr/hal-01331565`.

[18] Mihai Anitescu and Florian A. Potra. Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems. *Nonlinear Dynamics*, 14:231–247, 1997.

[19] David E. Stewart and Jeffrey C. Trinkle. An implicit time-stepping scheme for rigid-body dynamics with inelastic collisions and Coulomb friction. *International Journal for Numerical Methods in Engineering*, 39:2673–2691, 1996.

[20] Alessandro Tasora and Mihai Anitescu. A matrix-free cone complementarity approach for solving large-scale, nonsmooth, rigid body dynamics. *Computer Methods in Applied Mechanics and Engineering*, 200(5-8):439 – 453, 2011. URL: `http://www.sciencedirect.com/science/article/B6V29-50GWN46-2/2/47d6da2ab8348d986a3f57335006bfae`, `doi:DOI:10.1016/j.cma.2010.06.030`.

[21] Friedrich Pfeiffer and Christian Glocker. *Multibody Dynamics with Unilateral Contacts*. John Wiley, New York City, 1996.

[22] J.-B. Hiriart-Urruty and C. Lemarechal. *Convex Analysis and Minimization Algorithms*. Springer Verlag, Berlin, 1993.

[23] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, jan 1965. `doi:10.1093/comjnl/7.4.308`.

[24] Sanjay Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on Optimization*, 2(4):575–601, 1992.

URL: http://dx.doi.org/10.1137/0802028, arXiv:http://dx.doi.org/10.1137/0802028, doi:10.1137/0802028.

[25] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, December 1984. URL: http://dx.doi.org/10.1007/BF02579150, doi:10.1007/BF02579150.

[26] M. Renouf and P. Alart. Conjugate gradient type algorithms for frictional multi-contact problems: applications to granular materials. *Computer Methods in Applied Mechanics and Engineering*, 194(18-20):2019–2041, May 2005. URL: http://dx.doi.org/10.1016/j.cma.2004.07.009, doi:10.1016/j.cma.2004.07.009.

[27] J. S. Pang and D. E. Stewart. Differential variational inequalities. *Mathematical Programming*, 113:1–80, 2008.

[28] Vincent Acary, Maurice Brémond, Tomasz Koziara, and Franck Pérignon. Fclib: a collection of discrete 3d frictional contact problems. Technical report, INRIA, 2014. URL: https://hal.inria.fr/hal-00945820v1.

[29] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer New York, 2006. URL: https://books.google.it/books?id=eNlPAAAAMAAJ.

[30] James Diebel. Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Matrix*, 58(15-16):1–35, 2006.

[31] Young J. Kim, Ming C. Lin, and Dinesh Manocha. Deep: Dual-space expansion for estimating penetration depth between convex polytopes. In *Proceedings of the 2002 International Conference on Robotics and Automation*, volume 1, pages 921–926. Institute for Electrical and Electronics Engineering, 2002.

[32] Mihai Anitescu and Gary D. Hart. A constraint-stabilized time-stepping approach for rigid multibody dynamics with joints, contact and friction. *International Journal for Numerical Methods in Engineering*, 60(14):2335–2371, 2004.

[33] Christer Ericson. *Real-Time Collision Detection*. Elsevier, 2005.

[34] S.S. Keerthi E.G. Gilbert, D.W. Johnson. A fast procedure for computing the distance between complex objects in three-dimensional space. *Robotics and Automation*, 4(2):193–203, 1988.

[35] David E. Stewart. Convergence of a time-stepping scheme for rigid body dynamics and resolution of Painleve's problems. *Archive Rational Mechanics and Analysis*, 145(3):215–260, 1998.

[36] A. Tasora. Time integration in chrono::engine. Technical report, Università degli Studi di Parma, 2017. URL: `http://www.projectchrono.org/assets/white_papers/ChronoCore/integrator.pdf`.

[37] Jacek Gondzio. Interior point methods 25 years later. *European Journal of Operational Research*, 218(3):587–601, 2012.

[38] David Baraff. Fast contact force computation for nonpenetrating rigid bodies. In *Computer Graphics (Proceedings of SIGGRAPH)*, pages 23–34, 1994.

[39] Thomas Reslow Kruth. Interior-point algorithms for quadratic programming. Master's thesis, Technical University of Denmark, DTU, DK-2800 Kgs. Lyngby, Denmark, 2008.

[40] Lieven Vandenberghe. *The CVXOPT linear and quadratic cone program solvers*, 2010.

[41] Jan Kleinert, Bernd Simeon, and Martin Obermayr. An inexact interior point method for the large-scale simulation of granular material. *Computer Methods in Applied Mechanics and Engineering*, 278(0):567 – 598, 2014. URL: `http://www.sciencedirect.com/science/article/pii/S0045782514001959`, `doi:http://dx.doi.org/10.1016/j.cma.2014.06.009`.

[42] Erling D Andersen, Jacek Gondzio, Csaba Mészáros, Xiaojie Xu, et al. *Implementation of interior point methods for large scale linear programming*. HEC/Université de Genève, 1996.

[43] Marco D'Apuzzo, Valentina De Simone, and Daniela di Serafino. Starting-point strategies for an infeasible potential reduction method. *Optimization Letters*, 4(1):131–146, 2010. URL: http://dx.doi.org/10.1007/s11590-009-0150-9, doi:10.1007/s11590-009-0150-9.

[44] Dario Mangoni, Alessandro Tasora, and Iryna Yasinskaya. High performance linear solvers for large-scale problems in multibody dynamics. *International CAE Conference 2015 Proceedings*, 2015.

[45] Jacek Gondzio. Matrix-free interior point method. *Computational Optimization and Applications*, 51(2):457–480, Mar 2012. URL: https://doi.org/10.1007/s10589-010-9361-3, doi:10.1007/s10589-010-9361-3.

[46] Michele Benzi, Gene H Golub, and Jorg Liesen. Numerical solution of saddle point problems. *Acta numerica*, 14(1):1–137, 2005.

[47] Yu E Nesterov and Michael J Todd. Self-scaled barriers and interior-point methods for convex programming. *Mathematics of Operations research*, 22(1):1–42, 1997.

[48] Yu E Nesterov and Michael J Todd. Primal-dual interior-point methods for self-scaled cones. *SIAM Journal on optimization*, 8(2):324–364, 1998.

[49] Leonid Faybusovich. Linear systems in jordan algebras and primal-dual interior-point algorithms. *Journal of Computational and Applied Mathematics*, 86(1):149 – 175, 1997. Dedicated to William B. Gragg on the ocassion of his 60th Birthday. URL: http://www.sciencedirect.com/science/article/pii/S0377042797001532, doi:https://doi.org/10.1016/S0377-0427(97)00153-2.