I N A U G U R A L – D I S S E R T A T I O N

zur
Erlangung der Doktorwürde
der
Naturwissenschaftlich–Mathematischen Gesamtfakultät
der
Ruprecht – Karls – Universität
Heidelberg

vorgelegt von
Diplom–Physiker Christian Kraus
aus Eberbach

Tag der mündlichen Prüfung: 3. Juli 2006

# Efficient Object–Oriented Modelling, Simulation and Parameter Estimation for Biomechanical Problems

Gutachter: Prof. Dr. Dr. h.c. Hans–Georg Bock
Prof. Dr. Hanns Ruder

# Contents

# Acknowledgements

It was very interesting and a pleasure to work at the Interdisciplinary Center of Scientific Computing (IWR) of the University of Heidelberg, which is somehow summarized in this thesis. This is mainly due to the people I worked and discussed with.

I thank Prof. Dr. Dr. h.c. Hans Georg Bock and Dr. Johannes Schlöder for the interesting topics, the possibility to study them with an open mind and in a real interdisciplinary environment. Their support not only included their profound knowledge of optimization and how to present it interestingly but also an excellent working atmosphere.

Moreover, I want to thank Prof. Hanns Ruder of the University of Tübingen and the biomechanical part of his work group for the very good cooperation during and beyond the project we did together. Especially, I want to mention my co–operation partner Helmut Mutschler.

Of my own group I want to thank the ”mechanics” Katja Mombaur, Jan Simon, Thorsten Stossmeister and Michael Winckler for the interesting discussions about optimization applied to mechanical systems and their friendship. In this context there is not to forget the over the time changing group of people I went to the mensa with; a very important social event and possibility to keep in touch with what is happening in the group.

In the final part of the thesis, several people helped with proof reading and critical remarks. These are especially Ullrich Brandt–Pollmann, Johannes Schlöder, Katja Mombaur, Jan Simon, Moritz Diehl and not to forget Dorothea Gmeiner, who not only corrected my less than perfect English, but also gave encouraging support.

Financial support by the *Landesstiftung WiR–BaWü: Parallele Simulationswerkzeuge* and the University of Heidelberg is gratefully acknowledged.

# Abstract

We identify anthropometric parameter for models of human beings and the corresponding macroscopic movement. The models are based on rigid–body formalisms and formulated as mechanical DAEs. We use the Generalized Gauß–Newton method based on multiple shooting discretization to estimate the parameters of this dynamic nonlinear parameter estimation problem using measurement data taken from motion capturing.

We adapt modelling, integration and optimization independently and in combination. The modelling based on Natural Coordinates is modified to efficiently evaluate the right–hand side of the DAE in linear time w.r.t. the number of bodies. The formulation does not introduce additional redundant constraints and we developed constraint partitioning to treat inherent singularities of the physical model. Furthermore, we include additional biomechanical elements like passive muscles and wobbling masses.

As Natural Coordinates lead to a set of redundant coordinates, we have to relax to treat inconsistent variables during the optimization process. To efficiently generate the sensitivity information needed for the optimization algorithm, we apply reduced methods to evaluate a minimal number of directional derivatives and exploit the structure of the model equations to calculate each of them. Finally, we present parameter estimation results for a complex, full three-dimensional biomechanical model of the human body with 82 kinematic degrees of freedom.

This is implemented in the object–oriented modelling tool MBSNAT and the parameter estimation package Parfit++.

# Zusammenfassung

Wir identifizieren die anthropometrischen Parameter von Menschmodellen für die Untersuchung makroskopischer Bewegung. Diese Modelle basieren auf Starrkörperformalismen and werden als mechanische DAE formuliert. Wir verwenden das Verallgemeinerte Gauß–Newton Verfahren basierend auf Multiple–Shooting Diskretisierung, um die Parameter dieses nichtlinearen dynamischen Parameterschätzproblems an Messdaten anzupassen, die mit Hilfe von Motion Capturing gewonnen wurden.

Hierfür adaptieren wir die Modellierung, die Integration und die Optimierung sowohl einzeln als auch in Kombination. Die Modellierung basierend auf Natürlichen Koordinaten wird modifiziert, so daß die rechten Seite der DAE in linearer Komplexität bzgl. der Anzahl der Körper effizient ausgewertet werden kann. Unsere Formulierung führt zu keinen zusätzlichen redundanten Nebenbedingungen und inherente Singularitäten des physikalischen Modells werden mit Hilfe von constraint partitioning behandelt. Desweiteren verwenden wir zusätzliche biomechanische Elemente wie passive Muskeln und Schwabbelmassen.

Da die Verwendung von Natürliche Koordinaten auf einen Satz redundanter Variablen führt, muß zur Behandlung von inkonsistenten Variablen während der Optimierung relaxiert werden. Für die Generierung der für das Optimierungsverfahren benötigten Ableitungsinformation verwenden wir reduzierte Verfahren, um die Anzahl der benötigten Richtungsableitungen zu minimieren, und nutzen die Struktur des Modells für die Berechnung jeder einzelnen aus. Zuletzt präsentieren wir Parameterschätzungsergebnisse für ein komplexes, voll dreidimensionales biomechanisches Menschmodell mit 82 kinematischen Freiheitsgraden.

Dies ist im objekt–orientierten Werkzeug MBSNAT und dem Parameterschätzpaket Parfit++ implementiert.

# Introduction

The study of the human itself is perhaps the most important task of natural science. It comprises the examination of the human on different scales: from biochemical reactions in the cell to – as the most palpable field – the movement of the human body as a whole. This macroscopic motion is typically studied based on rigid–body models developed for technical mechanics which are formulated based on ordinary differential equations (ODE) or mechanical differential algebraic equations (DAE). These formalisms have to be extended for biomechanical models, which include a larger variety of joint types and body forms, complex actuators in the form of muscles, connected groups of bones and neighboring soft tissue, and specific forms of passive plasto–elastic elements.

A particularly difficult issue in modelling human motion and the previously described elements is the determination of correct model parameters. Anthropometric data are typically assembled by a combination of in vivo measurements and extrapolations of cadaver studies. When we consider idealized models, some parameters may be different to real values and we have to rely on heuristic rules. Furthermore, most of the model parameters are highly dependent on the individual person's age, size, height, sex, physical condition etc. Thus, there is a tremendous need for a reliable method to determine correct model parameters for a given individual. In this work, we contribute to this aim by providing a different approach to estimate the anthropometric data of a human. It is not based on invasive surgery but on data from motion capturing, i.e. measurement data that are easy to get, like the position of some markers in a given time–interval which are filmed by several special synchronized cameras – a typical setting for modern gait laboratories. Estimating the parameters of a model based on measurement data, leads to a nonlinear high–dimensional parameter estimation problem.

Due to the complexity and size of the resulting dynamic parameter estimation problem, it is not sufficient to just combine an arbitrary *modelling tool*, an *integration scheme* and an *optimization algorithm*. To achieve the efficiency that is needed, one has to reexamine each component independently and in combination in this context.

As a starting point, we apply a Generalized Gauß–Newton (GGN) method with multiple shooting discretization as introduced by Bock [Boc85] to solve the nonlinear parameter estimation problem. We provide the biomechanical model as a mechanical DAE of index 3, and we use our version of Natural Coordinates originally introduced by García de Jalón and Bayo [GB94]. The solution of this differential equation and the corresponding sensitivity differential equation – to provide the necessary information for the derivative–based GGN – is the most time–consuming part of the overall optimization process. Therefore, we apply methods that reduce the number of directional derivatives one has to calculate explicitly.

## Contributions

To gain the high efficiency we need, we adapt the parameter estimation methods to large (bio–) mechanical models. We want to focus on the following topics, which summarize the various modifications needed:

- We adapt Natural Coordinates as introduced by García de Jalón and Bayo to high–dimensional models as encountered in biomechanics. We use a different linear algebra method, which is of **linear complexity** in the number of bodies. In contrast to García de Jalón and Bayo, who use point–sharing to reduce the dimension, but thereby destroy the structure, we **exploit the block–structure** of the Augmented system. Additionally, by using sub–block structures and the simplicity of the equations based on our version of Natural Coordinates, we are able to efficiently evaluate the right–hand side of the differential equation.

- In order to fulfil the prerequisites of our linear algebra method, we adapt the formulation of certain constraints to **avoid redundancy**. Furthermore, we develop **constraint partitioning** to handle closed loop systems, which may exhibit redundant constraints, or even systems where no globally unique set of non–redundant constraints exists. Therefore, we are able to supply an efficient and robust algorithm for **arbitrary topologies**.

- Using the simplicity and structure of the equations formulated based on Natural Coordinates, we are able to efficiently calculate the derivative of the right–hand side of the differential equation. We calculate one **directional derivative in linear complexity** w.r.t. the number of bodies. Due to the use of IND, we are able to reuse the decomposition of the Augmented system. Therefore, the evaluation of a directional derivative is approximately **two to three times faster** than the evaluation of the right–hand side of the differential equation.

- Due to the huge amount of redundant variables of models based on Natural Coordinates, it is of upmost importance to provide methods which **reduce the number of directional derivatives** one has to calculate explicitly. We study the Fixfit–approach of Schlöder [Sch88], which uses constraints of the parameter estimation problem – especially fixed initial values – and two methods which exploit constraints corresponding to the invariants of the IVP, based on [SBS98]. Fixfit–approach is especially useful, if the initial state variables are fixed, whereas the invariant–based methods allow for a **parallelization** on the level of the multiple shooting discretization.

- We introduce the **limit form of non–stiff Baumgarte relaxation** to treat in–consistent variables of the mechanical DAE during the optimization process. This method is equivalent to an ideal projection at the start of the integration interval and classical drift–avoidance during the simulation. Combined with a version of the projection algorithm for the initial states, which is **adapted to mechanical systems**, we have been able to show that our approach performs better and is more robust than methods based on consistent optimization variables as well as methods using other relaxation approaches.

- Furthermore, we study the influence of various options of the parameter estimation algorithm, e.g. the globalization strategy, numerically. Especially the **Restrictive Monotonicity Test with Back Projection** is best suited for the mechanical systems we examine here.

Due to the size of the resulting software, we encounter the difficulty of implementing these methods in one package and to maintain usability, maintainability and flexibility. As we attach high importance to these properties of the code, we apply recent results of software engineering to design the object–oriented packages Parfit++ and MSBNAT, the first of which implements the parameter estimation algorithm, while the second provides an interactive tool to model and evaluate biomechanical models. We implement both packages based on a **three–tier model**. Furthermore, we use **interface–oriented programming** to modularize the code and **levelization** when an additional overhead is not desirable.

These mathematical and numerical results implemented in Parfit++ and MBSNAT are studied based on several examples including the parameter estimation of a hexapod machine tool. By this, we show the performance of our methods.

Furthermore, we study the parameter estimation of a huge biomechanical model of a sitting human. We want to provide a validated model which can be used to evaluate the effect of vibrations on the human body, for example, in a car. Therefore, we focus on the reaction of the spine to vertical excitations of the car and model this part of the human body in more detail. The rigid body model consists of 17 bodies representing the skeletal system. Additionally, we include five bodies representing wobbling masses [GDSR87]. A simple model for the car consisting of six bodies is assumed. The motion of these bodies is restricted by constraints, which results in 82 kinematic degrees of free-

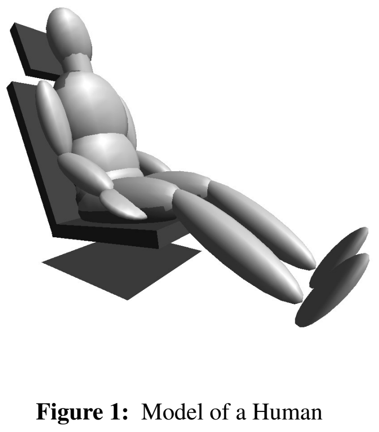


**Figure 1:** Model of a Human

dom. We use biomechanical forces with high stiffness coefficients to realistically represent the behavior of the soft tissue, which is important for studying the effect of vibrations. This property and the size of the model lead to a large computational effort even in the simulation context. As we want to do parameter estimation an efficient treatment is mandatory. We will present the identification of a selection of ten parameters associated with either idealized force–elements or parameters of the spine which are difficult to identify otherwise.

## Thesis Outline

The work is structured into five parts each of which starts with a more detailed introduction and is divided into several chapters.

Part I In this part we introduce the classical parameter estimation used in the ODE context. We present an overview of the well–known theory of parameter estimation.

- Ch. 1 We first introduce the GGN method for a general class of parameter estimation problems. This includes the formulation of the problem, optimality conditions, convergence properties and globalization strategies (including a short description of the Restrictive Monotonicity Test).
- Ch. 2 Then we specialize to dynamical systems. We present a short overview of different types of differential equations, including sensitivity generation and integration schemes to solve these equations. Furthermore, we introduce the multiple shooting discretization.
- Ch. 3 At last we discuss further topics concerning parameter estimation. We present how to handle multiple experiment structures and how to automatically generate good initial values and present a sensitivity analysis of the solution. Furthermore, we briefly discuss differential equations with discontinuities and their consequence for parameter estimation.

Part II The second part is dedicated to the introduction of the modelling of multi body systems based on Natural Coordinates including biomechanical extensions. We will restrict ourselves to the simulation context.

- Ch. 4 We first introduce our version of Natural Coordinates for modelling multi body systems. This chapter includes the description of the block–oriented solver to decompose and solve the linear system in linear complexity. Moreover, we exploit the sub–block–structure to provide a very efficient algorithm.
- Ch. 5 Furthermore, we focus on the more advanced topics of alternative formulations, constraint partitioning and singular kinematic positions to treat redundant constraints and event handling. Thus, we are able to model arbitrary topologies.

- Ch. 6 Additionally, we extend to the field of biomechanics. We show how to cope with the addition of different 3D forces and wobbling masses, ground contacts and muscles, which is needed in biomechanics.

Part III Next, we discuss the various adaptations to the parameter estimation algorithms presented in part I and the formulation of the mechanical DAE in part II, if we combine them.

- Ch. 7 Therefore, we focus on general properties of the variational differential equation and on the efficient calculation of one directional derivative in linear complexity w.r.t. the number of bodies.
- Ch. 8 In this chapter we discuss how to handle inconsistent initial values for the mechanical DAE and the corresponding variational differential equation. In particular, we present the limit form of non–stiff Baumgarte relaxation in combination with a projection which is adapted to the mechanical systems.
- Ch. 9 At last we discuss various approaches that reduce the number of directional derivatives one has to calculate explicitly.

Part IV In this part we present the software engineering background applied to the implementation of the methods presented in the first three parts.

- Ch. 10 We first give a brief overview of object–oriented programming and motivate to apply techniques of software design as described in the next chapter.
- Ch. 11 Then we present different approaches, like the three–tier model, levelization and interface–oriented programming, to cope with the complexity of object–oriented software. Furthermore, we concentrate on how they were applied in the design of Parfit++ and MBSNAT, which have been implemented as a result of this work.

Part V In the last part we present numerical results.

- Ch. 12 The linear complexity w.r.t. the number of bodies and the efficiency of our modelling approach in the simulation and optimization context is shown numerically.
- Ch. 13 Then we compare the options presented before when applying parameter estimation to mechanical models based on Natural Coordinates.
- Ch. 14 As the highlight, we present the solution of a parameter estimation problem based on a high–dimensional biomechanical problem in 3D.

# Part I

# Parameter Estimation

To study nature, it is possible to use mathematical models which describe the real process. Besides choosing the correct type of equation, it is necessary to adapt some numerical values of the model. A convenient way to adapt these so–called parameters to the specific process – and to estimate how good these values are – is to use parameter estimation (PE).

This part is dedicated to presenting the well–known theory of the (Generalized) Gauß–Newton method, which is used to solve the non–linear parameter estimation problem. For this purpose, we first recapitulate optimality conditions, convergence theorems and globalization strategies. In the first chapter, the methods are applied to algebraic problems with general nonlinear conditions. Not until in the second chapter will we extend to dynamic problems, i.e. problems with an underlying differential equation. Doing this, we first want to stress that the same theory and algorithms presented in the first chapter is used w.r.t. various types of parameter estimation problems, which is represented in the structure of the implementation (see part IV). Second, we want to emphasize that in order to treat the dynamic optimization problem we are interested in, it is necessary to solve several underlying algebraic sub problems.

To solve dynamic PE problems, one has to solve the underlying differential equation using integration schemes, which must be able to provide first order derivatives needed for the derivative based Gauß–Newton method. Multiple shooting discretization is used to increase the performance and stability of the method, while the resulting structure of the Jacobian matrix is exploited by condensing algorithms to be efficiency.

After describing the basic algorithm, several more advanced topics like multiple experiment problems, automatic generation of good initial values using measurement information, discontinuous differential equations and the statistical analysis of the solution are discussed in the last chapter of this part. Again we stick to PE problems of general type not specializing to mechanical systems. After discussing the modelling (and simulation) of mechanical systems in part II, we then adapt the optimization methods of this part to this special class of differential equations.

Chapter 1

# The Generalized Gauß–Newton Method

In this first chapter we discuss non–linear constrained least squares problems and their solution with a generalization of the Gauß–Newton method proposed by Bock [Boc81],[Boc83],[Boc87]. At first we will restrict ourselves to algebraic problems without an underlying differential equation. This contributes to a modularization of the theoretical presentation, which finds its counterpart in the software implementation presented in part IV. Furthermore, solving algebraic problems is a task encountered several times as a subproblem of a dynamic parameter estimation problem (see section 3.2). In the next chapter, we treat problems which are constrained by differential equations of various types.

We first present the general form of parameter identification problems as non–linear constrained least squares problems. Carrying on, these non–linear problems will be solved using the Gauß–Newton method, which needs derivatives of first order. In the context of parameter estimation, this method is typically able to perform better than methods using derivatives of higher order like SQP [Wil63], [Han76], [Pow78] or methods like the Nelder–Mead algorithm [GMW81],[NM65] which do not need derivative information. The preconditions and optimality conditions needed are briefly shown in the next section, followed by the local contraction theorem, which proves linear convergence for well–defined problems. Finally, we will show several globalization strategies to (theoretically) achieve a solution for arbitrary initial guesses.

## 1.1 Problem Definition

The aim of this section is to study the class of nonlinear, constrained least squares problems, that is finding the minimum of a nonlinear function in some norm –

typically the $l_2$–norm – w.r.t. equality and inequality constraints

$$\min_x \|r_1(x)\|_2^2 \tag{1.1}$$

$$r_2(x) = 0 \tag{1.2}$$

$$r_3(x) \geq 0 \quad , \tag{1.3}$$

where $r_i \epsilon C^3(D, \mathcal{R}^{n_i})$, and we have $n_1$ least squares conditions and $n_2$ respectively $n_3$ equality or inequality constraints.

The solution method we use is a damped Generalized Gauss-Newton method, that is we iteratively

$$x_{i+1} = x_i + \alpha_i \Delta x_i \tag{1.4}$$

solve the nonlinear problem where the increment is the solution of a linearized problem

$$\min_{\Delta x_i} \|J_1(x_i)\Delta x_i + r_1(x_i)\|_2^2 \tag{1.5}$$

$$J_2(x_i)\Delta x_i + r_2(x_i) = 0 \tag{1.6}$$

$$J_3(x_i)\Delta x_i + r_3(x_i) \geq 0 \quad , \tag{1.7}$$

with Jacobian $J_j(x) := \frac{\partial r_j(x)}{\partial x}$. Appropriate relaxation factors $\alpha_i$ (see section 1.3.2) are chosen for every step to globalize the convergence of the method.

In the case of parameter estimation, we have measurement data $\eta_i$ corresponding to observation functions $o(x)$

$$\eta_i = o_i(x) + \epsilon_i, \tag{1.8}$$

with a measurement error of $\epsilon_i$. Therefore, the *least squares* condition $r_1$ is specified further by

$$\min_x \|r_1(x)\|_2^2 = \min_x \sum_i \|\sigma_i^{-1}\left(\eta_i - o_i(x)\right)\|_2^2. \tag{1.9}$$

This weighted minimization w.r.t. $l_2$–norm yields a Maximum–Likelihood estimation, if the following conditions are fulfilled. The measurement errors $\epsilon_i$ must be independent and normally distributed with zero mean. We choose $\sigma_i^{-1}$, i.e. the inverse standard deviation of $\epsilon_i$, as the weight.

## 1.2 Optimality Conditions

To be able to solve this problem, a number of criteria have to be fulfilled. For the ease of presentation we restrict ourselves to equality constrained problems in the following sense:

**Definition 1 (Active Inequality Constraints)**
*We define the index set of the active inequality constraints by*

$$I(x) := \{i \in \{1, ..., n_3\} | \, r_{3i}(x) = 0\} \tag{1.10}$$

*at some state $x$, furthermore*

$$r_c(x) = \begin{pmatrix} r_2(x) \\ \bar{r}_3(x) \end{pmatrix} \quad , \tag{1.11}$$

*where $\bar{r}_3$ consists of all components of $r_2$ and the components $r_{3,i}$ with $i \in I(x)$ of $r_3$.*

Using this definition, it is possible to show that a modified problem

$$\min_x \|r_1(x)\|_2^2 \tag{1.12}$$

$$r_c(x) = 0 \tag{1.13}$$

with the corresponding linearized problem

$$\min_x \|J_1(x)\Delta x + r_1(x)\|_2^2 \tag{1.14}$$

$$J_c(x)\Delta x + r_c(x) = 0 \tag{1.15}$$

is equivalent to the original problem in a neighborhood of the solution.

Problems containing inequality constraints are solved based on active set strategies as described in [Boc87]. As the topic of inequality constraints and active set strategies is not a central part of this work, it will not be discussed further.

**Definition 2 (Constraint Qualification (CQ))**
*The Constraint Qualification is satisfied if*

$$Rg(J_c(x)) = n_c = n_2 + |I(x)| \quad . \tag{1.16}$$

*A state $x \epsilon D$ is said to be regular if x satisfies* **CQ**.

**Definition 3 (Positive Definitness (PD))**

$$\mathbf{PD}: \qquad Rg \begin{pmatrix} J_1(x) \\ J_c(x) \end{pmatrix} = n \qquad \forall \; x \epsilon D \quad , \tag{1.17}$$

*is equivalent to*

$$z^T J_1^T J_1 z > 0 \qquad \forall \; z \neq 0 \vee J_c z = 0. \tag{1.18}$$

**Definition 4 (Lagrangian Function)**
*The Lagrangian function for the nonlinear problem (1.12) - (1.13) is defined as*

$$L(x, \lambda) = \frac{1}{2}\|r_1(x)\|_2^2 - \lambda^T r_c(x) \quad , \tag{1.19}$$

*where $\lambda \; \epsilon \; \mathcal{R}^{n_c}$ is called* Lagrange multiplier.

Using these definitions, we are able to formulate the basic theorem.

**Theorem 1 (Necessary Conditions)**
*Let $x^*$ be regular and a solution of (1.12)-(1.13). Then $x^*$ is feasible, i.e. $r_c(x^*) = 0$, and the first order necessary condition holds: There exists a multiplier vector $\lambda^*$ that uniquely solves the stationary condition*

$$\frac{\partial}{\partial x} L(x^*, \lambda^*) = r_1(x^*)^T J_1(x^*) - \lambda^{*T} J_c(x^*) = 0 \quad . \tag{1.20}$$

*Additionally, the second order necessary condition is satisfied: For all directions*

$$w \neq 0 : \qquad J_c(x^*)w = 0$$

*the Hessian matrix of L is positive semi-definite*

$$w^T H(x^*, \lambda^*) w \geq 0, \qquad H(x^*, \lambda^*) := \frac{\partial^2}{\partial x^2} L(x^*, \lambda^*) \quad . \tag{1.21}$$

**Definition 5 (Karush-Kuhn-Tucker Point (KKT Point))**
*A vector $(x^*, \lambda^*)$, which is feasible and satisfies the stationary condition (1.20), is called a Karush-Kuhn-Tucker point (KKT point).*

**Lemma 1**
*If $CQ$ (2) and $PD$ (3) are satisfied, then the following equivalence holds:*

$$\begin{array}{ll} & (x^*, \lambda^*) \text{ is a KKT point of (1.12)-(1.13)} \\ \iff & (0, \lambda^*) \text{ is a KKT point of (1.14)-(1.15)} \quad , \end{array}$$

*i.e. a KKT point of the nonlinear problem (1.12)-(1.13) is a fixed point of the Gauss-Newton iteration.*

A great advantage of the Generalized Gauss-Newton method is that a generalized inverse $J^+$ can be defined, which solves the linearized problem (1.14)-(1.15). This allows for a uniform theoretical treatment of nonlinear equation systems, unconstrained and constrained least squares problems.

**Theorem 2 (Existence of a Generalized Inverse)**
*If $J = (J_1^T, J_c^T)^T$ in the linear constrained least squares problem (1.14)-(1.15) satisfies the conditions $[CQ]$ and $[PD]$, we can conclude:*

1. *For any $r = (r_1^T, r_c^T) \in \mathcal{R}^{n_1+n_c}$ there is exactly one Karush-Kuhn-Tucker point $(\Delta x, \lambda)$ for (1.14)-(1.15), and $\Delta x$ is a strict minimum.*
2. *There exists a linear mapping $J^+ : \mathcal{R}^{n_1+n_c} \longrightarrow \mathcal{R}^n$, such that*

$$\Delta x = -J^+ r \tag{1.22}$$

*solves (1.14)-(1.15) for any $r \in \mathcal{R}^{n_1+n_c}$ with*

$$J^+ = \begin{pmatrix} \mathcal{I} & 0 \end{pmatrix} \begin{pmatrix} J_1^T J_1 & J_c^T \\ J_c & 0 \end{pmatrix}^{-1} \begin{pmatrix} J_1^T & 0 \\ 0 & \mathcal{I} \end{pmatrix} \quad . \tag{1.23}$$

3. *The solution operator $J^+$ is a generalized inverse and satisfies the defining condition*
$$J^+ = J^+ J J^+. \tag{1.24}$$

## 1.3 Convergence Properties

### 1.3.1 Local Convergence

The *Local Contraction Theorem* of Bock [Boc87] provides conditions for the local convergence of the Generalized Gauss-Newton method and it also quantifies the convergence. This theorem also holds for the unconstrained case, where $J^+$ becomes the Moore-Penrose pseudo-inverse, and for nonlinear equation systems, where $J^+$ reduces to the normal inverse $J^{-1}$. In the latter case, convergence can also be shown for suitable approximations of $J^{-1}$.

**Theorem 3 (Local Convergence Theorem)**
*Let $J(x)^+$ be the generalized inverse of the Jacobian $J(x) = (J_1(x)^T, J_c(x)^T)^T$ with respect to the function $r = (r_1^T, r_c^T)^T \in C^1(D, \mathcal{R}^{n_1+n_c})$ in a nonlinear constrained least squares problem. The Jacobian $J$ respectively its generalized inverse $J^+$ satisfy the following Lipschitz conditions:*

$$||J(y)^+\big(J(x+t(y-x)) - J(x)\big)(y-x)|| \le \omega t ||y-x||^2, \qquad \omega < \infty \tag{1.25}$$

$$||\big(J(z)^+ - J(x)^+\big)R(x)|| \le \kappa(x)||z-x|| \le \kappa||z-x|| \tag{1.26}$$

$$R(x) := r(x) - J(x)J(x)^+ r(x) \qquad \text{(residuum)} \qquad \kappa < 1$$

$\forall\, t \in [0,1],\ \forall\, x, y, z \in D$ *and* $x - y = J(x)^+ r(x)$.

*Then for all $x_0 \in D$ with*

$$\delta_0 := \frac{\alpha_0 \omega}{2} + \kappa < 1, \qquad \alpha_j := ||J(x_j)^+ r(x_j)|| \tag{1.27}$$

$$D_0 := K\big(x_0, \frac{\alpha_0}{1-\delta_0}\big) \subset D \tag{1.28}$$

*follows:*

1. *The iteration*
$$x_{i+1} = x_i + \Delta x_i, \qquad \Delta x_i = -J(x_i)^+ r(x_i) \tag{1.29}$$
*is well-defined and remains in $D_0$.*

2. $x_i \to x^* \in D_0$ $(j \to \infty)$ *with* $J(x^*)^+ r(x^*) = 0$.

3. *The convergence is linear with*
$$||\Delta x_{i+1}|| \le \big(\frac{\alpha_i \omega}{2} + \kappa\big)||\Delta x_i|| =: \delta_i ||\Delta x_i||.$$

4. *For the $i$th iteration the following a priori estimation holds*

$$||x_i - x^*|| \leq \delta_0^i \frac{\alpha_0}{1-\delta_0}.$$

**Remark 1**

*The Lipschitz constant $\omega$ characterizes the nonlinearity of the model. Its inverse limits the region in which the linearization is valid. See also the next sub–section (1.3.2).*

*The Lipschitz constant of $J^+$, $\kappa$, measures the compatibility of the data with the model. For $\kappa < 1$, the fixed point $x^*$ is not only a stationary point but a strict local minimum. In contrast to this, for $\kappa \geq 1$ there may exist perturbations of the measurement data in the same order of magnitude as the measurements, such that the fixed point is stationary, but not a minimum ([Boc87]).*

### 1.3.2 Globalization of Convergence

Considering highly complex and non–linear real-life parameter estimation problems, one often can not provide initial guesses $x_0$ that are sufficiently close to the solution to satisfy the requirements of the *Local Convergence Theorem.*

One way to globalize the convergence of the method is by damping or underrelaxation. The iterations are then defined by

$$x_{k+1} = x_k + \alpha_k \Delta x_k \qquad \alpha_k \in (0,1] \tag{1.30}$$

with a relaxation factor $\alpha_k$. The relaxation factor is chosen such that the next iterate $x_{k+1}$ is "better" than $x_k$. As a measurement of convergence, we ask for descent of a suitably chosen level function $T$. Because of lemma (1), we reach a stationary point if a descent w.r.t. such a level function is no longer possible. The choice of the level function is crucial for fast convergence. A good choice is the iteratively weighted natural level function

$$T^k(\epsilon) := \|J^+(x_k) r(x_k + \epsilon \Delta x_k)\|_2 \quad . \tag{1.31}$$

Out of this condition and an estimate of the nonlinearity $\omega$, we are able to estimate a suitable steplength without a complete linesearch. For further study, see [Boc81], [Boc87] and [Sch88].

Additionally, we want to consider a new approach, the so–called restrictive monotonicity test (RMT). This technique may be interpreted as a stepsize strategy analogous to those used in numerical methods for the discretization of ODE with invariants for the Davidenko differential equation associated with the solution of the parameter estimation problem (for a more detailed discussion see [BKS00]).

It is possible to additionally apply back projection

$$\begin{array}{rcl} \bar{x}_{k+1} & = & x_k + \alpha_k \Delta x_k \\ x_{k+1} & = & \bar{x}_{k+1} + \bar{\Delta} x_k \end{array} \tag{1.32}$$

with

$$\bar{\Delta} x_k = -J(x_k)^+ \left(r(\bar{x}_{k+1}) - (1-\alpha_k) r(x_k)\right) \tag{1.33}$$

instead for (1.30) as a form of second order error correction [BKS00]. Doing this, we better follow the trajectory defined by the Davidenko differential equation.

## 1.4 Solution of Constrained Minimization Problems

To solve the nonlinear problem, it is necessary to repeatedly solve the linearized problem (see equations 1.14 and 1.15)

$$\begin{aligned} \min_{\Delta x} & \|J_1(x)\Delta x + r_1(x)\|_2 \\ & J_c(x)\Delta x + r_c(x) = 0. \end{aligned}$$

For the solution of this linear constrained least squares problem, the first step is to eliminate part of the variables using the constraints. One possibility is to apply an LQ–decomposition, which results in the following constrained system

$$\begin{pmatrix} L_c & 0 \end{pmatrix} Q \begin{pmatrix} \Delta x_c \\ \Delta x_l \end{pmatrix} = \begin{pmatrix} L_c & 0 \end{pmatrix} \begin{pmatrix} \Delta \tilde{x}_c \\ \Delta \tilde{x}_l \end{pmatrix} = -r_c(x) \tag{1.34}$$

and, therefore,

$$\Delta \tilde{x}_c = -L_c^{-1} r_c \tag{1.35}$$

After applying this transformation to the minimization part and eliminating the part corresponding to the known increment $\Delta \tilde{x}_c$, we get:

$$\min_{\Delta \tilde{x}_l} \|\tilde{J}_1^l \Delta \tilde{x}_l + \tilde{r}_1(x)\|_2 \tag{1.36}$$

where

$$\begin{pmatrix} J_1^c & J_1^l \end{pmatrix} = \begin{pmatrix} \tilde{J}_1^c & \tilde{J}_1^l \end{pmatrix} Q \tag{1.37}$$

and

$$\tilde{r}_1(x) = r_1(x) - \tilde{J}_1^c \Delta L_c^{-1} r_c \tag{1.38}$$

This linear unconstrained minimization problem is solved using a QR–decomposition. To establish the complete increment we compute

$$\begin{pmatrix} \Delta x_c \\ \Delta x_l \end{pmatrix} = Q^T \begin{pmatrix} \Delta \tilde{x}_c \\ \Delta \tilde{x}_l \end{pmatrix}. \tag{1.39}$$

# Chapter 2

# Parameter Estimation for Dynamic Systems

While we previously focused on the properties of parameter estimation problems, which are also valid for algebraic ones, we now focus on problems constrained by an differential equation.

For simplicity reasons we will start with an Ordinary Differential Equation (ODE)

$$\dot{y} = f(t, y, q) \tag{2.1}$$

with time $t$, differential variables $y(t)$ (states) and parameters $q$. In this case, the least squares conditions and constraints are functions of the states at different time points and the parameters

$$r_i(x) = r_i(y(t_0), y(t_1), ..., y(t_k), q) \quad . \tag{2.2}$$

When we examine least squares conditions, the time points $t_i$ correspond to the time of measurement, while constraints can be initial conditions at time $t_0$, boundary conditions at times $t_0$ and $t_k$ or interior point conditions.

One possibility to parameterize, is to use initial states $y_0$ and parameters $q$, where

$$y(t_0) = y_0. \tag{2.3}$$

Then the optimization variables $x$ are $\begin{pmatrix} y_0^T & q^T \end{pmatrix}$. A different parameterization will be presented in section 2.3.

In many cases, the general form of conditions (2.2) can be simplified to a more specific form

$$r_i(x) = \sum_{j=1}^{M} r_i^j(y(t_j), q) \quad , \tag{2.4}$$

where the function $r_i$ is written as a sum of parts only depending on the state at one time point and the parameters. A typical situation in parameter estimation problems is that each component of $r_i$ does only depend on the state at *one* time–point.

As a result, we formulate a non–linear parameter estimation problem with an underlying differential equation (here ODE)

$$\min_{y_0,p} \|r_1(y(t_1;y_0,q),y(t_2;y_0,q),...,y(t_k;y_0,q),q)\|_2^2 \tag{2.5}$$

$$r_c(y(t_1;y_0,q),y(t_2;y_0,q),...,y(t_k;y_0,q),q) = 0 \tag{2.6}$$

$$\dot{y} = f(t,y,q), \quad y(t_0) = y_0 \tag{2.7}$$

## 2.1 Differential Algebraic Equations

The differential equation (2.1) is an ODE. In several applications it is useful to introduce algebraic conditions $g$ with algebraic variables $z$. This leads to a Differential Algebraic Equation (DAE)

$$A(t,y,z,q)\dot{y} \;=\; f_1(t,y,z,q) \tag{2.8}$$

$$0 \;=\; g(t,y,z,q) \quad , \tag{2.9}$$

where $A$ is a regular matrix and $f_1$ the right–hand side.

**Definition 6 (Differential Index)**
*Equation (2.9) has the differential index $i_d = k$, if $k$ is the minimum number of total differentiations*

$$g(t,y,z,q) = 0, \qquad \dot{g}(t,y,z,q), \qquad ... \qquad , g^{(k)}(t,y,z,q) \quad , \tag{2.10}$$

*such that $g$, $\dot{g}$, ..., $g^{(k)}$ can be transformed into an explicit ODE $\dot{z} = \tilde{g}(t,y,z,q)$ using algebraic manipulations.*

DAE's of higher index are often reduced to index 1.

### 2.1.1 DAE Index 3 (Full Descriptor Form)

Models of mechanical systems can be formulated in redundant coordinates as a DAE of index 3. To enhance readability we omit the parameters in the following.

$$M(p)\ddot{p} + G(p)^T\lambda \;=\; f_3(t,p,\dot{p}) \tag{2.11}$$

$$g_{pos}(p) \;=\; 0 \quad . \tag{2.12}$$

Here $M$ is the so–called mass matrix, $f_3$ are forces, $g_{pos}$ are kinematic constraints and $G$ is their derivative w.r.t. the positions $p$. Reformulation to order 1 and reduction of the index 3 constraint $g_{pos}$ to index 1 leads to the following linear system:

$$\begin{pmatrix} M(p) & G(p)^T \\ G(p) & \end{pmatrix} \begin{pmatrix} a \\ \lambda \end{pmatrix} = \begin{pmatrix} f_3(t,p,v) \\ \gamma(t,p,v) \end{pmatrix} \tag{2.13}$$

where

$$\begin{array}{rcl} G(p) & := & \frac{\partial}{\partial p} g_{pos}(p) \\ \gamma(p,v) & := & \frac{d^2}{dt^2} g_{pos}(p) \text{ - } G(p)a \quad . \end{array}$$

In this context, position variables $p$ and velocity variables $v := \dot{p}$ are dynamic variables, while the Lagrange multipliers $\lambda$ and the acceleration $a := \ddot{p}$ are algebraic variables.

Furthermore, it should be possible to include behavior which is not described by a multi–body approach, i.e. introduce so–called variables of external dynamic $e$ and corresponding differential equations.

This previous constraint $g_{pos}$ is of index 3 and is implicitly included by the index 1 acceleration constraint (the lower part of equation (2.13)). The original kinematic constraints $g_{pos}$ and the corresponding first derivatives $g_{vel}$ are known as invariants, and are used to project to the corresponding manifold. This avoids a drift of the numerical solution away from the manifold defined by the position and velocity invariants [Sha86], [Eic92].

The inclusion of external variables and invariants results in the so–called *Full Descriptor Form*:

$$\begin{array}{rcl} \dot{e} & = & f_e(t,p,v,e) \\ \dot{p} & = & v \\ \dot{v} & = & a \end{array} \tag{2.14}$$

$$\begin{pmatrix} M(p,e) & G(p,e)^T \\ G(p,e) & \end{pmatrix} \begin{pmatrix} a \\ \lambda \end{pmatrix} = \begin{pmatrix} f_3(t,p,v,e) \\ \gamma(t,p,v,e) \end{pmatrix} \tag{2.15}$$

with invariants:

$$\begin{array}{rcl} g_{pos}(p,e) & = & 0 \\ g_{vel}(p,v,e) = \frac{d}{dt} g_{pos}(p,e) = G(p,e)v & = & 0. \end{array} \tag{2.16}$$

Here we focus on the mathematical formulation. To be able to solve the linear system (2.15), $G$ must have full rank. Furthermore, $M$ must be positive definit on the null space $ker(G)$ of $G$. For a further discussion we refer to the part II.

**Remark 2 (Non-Autonomous and Non–Holonomic constraints)**
*Besides purely kinematic constraints $g_{pos}(p)$, there are explicit time–dependent constraints $g_{naut}(t,p)$ or constraints which can not be described w.r.t. position variables only $g_{nhol}(t,p,v)$. These types of constraints are easily included in the scheme above and do not present any further numerical problems [Che96]. For notational reasons, these types will be omitted further on.*

An up–to–date and detailed reference of several tailored integration schemes to solve mechanical DAEs in Full Descriptor Form can be found, for example, in [Sch99] and [Win01]. In this work, we restrict ourselves to the use of these algorithms and as such will not go into detail.

## 2.2 Sensitivity Generation

Besides supplying a function evaluation, derivative based optimization algorithms like the Generalized Gauß–Newton method need the first derivative of the defining functions (2.2) w.r.t. optimization variables:

$$\frac{\partial r_i(x)}{\partial x} = \frac{\partial}{\partial(y_0,q)} r_i(y(t_1;y_0), y(t_2;y_0), ..., y(t_k;y_0), q) \quad . \tag{2.17}$$

A function may be defined at several time-points which are in general not the start of the underlying integration interval. Therefore, the derivative of the conditions w.r.t. the optimization variables can be written as

$$\frac{\partial r_i(x)}{\partial x} = \frac{\partial r_i}{\partial y(t_1)}\frac{\partial y(t_1)}{\partial(y_0,q)} + \frac{\partial r_i}{\partial y(t_2)}\frac{\partial y(t_2)}{\partial(y_0,q)} + \ldots + \frac{\partial r_i}{\partial y(t_k)}\frac{\partial y(t_k)}{\partial(y_0,q)} + \frac{\partial r_i}{\partial q}. \tag{2.18}$$

Every part, except for the derivative w.r.t. the parameters, is separated into a derivative of $r_i$ w.r.t. the states at this time–point and the derivative of this state w.r.t. initial states and parameter.

We now take a closer look at the second part:

$$\begin{pmatrix} W(t,t_0) & W^q(t,t_0) \end{pmatrix} := \begin{pmatrix} \frac{\partial y(t;y_0,q)}{\partial y_0} & \frac{\partial y(t;y_0,q)}{\partial q} \end{pmatrix}. \tag{2.19}$$

One way to calculate this is to solve the *variational differential equation* corresponding to the differential equation (here shown for the ODE case (2.1)):

$$\begin{aligned} \begin{pmatrix} \dot W(t,t_0) & \dot W^q(t,t_0) \end{pmatrix} &:= \begin{pmatrix} \frac{\partial \dot y(t)}{\partial y_0} & \frac{\partial \dot y(t)}{\partial q} \end{pmatrix} \\ &= \begin{pmatrix} \frac{\partial f(y,q)}{\partial y}\frac{\partial y}{\partial y_0} & \frac{\partial f(y,q)}{\partial y}\frac{\partial y}{\partial q} + \frac{\partial f(y,q)}{\partial q} \end{pmatrix} \\ &= \begin{pmatrix} \frac{\partial f}{\partial y} W(t,t_0) & \frac{\partial f}{\partial y} W^q(t,t_0) + \frac{\partial f}{\partial q} \end{pmatrix} \end{aligned} \tag{2.20}$$

with initial values

$$\begin{pmatrix} W(t_0,t_0) & W^q(t_0,t_0) \end{pmatrix} = \begin{pmatrix} \frac{\partial y(t_0)}{\partial y_0} & \frac{\partial y(t_0)}{\partial q} \end{pmatrix} = \begin{pmatrix} \mathcal{I} & \mathcal{O} \end{pmatrix} \quad . \tag{2.21}$$

A detailed study for mechanical DAE will be presented in chapter 7. For DAE of index 1 we refer, for example, to [Bau99].

In some cases, only a subspace $\bar D$ is of interest, then we restrict the derivative generation to this part. The corresponding initial directions are

$$D(t_0,t_0) = D_0 \quad , \tag{2.22}$$

where $D_0 \in \mathcal{R}^{n \times n_{Dir}}$ is a basis of the tangent space of the subspace $\bar D$ at $y(t_0)$ and $n, n_{Dir}$ are the number of states respectively the dimension of the subspace. Derivatives in this form are called *directional derivatives*.

### 2.2.1 Internal Numerical Differentiation

To solve a variational differential equation corresponding to a differential equation one has to keep in mind, that the numerical solution of the DE is typically done based on an integration scheme with an adaptive stepsize and order strategy. To avoid errors of the derivative generation due to this, we use internal numerical differentiation (IND) to provide the derivative of the numerical solution of the DE w.r.t. initial values and parameters. For further readings we refer to [Boc85].

## 2.3 Multiple Shooting

One possibility to treat the dynamic parameter estimation problem (2.5,2.6,2.7) is to solve an initial value problem (IVP) (2.7). The solution of the IVP may not exist for arbitrary initial states or parameters. Even if there is not such an extreme situation, the initial trajectory may be far off the solution trajectory – with corresponding bad derivative information – and may introduce a high nonlinearity into the optimization problem. One solution is to introduce the multiple shooting discretization as in [Boc85].

### 2.3.1 Discretization

We subdivide the integration interval, inspired by the methods used for multi–point boundary value problem approaches (BVP). For a suitable grid of the integration interval $[t_0, t_{end}]$

$$t_0 = \tau_0 < \tau_1 < ... < \tau_{m+1} = t_{end}, \tag{2.23}$$

we introduce additional optimization variables $s_i$ which are initial values for the differential equations defined on each sub–interval. Using information originating from measurements, constraints or additional input, we are able to provide "good" initial values for the states at the multiple shooting nodes [Boc85],[Jac01]: The corresponding initial trajectory is discontinuous, but exists on the complete interval and is near to the solution trajectory. Furthermore, the nonlinearity of the parameter estimation problem may be reduced.

This leads to the following set of optimization variables

$$x^T = \left(s_0^T, s_1^T, ..., s_m^T, q\right), \tag{2.24}$$

where $s_i$ is used as the initial values of the integration interval $[\tau_i, \tau_{i+1})$. To eventually achieve a continuous solution trajectory, we have to state additional matching conditions:

$$h_i(s_i, s_{i+1}, q) := y(\tau_{i+1}; s_i, q) - s_{i+1} = 0 \qquad , \quad i = 0, ..., m-1 \tag{2.25}$$

At last, we formulate the parameter estimation problem using multiple shooting discretization:

$$\min_{s_0,..,s_m,q} \|r_1(s_0, s_1, ..., s_m, q)\|_2^2 \tag{2.26}$$

$$r_c(s_0, s_1, ..., s_m, q) = 0 \tag{2.27}$$

$$h_j(s_j, s_{j+1}, q) = 0 \quad \forall j \in \{0, ..., m-1\} \tag{2.28}$$

and an underlying differential equation

$$\dot{y}(t) = f(t, y(t; q, s_i), q) \quad t \in [\tau_i, \tau_{i+1}) \tag{2.29}$$

$$y(\tau_i) = s_i \tag{2.30}$$

### 2.3.2 Condensing Algorithm

The multiple shooting approach drastically increases the dimension of the parameter estimation problem. Fortunately, it exhibits a special structure of the Jacobian that can be exploited, which mainly offsets the increased computational cost resulting from a higher dimension.

Looking at the solution of the corresponding linearized system we recognize the following structure

$$\begin{pmatrix} \Delta s_0 \\ \Delta s_1 \\ \vdots \\ \Delta s_m \\ \Delta q \end{pmatrix} = - \begin{pmatrix} D_0 & D_1 & D_2 & \dots & D_m & D_q \\ H_0 & -\mathcal{I} & & & & H_0^q \\ & H_1 & -\mathcal{I} & & & H_1^q \\ & & \ddots & & & \vdots \\ & & & H_{m-1} & -\mathcal{I} & H_{m-1}^q \end{pmatrix}^+ \begin{pmatrix} d \\ h_0 \\ h_1 \\ \vdots \\ h_{m-1} \end{pmatrix} , \tag{2.31}$$

where

$$d = \begin{pmatrix} r_1 \\ r_c \end{pmatrix} \quad , \quad D_i = \frac{\partial d}{\partial s_i} \quad , \quad D_q = \frac{\partial d}{\partial q} \quad ,$$

$$H_i := \frac{\partial h_i(s_i, s_{i+1}, q)}{\partial s_i} \quad , \quad H_i^q := \frac{\partial h_i(s_i, s_{i+1}, q)}{\partial q} \quad .$$

The condensing algorithms presented here use the structure of (2.31) to efficiently reduce the dimension. The computational cost is typically negligible compared to that of solving the differential equation and variational differential equation.

In the following, we will study the block–oriented Gauß–decomposition. The $-\mathcal{I}$ matrix is used to eliminate one $H_i$, $H_i^q$-block. In part III we will adapt this form of the condensing algorithm to mechanical systems.

We define

$$\tilde{D}_m := D_m, \quad \tilde{D}_q^{(m)} := D_q \quad . \tag{2.32}$$

For $i = m-1, ..., 0$ we apply column transformations to parts of the Jacobian and the increment:

$$\begin{pmatrix} D_i & \tilde{D}_{i+1} & \tilde{D}_q^{(i+1)} \\ -\mathcal{I} & & H_{i-1}^q \\ H_i & -\mathcal{I} & H_i^q \end{pmatrix} \begin{pmatrix} \Delta s_i \\ \Delta s_{i+1} \\ \Delta q \end{pmatrix}$$

$$\Downarrow$$

$$\begin{pmatrix} D_i + \tilde{D}_{i+1} H_i & \tilde{D}_{i+1} & \tilde{D}_q^{(i+1)} + \tilde{D}_{i+1} H_i^q \\ -\mathcal{I} & & H_{i-1}^q \\ & -\mathcal{I} & 0 \end{pmatrix} \begin{pmatrix} \Delta s_i \\ \Delta s_{i+1} - H_i \Delta s_i - H_i^q \Delta q \\ \Delta q \end{pmatrix}$$

$$=: \begin{pmatrix} \tilde{D}_i & \tilde{D}_{i+1} & \tilde{D}_q^{(i)} \\ -\mathcal{I} & & H_{i-1}^q \\ & -\mathcal{I} & \end{pmatrix} \begin{pmatrix} \Delta s_i \\ \Delta \tilde{s}_{i+1} \\ \Delta q \end{pmatrix}. \tag{2.33}$$

The resulting system is

$$\begin{pmatrix} \Delta s_0 \\ \Delta \tilde{s}_1 \\ \vdots \\ \Delta \tilde{s}_m \\ \Delta q \end{pmatrix} = - \begin{pmatrix} \tilde{D}_0 & \tilde{D}_1 & \tilde{D}_2 & \dots & D_m & \tilde{D}_q^{(1)} \\ & -\mathcal{I} & & & & \\ & & -\mathcal{I} & & & \\ & & & \ddots & & \\ & & & & -\mathcal{I} & \end{pmatrix}^+ \begin{pmatrix} d \\ h_0 \\ h_1 \\ \vdots \\ h_{m-1} \end{pmatrix}. \tag{2.34}$$

From this, we can easily calculate the increments $\Delta \tilde{s}_i$, $i \in 1, ..., m$

$$\Delta \tilde{s}_{i+1} = h_i \tag{2.35}$$

and get the so–called condensed system and a solution for $\Delta s_0$ and $\Delta q$

$$\begin{pmatrix} \Delta s_0 \\ \Delta q \end{pmatrix} = - \begin{pmatrix} \tilde{D}_0 & \tilde{D}_q^{(1)} \end{pmatrix}^+ \left( d + \sum_{i=0}^{m-1} \tilde{D}_{i+1} h_i \right). \tag{2.36}$$

To calculate the increments $\Delta s_i$, $i \in 1, ..., m$ we compute

$$\Delta s_{i+1} = h_i + H_i \Delta s_i + H_i^q \Delta q.$$

# Chapter 3

# Various Topics Concerning Parameter Estimation

The first two chapters of part I introduce a basic functionality to treat parameter estimation problems. In this chapter, we discuss several extensions needed for more demanding problems.

- We discuss parameter identification based on measurement data combining information of more than one experimental setting (*multiple experiments problem*). This type of problem shows a certain structure that can be exploited.

- A difficulty, especially with increasing problem dimension, is to *generate initial values* in the context of multiple shooting. In the case of parameter estimation, information in the form of measurements is available to automatically calculate good initial guesses in a preprocessing step.

- We examine problems containing an underlying differential equation with *discontinuities*. We will see that, under certain conditions, classical optimization methods can still be applied.

- The *statistical analysis* of the solution is studied as a postprocessing step.

## 3.1 Multiple Experiments Problems

In several occasions it is not possible to estimate all parameters using data evaluated in one type of experimental setting. Using several different settings provides additional information and allows for parameter estimation with better statistical properties. Furthermore, this typically improves the numerical condition number of the problem. This leads to the so–called multiple experiments problems. We formulate a problem that consists of set of $N$ experiments with diverse local parameters or states $x^i$ and a common set of global parameters $x^g$.

$$\min_{x^i,x_g} \sum_{i=1}^{N} \|r_1^i(x^i, x_g)\|_2^2 \tag{3.1}$$

$$r_c^i(x^i, x_g) = 0 \quad \forall i = 1, ..., N \tag{3.2}$$

The resulting problems are of high dimension, but the Jacobian exhibits a typical block-structure

$$\begin{pmatrix} J_{x^1}^1 & & & & J_{q_g}^1 \\ & J_{x^2}^2 & & & J_{q_g}^2 \\ & & \ddots & & \vdots \\ & & & J_{x^n}^n & J_{q_g}^n \end{pmatrix}, \tag{3.3}$$

where $J_{x^i}^i$ is the Jacobian of the $i$th experiment w.r.t. the corresponding local states $x^i$. $J_{x_g}^i$ is the Jacobian w.r.t. the global parameters $x_g$ of the $i$th experiment.

During the solution process, every block is treated separately, excluding the columns corresponding to the common parameters from pivoting strategies. The remaining sub–systems corresponding to the remainder of all $J_g^i$ are solved by standard minimization techniques. A detailed treatment can be found in [Sch88].

### 3.1.1 Using Multiple Experiment Formulation to Identify a Parameterized Surface

The multiple experiment formulation is applicable if one wants to identify a parameterized surface using measurements of the states at several points.

The surface is defined using a function $f$ of the form

$$f(x, q) = 0 \tag{3.4}$$

with states $x \in \mathcal{R}^n$, parameters $q \in \mathcal{R}^{n_p}$ and $f \in \mathcal{R}^{n+n_p} \to \mathcal{R}^{n_f}$. Because these parameters typically can not be estimated directly, it is useful to provide several points on the surface as measurements. This results in an algebraic constrained least squares problem

$$\begin{array}{c} \min_{x_i,q} \sum_i \| \frac{x_i - \hat{x}_i}{\sigma_i} \|_2^2 \\ f_i(x_i, q) = 0 \quad \forall i, \end{array} \tag{3.5}$$

where $x_i$ is the estimation of the state on the surface corresponding to the measurement $\hat{x}_i$, which includes a measurement error which is normally distributed with a standard deviation of $\sigma_i$. We typically use several of these measurements and, therefore, introduce multiple experiment problems. Every experiment corresponds to one set of states $x_i$, while every experiment have the parameters $q$ in common.

Especially in the case of complex surfaces, it is useful to combine measurement data from different experimental settings each measuring at several measurement points. This again has a multiple experiment structure which results in a stacked multiple experiment structure for the complete parameter estimation problem.

The resulting stacked multiple experiment is of the following form

$$\min_{x,q_l,q_g} \sum_{j=1}^{M} \sum_{i=1}^{N^j} \| \frac{x_i^j - \hat{x}_i^j}{\sigma_i^j} \|_2^2 \tag{3.6}$$

$$f_i^j(x_i^j, q_l^j, q_g) = 0 \quad \forall i = 1, ..., N^j \{\forall j = 1, ..., M\} \tag{3.7}$$

where $M$ denotes the number of experiments or series of measurements, $N^j$ the number of measurements in the $j$th series, $x_i^j$ the set of estimated states of the $j$th series and $i$th measurement, $x$ the complete set of $x_i^j$, $q_l^j$ the local parameter of the $j$th series, $q_l = \{q_l^1, ..., q_l^M\}$ all local parameters, $q_g$ the global parameters of all series and $f_i^j$ the constraints of the $j$th series and $i$th measurement point.

## 3.2 Automatic Generation of Good Initial Values

A crucial point of an iterativ method like the Generalized Gauß–Newton method is to have good initial values. We have shown in section (2.3) that by using multiple shooting we are able to give a better approximation of the solution trajectory. This is true, if we are able to provide good guesses for the initial values at the multiple shooting nodes.

The first option to provide initial guesses is to guess them. If we measure all states at several time–points and if we choose these time–points, or a selection of them, as multiple shooting nodes, we are easily able to use these measurements to achieve a non–continuous initial trajectory which fulfills all conditions except the matching conditions, and as such we have good initial values. Unfortunately, this simple scenario is not the typical case. In general, we measure only a selection of states or nonlinear functions of them, and the others must be given by hand.

Alternatively, one can provide initial values for the first multiple shooting node only and use the states, calculated by integration, at the end of a previous multiple shooting interval to initialize the others. The latter method may not succeed at all if the nonlinearity of the differential equation is so high that one is not able to even integrate till the end of the interval.

Better initial guesses may be provided by a mix of the methods mentioned above. We guess suitable initial values at the beginning of the complete interval. Using these initial values, we integrate to the first time–point $t_i$, at which a condition is stated (this may be the beginning of the integration interval and, therefore, supply good initial guesses for the start of the integration interval). At $t_i$ we solve an *algebraic* constrained optimization problem

$$\begin{array}{c} \min_{y(t_i)^+} \|y(t_i)^- - y(t_i)^+\|_2^2 \\ \bar{r}_i(t_i, y(t_i)^+, q) = 0 \end{array} \quad , \tag{3.8}$$

such that $y(t_i)^+$ fulfils all conditions $\bar{r}_i$ at time–point $t_i$ and has a minimal distance to the values $y(t_i)^-$ of the solution of the integration. Here $\bar{r}_i$ are the least squares conditions and constraints of the original problem, that are evaluated at $t_i$. If a condition is evaluated at several time–points, we include this condition only at the last time–point, because then we are able to evaluate the complete condition using the evaluation of parts at previous time–points. Repeating this for every time–point for which any condition is stated, we result in a discontinuous trajectory, which fulfils all conditions stated in the optimization problem, and as such is near the solution trajectory. The states of this trajectory at the multiple shooting nodes are used to provide initial values.

This approach is a generalization of the method which is used if we use the measurement information by hand, because we are no longer restricted to choosing the multiple shooting grid as a sub–grid of the measurement grid. Furthermore, this method is done *automatically* using the information and data–structure already included in the original optimization problem.

In the case of many measurement–points, it may be better not to project to the surface defined by $f$ but to use the weighted average of $y(t_i)^+$ and $y(t_i)^-$. This would eliminate a part of the statistical fluctuations.

For further readings and numerical examples we refer to [Jac01].

## 3.3 Discontinuities

Frequently the underlying differential equation has a discontinuous right–hand side $f$ or parameters $q$ which change discontinuously[1]

$$\dot{y} = \left\{ \begin{array}{lcl} f_1(t, y, q_1) & ; & s(t, y, q_3) \geq 0 \\ f_2(t, y, q_2) & ; & s(t, y, q_3) < 0 \end{array} \right\} , \tag{3.9}$$

or the states $y$ jump

$$y(t^+) = y(t^-) + j(t, x(t^-), q) \tag{3.10}$$

at the root of some *switching function* $s$.

In the case of mechanical models, the first happens if, for example, a force is defined by piecewise constant functions. A jump occurs if any body collides with another and this impact is modelled in an idealized discontinuous form.

The root $t_s$ of a switching function $s$ is defined explicitly by

$$s(t, y, q) = t - t_s = 0 \tag{3.11}$$

or implicitly by

$$s(t_s, y(t_s), q) = 0 \quad . \tag{3.12}$$

The root of such a switching function can be efficiently found based on the internal interpolated representation of the trajectory computed by the integration scheme

[1]here shown as an ODE for presentation reasons

[Win01]. If such a root is found, a modification of the differential equation or the states is possible. In the optimization context we additionally have to consider the sensitivity information.

### 3.3.1 Update Formula for Sensitivities

At the time-point of the root of a switching function, an update of the sensitivity matrix is done in the following way. This update is derived in [KE85], [Boc87]

$$\begin{aligned} W(t_{j+1}, t_j) &= W(t_{j+1}, t_s)^+ U W(t_s, t_j)^- \\ W_q(t_{j+1}, t_j) &= W(t_{j+1}, t_s)^+ \left(U W_q(t_s, t_j)^- + U_q\right) + W_q(t_{j+1}, t_s)^+, \end{aligned} \tag{3.13}$$

where

$$\begin{aligned} U &:= (f^+ - f^- - j_t - j_y f^-) \tfrac{s_y}{\dot{s}} + \mathcal{I} + j_y \\ U_q &:= (f^+ - f^- - j_t - j_y f^-) \tfrac{s_q}{\dot{s}} + j_q, \end{aligned} \tag{3.14}$$

$W(t, t_0) := \frac{\partial x(t)}{\partial x(t_0)}$, $W_q(t, t_0) := \frac{\partial x(t)}{\partial q}$ and $f^+, f^-$ are the right–hand side of the differential equation defined w.r.t. to the two regions separated by the switching function.

A formula adapted to mechanical DAE is given e.g. in [Sch99].

### 3.3.2 Differentiability Considerations for the Parameter Estimation Problem

The next question arising is: If we have a non–smooth trajectory, do we have to use an algorithm capable of treating discontinuous optimization problems?

To be more exact the question is, if a non–differential underlying differential equation leads to a non–differential parameter estimation problem, i.e. a problem where the differentiability conditions are violated.

If the *sequence* and the *number* of the roots of the switching functions do *not* change in an interval $[t_b, t_e]$, the states $y(t_e)$ at $t_e$ smoothly depend on initial states $y(t_b)$ and parameters. In parameter estimation context, this condition has to be fulfilled for every interval $[t_i, t_j]$, where $t_i$ and $t_j$ are time–points at which some condition, be it least squares equation or constraint, or a multiple shooting node is defined. If this condition is met, the assumptions of the optimization algorithms are fulfilled, and we can use the methods shown previously [Boc77].

Because we are able to give good initial values for the multiple shooting nodes based on measurement information, this is the case in a broad variety of practical applications. Therefore, we explicitly exclude the treatment of non–differential optimization problems in this work.

## 3.4 Statistical Analysis of the Solution

In addition to the solution of the parameter estimation problem, a statistical interpretation of the solution is necessary. We need an assessment of the reliability of the solution i.e. in the form of confidence intervals and variance-covariance matrices. In this section, we outline how this information can be provided in the framework of Generalized Gauss-Newton methods [Boc87].

Let $r$ contain all conditions, $J$ the Jacobian of the complete system (2.31) in the solution point $x^*$ and $J^+$ the respective generalized inverse. Under the assumption about the measurement error made in section (1.8), the solution $\Delta x = -J^+ r$ is also a random variable. An approximation to the corresponding variance-covariance matrix is given by

$$C(x) := \mathcal{E}(\Delta x \Delta x^T) = \mathcal{E}(J^+ \epsilon\epsilon^T J^{+^T}) = J^+ \mathcal{E}(\epsilon\epsilon^T) J^{+^T} = \beta^2 J^+ \Lambda J^{+^T} \quad , \tag{3.15}$$

where

$$\mathcal{E}(\epsilon\epsilon^T) = \beta^2 \begin{pmatrix} \mathcal{I}_{n_1} & 0 \\ 0 & 0 \end{pmatrix} = \beta^2 \Lambda \tag{3.16}$$

and $\epsilon$ is the measurement error, $\beta^2$ a common factor, and $\mathcal{I}_{n_1}$ a unity matrix corresponding to the $n_1$ leastsquares conditions.

This information is computed easily, due to the fact that a decomposition of $J$ at the solution is available at the end of an optimization process.

The variance–covariance matrix $C$ can be evaluated only for the diagonal elements and only for parameters $q$ or additionally the initial states $s_0$ or the complete dynamic variables. The latter supplies a statistical analysis of the solution trajectory in a discretized form [Sch88]. This is of particular interest if the quality of the solution of states has to be assessed and no direct data are available.

In case the common factor $\beta^2$ should be unknown, it can be obtained by

$$\beta^2 := \frac{||r_1(x)||_2^2}{l_2} \quad , \tag{3.17}$$

where $l_2 := n_1 - l_1$ and $l_1 := (n_y + n_p) - (n_c)$. Approximations to the corresponding joint confidence intervals $\theta_i$ can be computed by

$$\theta_i = b\big(C_{ii} l_1 F_{l_1,l_2;1-\alpha}\big)^{\frac{1}{2}} \qquad i = 1, \ldots, n_p \quad , \tag{3.18}$$

where $F_{l_1,l_2;1-\alpha}$ denotes the $(1-\alpha)$-quantile of the $F_{l_1,l_2}$-distribution.

# Part II

# Mechanical Systems

Several fields of biology, medicine are physics are interested in the movement of physical objects. If only the coarse motion is of interest neglecting e.g. deformations, it is useful and adequate to use a rigid body formulation. There are several approaches which are typically distinguished by the choice of coordinates and by the way the equations of motion, based on this choice, are built.

One possibility is to use Natural Coordinates as introduced by García de Jalón and Bayo . They first define the position (and velocity) of each body and then restrict the relative motion of two bodies by introducing additional constraints. By using the large number of twelve position–variables per body, they are able to restrict themselves to very simple constraint equations of typically maximal second order. Due to this, the system of differential equation is of high dimension, but each entry is very easy to compute. Until then do we follow the approach of García de Jalón and Bayo, but in the following the systems of equations are treated differently. García de Jalón and Bayo, continue by using the fact that one can share variables between different bodies to reduce the dimension of the system. A revolute joint between two bodies fixes one point and one axis, such that this point and axis can be shared between the two bodies. We do *not* apply point–sharing to preserve the structure of the system matrix, which can be exploited. Furthermore, we apply a different linear algebra method to the resulting linear system. Our method is able to evaluate the right–hand side of the differential equation with linear complexity in contrast to the quadratic complexity of the classical approach. To apply this method, one has to guarantee a set of non–redundant constraints. As this is not *automatically* given with the classical approach, several constraints are formulated differently. This is done in such a way that the basic property of the original formulation is regained, i.e. only constraint equations of maximal second order are needed.

In our formulation, the equation of motion leads to the so–called Full Descriptor Form, where the DAE of index 3 is reduced to index 1. The main computational effort is the decomposition and solution of the linear system of the acceleration constraint. Because we trade the reduction of the state dimension for the structure of the system matrix, special attention has been paid to the efficiency of the decomposition and the solution. First, because of the block–structure arising from our approach of not using point–sharing, we were able to provide it in linear complexity. Second, the constraint matrix and mass matrix of each block has a special sub–structure, too. This additional structure, which is due to the simplicity of modelling using Natural Coordinates, is used to further improve the efficiency of the algorithm. As a result, we are able to provide a method which is competitive to other methods, especially for a large number of bodies, while providing a set of equations which are very simple. This simplicity is very helpful if one wants to study more complex elements or provide derivative generation, which both is the case in this work of parameter estimation of bio–mechanical models.

After this first chapter, which presents the core modelling task, the second chapter leads us to several more advanced topics resulting from the use of Natural Coordinates in our formulation. We introduce an alternative formulation of Natural

Coordinates, which *eliminates* some constraints of the rigid body to reach into the field of flexible bodies.

The linear algebra method is applicable to a set of non–redundant constraints. Although we eliminate one possible source of redundancy, as discussed before, it is possible that no global set of non–redundant constraints exists. Therefore, we have developed constraint partitioning, which handles constraints in a similar way as coordinate partitioning handles variables. The main difference is that our approach does not lead to a restart of the integration method and, thus, improves the performance of the computation. The neighboring topic of singular kinematic positions is discussed briefly; nature or engineers are typically intelligent enough not to model such systems. This might not be the case for ”simple” test–examples or idealized models.

At last, we discuss discontinuous models again – this time from the perspective of the modelling task, and in a simulation context.

As a result, we are able to model systems of arbitrary topology in a way that combines simplicity of the formulation and numerical efficiency. Therefore, we are more than competitive with other methods.

# Chapter 4

# Modelling in Natural Coordinates

Rigid body formulations provide a physical description of a special kind of multi–body systems with several non–deformable bodies. They can rotate or translate as a whole, but do not change their shape. These bodies are connected by joints, i.e. elements that restrict several possible relative motions between two bodies or one body and the reference system. Therefore, the number of kinematic degrees of freedom – of originally six times the number of bodies – is reduced. Furthermore, there are forces interconnecting the bodies or applied to the bodies from the environment. It is typically assumed that these forces or torques can be formulated in such a way that they are applied to one point respectively around an axis of the body [Sch86].

At first glance, this seems to be a major idealization of reality. However, for a vast number of applications it is a suitable approximation. Therefore, we focus on this approach throughout this chapter, which deals with the basics that can be applied to technical mechanics. The last chapter of this part extends the applicability to the field of bio–mechanics. We will see, that even in the case of human beings, which do not consist of rigid elements as a robot, this idealized approach can be applied.

We first look at the basic question of how to choose the coordinates. We choose a modified version of Natural Coordinates [GB94] and derive the necessary information to provide the evaluation of the right–hand side of the differential equation describing this model. We will see that the simplicity of the basic transformation originating from the way we choose the coordinates leads to simple equations, too. Furthermore, we get a special structure of the linear system we have to solve; which can be exploited to get an algorithm of linear complexity w.r.t. the number of bodies.

## 4.1 Coordinates in Multi–Body Systems

The fundamental issue of modelling multi–body systems is how to choose the coordinates. There are several approaches to this issue, and the question of an optimal choice is an area of ongoing research. Tree–structured mechanisms without closed kinematic loops can be modelled recursively traversing through the kinematic chains. This usually leads to a set of ordinary differential equations (ODEs) for the equations of motion.

Alternatively, we see the usage of a variety of *redundant coordinate systems* for modelling multi–body systems (MBS) on the basis of Lagrangian equations of the first kind. This leads to the well–known set of differential algebraic equations of index 3 for MBS. Index–reduction for numerical treatment of this DAE leads us to the *Full Descriptor Form* of index 1 (see equations 2.14, 2.15, 2.16):

$$\begin{aligned} \dot{e} &= f_e \\ \dot{p} &= v \\ \dot{v} &= a \\ \begin{pmatrix} M & G^T \\ G & 0 \end{pmatrix} \begin{pmatrix} a \\ \lambda \end{pmatrix} &= \begin{pmatrix} f \\ \gamma \end{pmatrix} \\ g_{pos}(p) &= 0 \\ g_{vel}(p,v) &= 0 \end{aligned} \tag{4.1}$$

with position $p$, velocity $v = \frac{dp}{dt}$, acceleration $a = \frac{dv}{dt}$, external variables $e$, corresponding right–hand side of the differential equation $f_e$, mass matrix $M$, forces $f$, kinematic constraints $g_{pos}$, velocity constraints $g_{vel}$, acceleration–independent part of the acceleration constraints $\gamma$, Lagrange–multiplier $\lambda$ and constraint matrix $G = \frac{dg_{pos}}{dp}$.

A unifying idea for most modelling techniques is to establish a local coordinate system in each body of the MBS. The coordinate choice is usually the choice of an efficient parameterization of the affine transformation between the global (world) and each one of the local (body) coordinate systems. Reference point coordinates in 3D e.g. model the Cartesian coordinates of the origin of the local coordinate system together with three angles (Eulerian angles). While this leads to the most compact possible set of coordinates, they exhibit two singular position, which causes a variety of numerical problems. Since any parameterization of a free body in 3D with exactly 6 coordinates has at least two such singular points, it is only possible to overcome this drawback with redundant coordinate systems if one global map should be used.

Our modelling method of choice is based on *Natural Coordinates* as described by García de Jalón and Bayo in [GB94]. Although this method uses the 12 coordinates per body and, therefore, a model of very high dimension, we will show that the resulting equations are very simple and have a distinct structure. In contrast to García de Jalón and Bayo, who use this structure to reduce the dimension of the dimension of the variables, we exploit this structure. As a result, we are able to provide a evaluation of the right–hand side of the differential equation in $\mathcal{O}(n)$ w.r.t. the number of bodies, whereas García de Jalón and Bayo have a complexity of $\mathcal{O}(n^2)$ w.r.t. the number of bodies. To be able to do this, we heavily modified the original formulation.

## 4.2 Natural Coordinates

In most other methods, the affine transformation is specialized to the combination of a *translational* part of the origin and a *rotational* part. But the orientation matrix does not have to be an element of $\mathcal{SO}(3)$. A more general view that allows scaling and shearing can be used to adjust the coordinate system to the local topology of the model. Natural Coordinates parameterize the underlying affine transformation in a natural and more general way.

An affine transformation of a body–fixed point $r$, expressed in local coordinates, to its global representation $\bar{r}$ is given by

$$\bar{r} = o + Xr \tag{4.2}$$

where $o \,\epsilon\, \mathcal{R}^3$ is the translation of the origin of the local coordinate system and $X \,\epsilon\, \mathcal{R}^{3\times 3}$ is the corresponding orientation matrix.

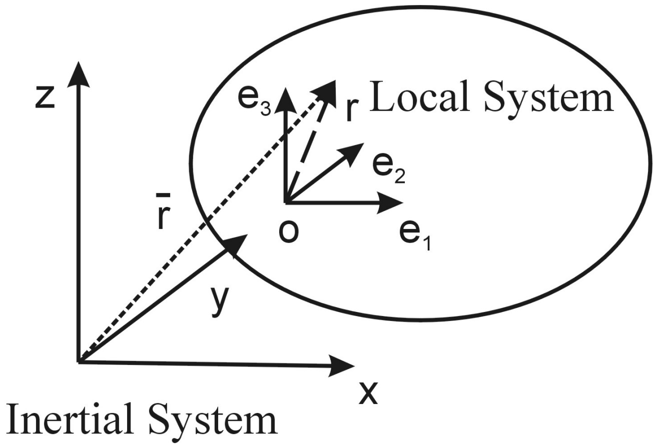


Figure 4.1: Affine Transformation

Natural Coordinates do *not* parameterize $X$, but uses *all* 9 entries of $X$ as dynamic variables. Because of this relation between the dynamic variables and the orientation matrix, every local point can be transformed into its global representation by a *linear* transformation of the dynamic variables.

$$\begin{aligned}
\bar{r} &= o + Xr = o + (e_1|e_2|e_3)r \\
&= 1o + r_1e_1 + r_2e_2 + r_3e_3 \\
&= \underbrace{\left(\begin{array}{ccc|ccc|ccc|ccc}
1 & & & r_1 & & & r_2 & & & r_3 & & \\
& 1 & & & r_1 & & & r_2 & & & r_3 & \\
& & 1 & & & r_1 & & & r_2 & & & r_3
\end{array}\right)}_{C_r}
\underbrace{\begin{pmatrix} o \\ e_1 \\ e_2 \\ e_3 \end{pmatrix}}_{p} \\
&= C_r p
\end{aligned} \tag{4.3}$$

where $C_r \,\epsilon \mathcal{R}^{3\times 12}$ is defined by the local affine coefficients and $p \,\epsilon\, \mathcal{R}^{12}$ are the dynamic variables of each body. The equation holds true for the first and second derivative with the same matrix $C_r$ ($\dot{\bar{r}} = C_r\dot{p}, \ddot{\bar{r}} = C_r\ddot{p}$).

**Remark 3 (Translation Vectors)**
*Besides affine points, as shown above, the modelling process uses* translation *vectors. Setting the leading 3x3 block of $C_r$ to zero gives the correct transformation while preserving the structure.*

## 4.3 Deriving the Mass Matrix

In the following section, we derive the mass matrix in a general form. To allow for a comparison to the better known moment of inertia w.r.t. Eulerian angles, we first derive this form for one free body[LL75]. We write the global position $\bar{r}$ of a point w.r.t. the reference frame as a superposition of the position of the origin of the body $o$, and $\tilde{r}$ as the position of one point w.r.t. the local frame ($\tilde{r} = Xr$)

$$\bar{r} = o + \tilde{r}. \tag{4.4}$$

In this formulation, the origin is equivalent to the center of mass.

We are able to write the differential of $\bar{r}$ in the following form

$$d\bar{r} = do + d\varphi \times \tilde{r}, \tag{4.5}$$

where $d\varphi$ is an infinitesimal rotation. Using the definition of the differential, we get the velocities

$$v = v_o + \Omega \times \tilde{r}, \tag{4.6}$$

where $v_o$ is the velocity of the center of mass and $\Omega$ the angular velocity of the body.

The kinetic energy $T$ is typically written as

$$T = \int_V \frac{\varrho v^T v}{2} dV \tag{4.7}$$

with constant or variable density $\varrho$, and integrated over the whole body. Substituting $v$ with equation (4.6), we get

$$T = \int_V \frac{\varrho}{2}(v_o + \Omega \times \tilde{r})^T(v_o + \Omega \times \tilde{r})dV. \tag{4.8}$$

Using the well–known transformations we get the final form of the kinetic energy

$$T = \frac{m}{2}v_o^2 + \frac{1}{2}\sum_{i,k}\Theta_{ik}\Omega_i\Omega_k, \tag{4.9}$$

with mass $m$ and the classical moments of inertia w.r.t. Eulerian angles $\Theta$ with components

$$\Theta_{ik} = \int_V \varrho(\tilde{r}_l^2 \delta_{ik} - \tilde{r}_i \tilde{r}_k) dV. \tag{4.10}$$

Therefore, the complexity of the form of $\Theta$ is a direct result of the definition of $v$ (4.6).

We will see that the very simple form of the corresponding equation in Natural Coordinates (first derivative of (4.3)) results in a less complex form of the moment of inertia.

To derive the moments of inertia w.r.t. Natural Coordinates, we start with the general form of the kinetic energy (4.7), too. Following the previous transformations, we use the first derivative of transformation (4.3). With respect to the dynamic variables $p$, we write the kinetic energy

$$T = \frac{1}{2} \int_V \varrho\, \dot{p}^T C_r^T C_r \dot{p}\, dV. \tag{4.11}$$

As the integral is independent of the dynamic variables which are derived from the transformation of the whole *rigid* body, we write

$$T = \frac{1}{2}\, \dot{p}^T \left( \int_V \varrho\, C_r^T C_r\, dV \right) \dot{p} =: \frac{1}{2}\, \dot{p}^T M \dot{p} \tag{4.12}$$

and, thereby, define a mass matrix $M \in \mathcal{R}^{12\times 12}$.

We get a bi–linear form with respect to the velocity, which means that we can establish the mass matrix for Natural Coordinates. Because of the structure of Natural Coordinates, the mass matrix $M$ does in general not separate into a translational and a rotational part. This property is reproduced only if the origin corresponds to the center of mass of the body, which is a prerequisite when modelled using Eulerian angles. We see that the mass matrix in Natural Coordinates is of simpler mathematical form than the Eulerian counterpart, which is a direct result of the simplicity of the underlying transformation. Furthermore, we stress that this formulation does *not* introduce additional terms originating from coriolis- or centrifugal forces as long as the reference system is an inertial system.

The mass matrix in Natural Coordinates is constant (as in many formalisms with a body–fixed coordinate system) and sparse – a fact that follows directly from the structure of the transformation matrix $C_r$.

As the classical moment of inertia is widely available, we give a transformation to Natural Coordinates if the origin of the coordinate system is the center of mass

$$M = \begin{pmatrix} m & \\ & \frac{\operatorname{tr}\Theta}{2}\mathcal{I} - \Theta \end{pmatrix} \otimes \mathcal{I}_3, \tag{4.13}$$

where $\operatorname{tr}\Theta$ is the trace of $\Theta$ and $\otimes\mathcal{I}_3$ denotes the tensorial product with the three–dimensional unity matrix, which results in the 12–dimensional quadratic mass matrix of Natural Coordinates. This is easily seen by comparing the formula w.r.t. Eulerian angles and Natural Coordinates. This formula is valid if the origin of the local coordinate system coincides with the center of mass of the body. Otherwise, one has to transform to this form.

## 4.4 Modelling Rigid Bodies

In Natural Coordinates the orientation matrix $X$ may not be an element of $\mathcal{SO}(3)$. However, when modelling *rigid bodies*, there is a set of algebraic relations between the natural coordinates. These so–called rigid body constraints restrict the coordinates to the suitable 6–dimensional subspace. The relations keep the length of the axis and the angles between them fixed

$$\begin{array}{rclcll}
g_{p,i}^{RC} & = & \frac{1}{2}e_i^T e_i & - & \frac{1}{2}l_i^2 & (i = \{1,2,3\}) \\
g_{p,4}^{RC} & = & e_1^T e_2 & - & a_1 & \\
g_{p,5}^{RC} & = & e_1^T e_3 & - & a_2 & \\
g_{p,6}^{RC} & = & e_2^T e_3 & - & a_3 & .
\end{array}$$

All these equations are polynomial of *second order*, so that they and their derivatives, which are used in the full descriptor form, are very cheap to compute. The resulting constraint matrix is *linear* with respect to the dynamic variables, and it has a special structure

$$G^{RC}(p) = \frac{\partial g_{pos}^{RC}}{\partial p}(p) = \begin{array}{c} \overbrace{\begin{array}{cccc} o & e_1 & e_2 & e_3 \end{array}} \\ \left(\begin{array}{cccc}
 & e_1^T & & \\
 & & e_2^T & \\
 & & & e_3^T \\
 & e_2^T & e_1^T & \\
 & e_3^T & & e_1^T \\
 & & e_3^T & e_2^T
\end{array}\right) \end{array} . \tag{4.14}$$

The computation of $\gamma$ can be rather time–consuming in other formulations because it corresponds to the second derivative of a possibly highly nonlinear kinematic constraint. As numerical experience show for other approaches, the computation of $\gamma$ consumed up to 80% of the time needed to evaluate the model. In our formulation, the computation of $\gamma$ is as cheap as the evaluation of the kinematic constraints,

$$\begin{array}{rcll}
\gamma_i^{RC} & = & -\dot{e}_i{}^T \dot{e}_i & (i = \{1,2,3\}) \\
\gamma_i^{RC} & = & -2\dot{e}_j{}^T \dot{e}_k & (i = \{4,5,6\}).
\end{array} \tag{4.15}$$

## 4.5 Defining Joints by Constraints

Introducing joints even more restricts the degrees of freedom of the complete system. This way, modelling joints can be achieved by imposing additional joint constraints. The two key concepts for modelling standard joints are, on the one hand, restricting translational movement in one direction and, on the other hand, rotational movement around an axis. Both are realized by keeping two vectors parallel.

A formulation introduced by García de Jalón and Bayo uses the vector product, leading to a set of three *linearly dependent* equations. But for numerical treatment, we need a constraint matrix with full row rank. This problem can be avoided by an adapted formulation using two orthogonality relations of the following form

$$a \| b \iff \exists\, h_1 \perp h_2, a \perp h_1, a \perp h_2 :\ b \perp h_1 \ \wedge\ b \perp h_2, \tag{4.16}$$

where $a, b, h_1$ and $h_2$ are the global representation, $a, h_1, h_2$ are fixed w.r.t. the first body whereas $b$ is fixed w.r.t. the second body and $h_1, h_2$ are unity vectors. The two orthogonality relations are realized using scalar products. Due to the linearity of the basic transformation (4.3) are the resulting kinematic constraints equations of second order w.r.t. the dynamic variables.

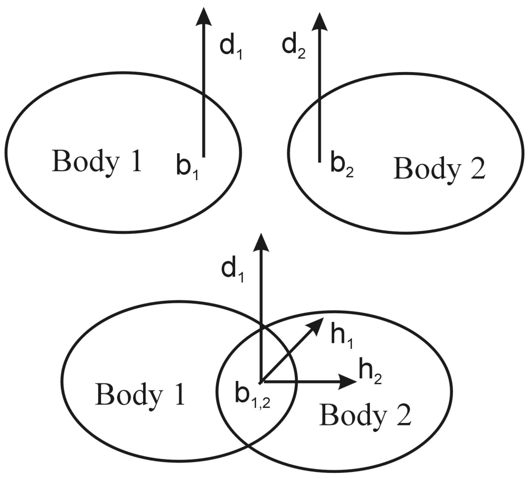


Figure 4.2: Assembly of a Revolute Joint

**Example (Deriving the constraints for a revolute joint)** We have to fix two bodies at a common point with local representations $b_1, b_2$, which is a special form of disallowing translational movement. This is modelled as a direct comparison of the two points and leads to three equations of first order. With transformation (4.3) we get

$$g^J_{p;1,2,3}(p_1, p_2) = [b_1(p_1) - b_2(p_2)]_{1,2,3} := [C_{b1}p_1 - C_{b2}p_2]_{1,2,3} = 0. \tag{4.17}$$

Furthermore, we have to forbid rotations other than around the rotation axis (local coordinates $d_1, d_2$)

$$g^J_{p;4}(p_1, p_2) = d_1(p_1)^T h_1(p_2) := (C_{d1}p_1)^T C_{h1}p_2 = 0 \tag{4.18}$$

$$g^J_{p;5}(p_1, p_2) = d_1(p_1)^T h_2(p_2) := (C_{d1}p_1)^T C_{h2}p_2 = 0. \tag{4.19}$$

where $h_1$ and $h_2$ are perpendicular to $d_2$.

For cylindrical or prismatic joints, we restrict translational movement in specific directions. This is accomplished by defining a translational vector by the difference vector of two points. Therefore, restricting translational motion leads again to formulating constraints which ensure parallelism of vector in their global representation; the same concept we used for restricting rotation. More complex joints or constraints are modelled using these basic formulations. For further reading concerning the definition of other joints we refer to [GB94] and [Kra97].

## 4.6 Forces

We now want to derive the formula for point-forces or volume-forces that are expressed in the first form, e.g. the gravitational force. We start with the principle of virtual work

$$\delta W = \delta r_P^T f_P \tag{4.20}$$

for a point force $f_P \in \mathcal{R}^3$ at point $P$, with global position $r_P \in \mathcal{R}^3$ and corresponding transformation matrix $C_P \in \mathcal{R}^{3\times 12}$. Using transformation (4.3), we get

$$\delta W = \delta p^T C_P^T f_P \tag{4.21}$$

and, by comparison, we get the force $\hat{f}$ w.r.t. Natural Coordinates

$$\hat{f} = C_P^T f_P. \tag{4.22}$$

Further information again can be found in [GB94] and [Kra97].

## 4.7 Solving the Linear System

We have introduced Natural Coordinates for mechanical systems resulting in elegant and very simple equations. But this is achieved by introducing a very high number of dynamic variables, which typically leads to increased computational costs, and, therefore, would be a major drawback.

In this chapter we show that Natural Coordinates in our approach do not only result in equations of low order, but in equations of a very special structure, too. Using this structure on two qualitatively different levels leads to an efficient algorithm of linear complexity w.r.t. the number of bodies in contrast to an algorithm by Gacía de Jalón and Bayo, which has quadratic complexity. Due to this, we get better performance for larger systems, retaining the advantages discussed in the previous chapter, and paving the way for the treatment of complex problems, as shown in the last chapter of this part.

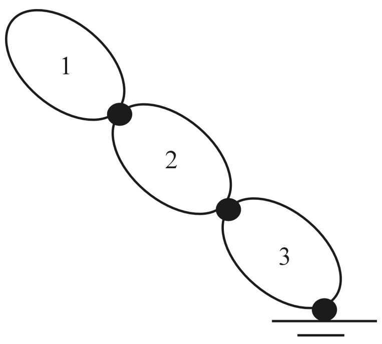
### 4.7.1 Block–Sparse Structure

The local topology of bodies and joints leads to a block–sparse structure of the linear system (2.15). In the original formulation of Gacía de Jalón and Bayo, *point–sharing*, that is sharing of variables corresponding to points or axes defining a joint between bodies, is used in order to shrink the system. They introduce a decomposition algorithm with quadratic complexity, which is fast for small models.

Instead of reducing the system via point sharing, our approach keeps the full structured matrix and *exploits* the structure based on a block–oriented Rational Cholesky–decomposition

$$A = L^T D L, \tag{4.23}$$

where $A$ is the original matrix, $L$ a lower left block–tridiagonal matrix and $D$ a block diagonal matrix. We study this at the three principle types of topology: linear chains, tree–structured systems and system with closed kinematic loops.



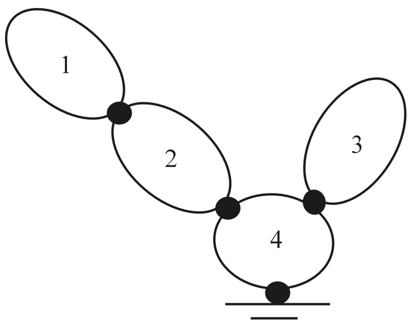
$$\begin{pmatrix} 1 & | & & & & \\ - & + & - & & & \\ & | & 2 & | & & \\ & & - & + & - & \\ & & & | & 3 & | \\ & & & & - & + \end{pmatrix}$$

Figure 4.3: Linear Chain

For systems consisting of *linear chains* (see figure 4.3), there exists a block–ordering that results in a block–banded matrix of the linear system. The Augmented system of the mass matrix and the rigid body constraints corresponding to each body are placed on the main diagonal, and the constraint matrix block corresponding to joints are placed on the first sub–diagonal. In the following figures, each block corresponding to a body is denoted by a number, while parts of the constraint matrix corresponding to the joints are marked by "$-$" and "$|$". As a result of the decomposition, the fill–in "$+$" is generated. The sparsity pattern of the figures show a combination of the original matrix and the result of the decomposition

$L$. In the original matrix there is obviously no fill–in, whereas $L$ is a lower left tridiagonal block–matrix.

This symmetric matrix is decomposed in linear time w.r.t. the number of bodies. In this case fill–in is not avoidable and corresponds to the Schur–complement in the non–blocked case.



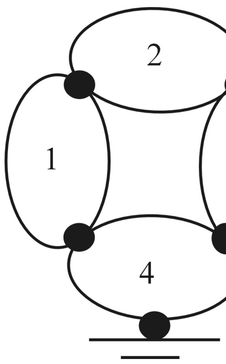
$$\begin{pmatrix}
1 & | & & & & & & \\
- & + & - & & & & & \\
 & | & 2 & & | & & & \\
 & & & 3 & & | & & \\
 & & - & & + & & - & \\
 & & & - & & + & - & \\
 & & & & | & | & 4 & | \\
 & & & & & & - & +
\end{pmatrix}$$

Figure 4.4: Tree–Structured System

If the model–topology is a *tree–structured* system (see figure 4.4), additional off–diagonal elements are introduced, but if a suitable ordering is used, these do not lead to an additional fill–in because of the block–pattern of the remaining matrix. The decomposition of this type of matrix can also be done in linear time.

If there are joints which lead to *closed kinematic loops* (see figure 4.5) off–diagonal blocks are introduced. These propagate fill–in ”$\circ$” in the magnitude of the size of the loop.

The behavior of more complex systems is qualitatively the same as shown in the simple examples. The linear complexity w.r.t. the number of bodies is regained if the number of closed kinematic loops is not too high. In the limit of a system where every body is connected with every other body, the complexity of the decomposition increases, but this is not the typical case in (bio-) mechanical systems. A complex example with many closed kinematic loops and a small overhead is the hexapod [WK00], also discussed in section (5.1.2). More examples can be found in [Kra97] or [KWB00].



$$\left(\begin{array}{ccccccccc}
1 & | & & & & & | & & \\
- & + & - & & & & \circ & & \\
 & | & 2 & | & & & \circ & & \\
 & & - & + & - & & \circ & & \\
 & & & | & 3 & | & \circ & & \\
 & & & & - & + & \circ & - & \\
- & \circ & \circ & \circ & \circ & \circ & + & - & \\
 & & & & & | & | & 4 & | \\
 & & & & & & & - & +
\end{array}\right)$$

Figure 4.5: System with closed kinematic loop

### 4.7.2 Sub–Block Structure

The method discussed in the previous section uses the inherent topology of the model in order to drastically reduce the computational effort. In addition to this, we recognize a specific substructure of each block, which can be exploited.

The matrices on the main diagonal correspond to the mass matrix and the rigid body constraints, i.e. the full definition of one body,

$$\bar{M} = \begin{pmatrix} M & {G^{RC}}^T \\ G^{RC} & \end{pmatrix}. \tag{4.24}$$

The mass matrix $M$ has a very special structure – it is constant and very sparse – but this can not be exploited in the context of our block–oriented sparse solver. Before the decomposition of $\bar{M}$, this block is modified with full matrices that are part of the joint treated before. A typical modification looks like

$$\tilde{M}_i = \bar{M}_i + \sum_j \tilde{L}_{ij} \tilde{M}_j^{-1} \tilde{L}_{ij}^T, \tag{4.25}$$

where $\tilde{L}$ are blocks corresponding to joint constraints. As a result of this, the pattern is destroyed. But there is a less obvious structure, which is used and leads to a speed–up of the computation by a factor of two.

During their decomposition of the block–structured system, block–operations of the type $\bar{M}^{-1}\tilde{L}^T$, comprise most of the computational effort. We take a closer look at the rigid body constraint matrix of one body, which is defined w.r.t. coordinates in standard form of one point as the origin and three additional vectors ($x$, $y$, $z$ as in equation 4.3):

$$G^{RC} = \begin{pmatrix} 0 & e_1^T & 0 & 0 \\ 0 & e_2^T & e_1^T & 0 \\ 0 & e_3^T & 0 & e_1^T \\ 0 & 0 & e_2^T & 0 \\ 0 & 0 & e_3^T & e_2^T \\ 0 & 0 & 0 & e_3^T \end{pmatrix} \tag{4.26}$$

The structure of $G^{RC}$ is further improved by multiplication from the right–hand side with

$$R(p) = \begin{pmatrix} \tilde{R} & & & \\ & \tilde{R} & & \\ & & \tilde{R} & \\ & & & \tilde{R} \end{pmatrix}, \tag{4.27}$$

where

$$\tilde{R}(p) = \begin{pmatrix} e_1^T \\ e_2^T \\ e_3^T \end{pmatrix}^{-1}. \tag{4.28}$$

This transformation reduces $G^{RC}$ to a matrix with only 9 entries equal to one. If one chooses the underlying coordinate system to be orthonormal, the inversion needed to compute $R$ is only a transposition, and therefore is immediately at hand. In this case, $R$ corresponds to a rotation.

By applying the null–space method and explicitly using the very simple structure of the transformed and constant matrix $\left(G^{RC}R\right)$, we are able to drastically reduce the overall computational time for this type of operation, but, on the other hand, the decrease of computational effort here is partially negated by the fact that the joint blocks have to be transformed, too. Much of the computational effort needed is avoidable if the transformed constraint matrix is computed directly. If this is done in an efficient way, the previously mentioned speed–up of two is achieved for the general class of mechanical models.

**Remark 4 (Adapting Coordinates to Joints)**
*The Adaptation of the choice of coordinates to the joints connected to the corresponding body leads to an even simpler transformation matrix $C$ (4.3), where some entries are zero, which reduces the computational effort of the mass-matrix update (4.25). This simplification is less universal than the one mentioned above, and the code–optimization of today's compilers are impaired, so that it is not implemented.*

# Chapter 5

# Advanced Topics of Modelling Based on Natural Coordinates

Chapter 4 has illustrated the convenience of modelling mechanical systems using Natural Coordinates in our approach. Now we focus on several special and advanced topics:

- We introduce an *alternative formulation* of Natural Coordinates to adapt to special kinematic structures or simple elastic bodies, which shows the high flexibility of our approach.

- We look at the problem of introducing redundant constraints due to automatic modelling. In addition we treat the situation that no global set of non–redundant constraints may describe the physical model correctly. We developed *constraint partitioning* which is able to efficiently treat this problem by selecting a non–redundant subset of the constraints. Our approach is qualitatively more efficient than the well–known coordinate partitioning, because we do *not* need to restart the integrator.

- We study physical models with *kinematic singularities*, which exhibit a varying number of degrees of freedom. This behavior is qualitatively different from the previous topic because it is a property of the physical model and not only its mathematical description.

## 5.1 Alternative Formulations

In the previous chapter, we have introduced Natural Coordinates in a specific form. Besides this standard formulation, many variations are possible and contribute to the flexibility of describing physical models in Natural Coordinates. We show some promising modifications of the basic definition.

### 5.1.1 Coordinate System Definition

The basic definition of a coordinate system of an affine vector space in 3D uses one point for the origin ($o$) and three translational vectors ($e_1, e_2, e_3$) that form the basis of the underlying vector space. Therefore, we are able to give the transformation of a locally defined point to global space as in equation (4.3):

$$\begin{aligned} \bar{r} &= o + r_1 e_1 + r_2 e_2 + r_3 e_3 \\ &= C_r p, \end{aligned}$$

where $p^T = \left(o^T e_1^T e_2^T e_3^T\right)$ and

$$C_r = \begin{pmatrix} 1 & & & r_1 & & & r_2 & & & r_3 & & \\ & 1 & & & r_1 & & & r_2 & & & r_3 & \\ & & 1 & & & r_1 & & & r_2 & & & r_3 \end{pmatrix}.$$

A translational vector is described by a modified matrix $\acute{C}_r$, where the leading $3x3$-unity matrix is replaced by a zero matrix.

An alternative formulation of the coordinate system is given when we define one or more of the axes $e_i$ by the difference of two points. This is useful if, for example, the body is connected to its neighbor via two spherical joints. Each joint is defined by a constraint using only one point. So if two points are available at the correct local position, the definition of the constraint is much simpler; two sets of dynamic variables, in contrast to two linear combinations of dynamic variables, have to be equal.

Let us use two points $a, b$ and two vectors $u, v$ instead of one point $o$ and three axes $e_1, e_2, e_3$. Using this definition, the corresponding transformation is

$$\begin{aligned} \bar{r} &= a + r_x(b-a) + r_y u + r_z v \\ &= (1-r_x)a + r_x b + r_y u + r_z v \\ &= \tilde{C}_r p, \end{aligned} \tag{5.1}$$

with

$$\tilde{C}_r = \begin{pmatrix} 1-r_x & & & r_x & & & r_y & & & r_z & & \\ & 1-r_x & & & r_x & & & r_y & & & r_z & \\ & & 1-r_x & & & r_x & & & r_y & & & r_z \end{pmatrix}. \tag{5.2}$$

Based on these transformations, we are able to derive (modified) mass matrices as well as constraints or joints.

An interesting type of coordinate definition uses four affine points and no translational vectors to describe the position and orientation of a body.

### 5.1.2 Modelling Non-Rigid Bodies

Another possible modification can be seen if we look at the physical interpretation of the so-called rigid body constraints. They are introduced in order to guarantee a fixed local coordinate system that implies a rigid body.

Vice versa, we introduce an additional degree of freedom to the body, if one or more of these rigid body constraints, like one that fixes a length, is omitted. In this case, we model a deformable body and enter the domain of simple elastic bodies (with low discretization).

We described one example for this type of modelling in [WK00], where a hexapod machine tool was modelled, consisting of a platform that can be moved in all six degrees of freedom by six actuators. A classical way of modelling uses thirteen bodies: one body for the platform and two rigid bodies per actuator. The alternative formulation uses only one body per actuator, which has no fixed length. The corresponding constraint is replaced by an actuating force element. This does not only reduces the number of bodies but also the computational effort per body.

Furthermore, a simple length-dependent moment of inertia is automatically included in this formulation because of the definition of the mass matrix in Natural Coordinates, which includes a scaling according to the length of the defining coordinate axes.

## 5.2 Constraint Partitioning

In modern interactive user interfaces, it is suitable to hide internal structures. If possible, an application–oriented user should not need to "know" the underlying mathematics. If one automatically builds up a model that has closed kinematic loops, it might happen that one of the prerequisites of the mathematical equations is violated: *the constraints must not be redundant*. A violation leads to a singular constraint matrix and therefore to a singular system matrix. An automatic treatment of such a situation drastically improves the usability of the modelling approach.

The first question is: How can constraints be redundant? Let us look at a closed chain of four bodies connected to each other with revolute joints, such that they are restricted to a two–dimensional plane. If one looks at the last joint which closes the revolute joint, the possible movement of the bodies is already restricted to a plane. As a result of this, the restriction of the rotational freedom, imposed by the joint, is not necessary. The corresponding constraints are redundant. In this simple example, this fact is obvious, and the problem may be interactively fixed by the user. Since the model can get arbitrarily complex, an *automatic* treatment is desirable.

As mentioned before, redundant constraints lead to a rank–deficient system matrix. The other way round, a rank analysis can be used to detect redundant constraints. If they are detected, this enables us to eliminate them. The result is a *physically equivalent mathematical formulation* without singularities. As we partition the constraints in a set of independent and redundant constraints, this method is called *constraint partitioning*. It is similar to *coordinate partitioning*, which selects a number of non–redundant variables. We want to stress that a globally valid set of non–redundant constraints need not exist. Therefore, we have to monitor and adjust the set during the simulation. As we have to check at every integration–step, a method with only a small additional effort is of upmost importance.

It is convenient to define the rigid body constraints to be always in the set of used constraints, because they would also be present in the non–redundant open loop case. Furthermore, we restrict ourselves to only monitoring each diagonal joint block independently, using the same argument.

We use the Cholesky decomposition for the joint–block. If an (almost) zero diagonal element appears during the decomposition, the corresponding constraint is dynamically eliminated. We compute this information, because we need to monitor a change in the set of redundant constraints. If a change in the active set of non–redundant constraints is detected, we do not need to re–decompose, but we can keep the previously computed part of the decomposition. In this case, the computational effort is equivalent to a decomposition of the full system using all constraints. In the typical case of relatively few redundant constraints, we can neglect the additional effort.

It is important to mention that a restart of the integration scheme is *not* needed, because a change in the non–redundant set of constraints only leads to a discontinuous Lagrange multiplier. The differential part of the DAE does not depend explicitly on these Lagrange multipliers. This drastically improves the performance of the method in comparison to coordinate partitioning, where every change leads to a very costly restart of the integration scheme. The method of detecting rank–deficiency is only valid if one stays near the consistent manifold, which is guaranteed by all state–of–the–art mechanical integration packages.

As an example for the performance of this technique, we present the six–bar–mechanism in section 12.3.

## 5.3 Singular Kinematic Positions

As mentioned in the previous section, it is not always possible to give a global non–redundant set of constraints. We considered the fact that constraints may be redundant at some point of the manifold, but we kept the number of non–redundant constraints constant up to now. There are several examples where at one point of the manifold its dimension changes; the physical model exhibits singular kinematic points.

Singularities of lowest order, and therefore the most common ones, can be categorized as: One isolated point with an additional degree of freedom and a crossing of two manifolds, which results in an additional degree of freedom.

A simple two–dimensional example of the first case is defined by

$$g_1(x_1, x_2) = \begin{pmatrix} v_+^T v_+ - 1 \\ v_-^T v_- - 1 \end{pmatrix} = 0, \tag{5.3}$$

where

$$v_+ := \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - \begin{pmatrix} +1 \\ 0 \end{pmatrix}$$

and

$$v_- := \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - \begin{pmatrix} -1 \\ 0 \end{pmatrix}$$

The solution of the system is the origin.

Let us look at the constraint matrix

$$\frac{\partial g_1(x_1,x_2)}{\partial(x_1,x_2)} = \begin{pmatrix} 2(x_1-1) & 2x_2 \\ 2(x_1+1) & 2x_2 \end{pmatrix}. \tag{5.4}$$

It has full rank except for the origin, which is the solution. As a result of this, we encounter an isolated singular point at which the constraints are redundant, while they are non–redundant everywhere else. To avoid this difficulty it is necessary to reformulate the constraint. One possibility is

$$\bar{g}(x_1,x_2) = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = 0. \tag{5.5}$$

In our modelling package MBSNAT, we avoid this kind of singularity by simply not formulating constraints of this type where the distance to a point is fixed. The alternative presented here or, to be more exact, its generalization is used exclusively.

The second type can be shown in

$$g_2(x_1,x_2) = x_1 \cdot x_2 = 0 \tag{5.6}$$

with the constraint matrix

$$\frac{\partial g_2(x_1,x_2)}{\partial(x_1,x_2)} = \begin{pmatrix} x_2 & x_1 \end{pmatrix}. \tag{5.7}$$

This single equation defines the $x_1$–axis and the $x_2$–axis as the allowed manifold. The two parts of the manifold cross in the origin, where we have a singular point with two degrees of freedom and as such a rank–deficient constraint matrix. If the residuum is changed the manifold will change qualitatively, depending on the sign of the residuum

$$\tilde{g}_2(x_1,x_2) = x_1 \cdot x_2 = \pm\epsilon \quad ; \epsilon > 0. \tag{5.8}$$

This problem of changed kinematic will be treated in section 8.6.2. We now want to focus on the difficulties arising for the simulation of this kind of problem.

This case is more difficult than the first one, because no universal and simple reformulation is possible. This kind of singularity can be detected but not avoided by applying a rank analysis to the linear system as described in the previous section. A general treatment is possible based on a Gröbner basis and an algebraic reformulation as described by Stossmeister [Sto04].

# Chapter 6

# Biomechanical Extensions

The modelling techniques described so far can be easily applied to the field of technical mechanics. Bodies are typically well described by rigid bodies. Joints as mentioned before originate from this field, and one typically tries ro build the real system to conform with the idealized description. Furthermore, we encounter actuators, springs and dampers which drive a specific degree of freedom. This characterization may be a bit too optimistic, because there are several deviations from this ideal behavior that have to be modelled in order to be precise, like e.g. vibrations in motors or flexibility of segments, but the methods described so far are a very good first order approximation. As a generalization, one is able to state: Technical systems are built based on a model that is described using the idealized elements of rigid body dynamics.

This changes in the field of bio–mechanical systems. Nature as an engineer is not based on some idealized physical description, which leads to a more complex realization: e.g. the human body. Only the bones of the human body are (more or less) rigid. This does not apply for other parts like muscles or fat. The muscular system as the driving force is very complex in terms of where and how it connects, and in terms of the internal dynamics. Moreover, the previously rigid joints are realized using tendons and do not have the simple structure of technical systems. Besides this complexity of the elements, additional difficulties arise because of the overall number of elements like bones, muscles, tendons and other organs. As a result of this, an exact treatment of bio–mechanical systems is only possible by using finite–element models. But the use of these methods would lead to a very high computational cost, which is typically not acceptable at the moment. Finite–elements are only usable for short time–spans or for a part of the complete system.

To solve this dilemma, one has to select the parts of the system one wants to model in more detail and simplify the rest. This leads to a reexamination of rigid body dynamics adapted to bio–mechanical systems. If a first approximation using only the elements described before is not accurate enough, which is true in most cases, one has to think about extensions that are not commonly used in technical mechanics but still can be described using the typical methods of rigid body dynamics like:

- Several joints of the human body are described using spherical joints or, like the knee joint, even more complex joints with moving rotational axes.
- When modelling intervertebral disks, i.e. tissue that provides a connection between vertebras of the spine, we have to include tensorial force–elements (see section 6.2.2).
- When thinking about the soft tissue of the human body, the usage of rigid bodies is a very bad approximation. Better results can be achieved using additional ”wobbling masses”. The soft tissue is modelled using an additional body which is connected to the bone (the original rigid body) via a general tensorial force (see section 6.2.3).
- In several occasions, it is useful to model muscles in more detail, and not to use an idealization by simple force–elements. Because muscles typically do not connect two neighboring segments and display a more complex force law, they will be treated separately in section 6.4.
- The contact of the human body with the environment, like the foot–ground contact, is very common and must be treated (see section 6.3).

Before all these extensions are described in the following sections, we first want to look at the question how to provide valid anthropometric data to describe a human.

## 6.1 Anthropometry

The anthropometric data needed to describe the human body consist of the kinematic definition (e.g. position and types of forces, length of segments) and the dynamic properties of the bodies (i.e. mass, center of mass and moment of inertia) as well as of the internal force–elements which represent, for example, tendons, intervertebral discs and other cartilage, and especially muscles. Even in the case of technical mechanics, these quantities are difficult to estimate. In the case of humans, this is even more difficult, because we do not have access to the construction plan of a human and detailed measurements are either very complex, of an invasive nature or both. Furthermore, there is a big variety in individual humans, which turns the generalization into an exercise of statistics. Nevertheless, there are databases available [NAS78], [Erg86] which provide a statistically valid set of anthropometric data based on correlated data – originating from such experiments – and based on user inputs like gender, mass, height etc, which are easy to provide.

Through surgery, x-ray examinations or magnetic resonance imaging, one is able to measure several internal data of humans, like structure of the skeleton, location of muscles, tendons and other tissues. Furthermore, one is able to provide data using minor invasive surgery on selected individuals to directly get information. Doing this for a big number of individuals results in the correlated anthropometric data we need. Further studies can be made using corpses. Although this may lead to further insight, some properties differ from living humans. The most problematic information is about dynamic data. Although it is possible to study muscles from animals individually and extrapolate to human living tissue, this includes several possible sources of error.

Measurements like the ones described above are done and compiled in a correlated database. In connection with that, we want to mention a thorough study of the NASA (NASA reference publication 1024) [NAS78] which used male personal of European origin in the 50's to 70's and a study of the Deutsche Institut für Normierung e.V. [Erg86] resulting in DIN–Norm 33402. Both studies have their drawbacks. The NASA study only used humans of European origin, and moreover, they used their own personal, so that the database corresponds well to better trained individuals. The German study focused on ergonomic values. Furthermore, the set of data is naturally not complete for all possible applications. As a result, it is necessary to call in additional medical information of literature and geometric approximations.

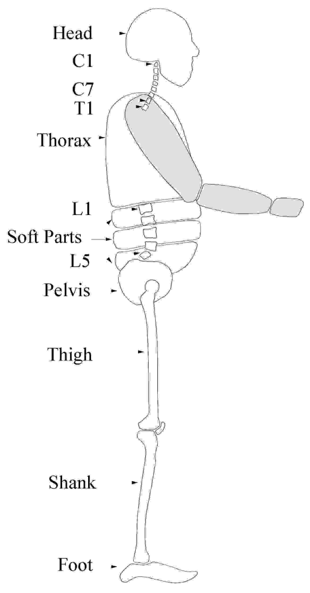


Figure 6.1: One possible Segmentation of a human body [Fri00]

Hahn [Hah93] and Günther [Gue97] provide the software package Calcman3d, which is able to generate anthropometric data for a good rigid body approximation based on values that are easy to estimate, like height, gender and mass. Calcman3d is extensible in the sense that additional data originating from further experiments can be included to increase the accuracy of the model and in the sense that it is possible to generate a model for different idealizations and situations. The resulting models are accurate enough to study general human behavior, like human walking, where one only needs a sensible set of data compatible with the assumptions.

One possibility to provide better anthropometric data for individual humans is to measure more information than just height, gender or mass. Several methods exists, that scan more information, e.g. the Body Scanner VITUS of tecmath AG. These systems typically provide very good static information, but lack the possibility to provide information about dynamic properties of human.

Our aim is to use a non–invasive and simple type of measurement setting to provide accurate static and dynamic information about one individual. We use motion capturing for measurements and a complex numerical method – nonlinear parameter estimation – to estimate the data, which is only implicitly defined by these measurements.

As the performance of the Generalized Gauß–Newton method we use to solve the parameter estimation problem heavily depends on good initial guesses, the estimates of the methods presented before are still of upmost importance.

## 6.2 Force–Elements

Typical force–elements of technical mechanics depend on the relative position or velocity of two points. They apply forces to one rotational or translational degree of freedom or to model external forces like gravitation. All these forces are relatively simple and easy to model using the techniques as shown in the first chapters of this part. Because of the complexity of biomechanical models, it is more likely to encounter more complex force–elements in this case.

### 6.2.1 Spherical Forces

In the field of bio–mechanics, several segments i.e. bodies are connected with joints which restrict a lower degree of freedom. This leads to the question of how to model force–elements that depend on all three rotational degrees of freedom, which can be parameterized using Eulerian or Cardan angles $(\alpha, \beta, \gamma)$.

Using this definition of variables, one can formulate a spherical force as

$$f = \begin{pmatrix} c_{11} & c_{21} & c_{31} \\ c_{21} & c_{22} & c_{32} \\ c_{31} & c_{32} & c_{33} \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} + \begin{pmatrix} d_{11} & d_{21} & d_{31} \\ d_{21} & d_{22} & d_{32} \\ d_{31} & d_{32} & d_{33} \end{pmatrix} \begin{pmatrix} \dot{\alpha} \\ \dot{\beta} \\ \dot{\gamma} \end{pmatrix} \quad . \tag{6.1}$$

To define the angles we need, one has to calculate the rotation between the two connected bodies w.r.t. each other and to an initial rotation. We first calculate the initial difference of orientation

$$D_0 = B(p_0)E(p_0)^T \quad , \tag{6.2}$$

where $B(p), E(p)$ are the orientation matrices of the first respectively second body.

Using this, we calculate the difference of the rotation at position $p$

$$D^p = B(p)^T D_0 E(p) = \left(B(p)^T B(p_0)\right)\left(E(p_0)^T E(p)\right) \tag{6.3}$$

and velocity level $v$

$$\begin{array}{rcl} D^v & = & B(v)^T D_0 E(p) + B(p)^T D_0 E(v) \\ & = & \left(B(v)^T B(p_0)\right)\left(E(p_0)^T E(p)\right) \\ & + & \left(B(p)^T B(p_0)\right)\left(E(p_0)^T E(v)\right) \end{array} \tag{6.4}$$

w.r.t. this initial torsion. The resulting matrices contain the whole information we need in order to calculate the rotational angles w.r.t. the initial position. A detailed derivation can be found in [Mut04].

The crucial part is how to define these angles. A difficulty of Eulerian an Cardan angles are two singular points, as described in section 4.2. These singularities are in a position corresponding to no rotation in Eulerian angles and a position where the z-axis is perpendicular w.r.t. the original axis in Cardan angles. Since we want to formulate force–elements near the initial orientation, i.e. a unity rotation matrix, we choose Cardan angles.

To determine the Cardan angles using a rotation matrix, one has to calculate the rotation matrix using Cardan angles, and then to compare. This is easily possible if the angles are small and a linear approximation is valid. If this prerequisite does not hold, a more complex algorithm has to be used. This still does not solve the problem of singularities, which is only treated by switching between the two definitions of angles. Because of the types of models of humans we want to study, we restrict ourselves to the simple first approximation. This decision is backed by knowing the fact that the whole treatment using spherical forces is a drastic simplification of the original biological situation in both cases.

### 6.2.2 Tensorial Forces

In the previous section we introduced spherical forces to model passive forces between two bodies connected by a spherical joint. In this sub–section we want to generalize this type of force to tensorial forces and apply them to a more realistic idealized model of the biological tissue.

In some cases, not only the rotational degrees of freedom are correlated, but also the translational ones, which results in the following formula:

$$f = \bar{C}x + \bar{D}\dot{x} \, , \tag{6.5}$$

where $x$ are the translational *and* rotational degrees of freedom, and $\bar{C}, \bar{D} \in \mathcal{R}^{6\times 6}$. An application of this kind of force–elements are intervertebral discs between the vertebras. Several authors [Den85],[DG87] and [Mut04] use so–called bushing–elements, i.e. special tensorial forces.

In this special case, the stiffness matrix $\bar{C}$ is set to

$$\begin{pmatrix} \begin{array}{l} x_0 \geq 0 : 140000 \\ x_0 < 0 : \ 50000 \end{array} & 0 & 8000 & 0 & -800 & 0 \\ 0 & 122000 & 0 & 450 & 0 & 300 \\ 8000 & 0 & \begin{array}{l} x_2 \geq 0 : \ 390000 \\ x_2 < 0 : 1083000 \end{array} & 0 & -380 & 0 \\ 0 & 450 & 0 & 179.9 & 0 & -1.5 \\ -800 & 0 & -380 & 0 & \begin{array}{l} x_4 \geq 0 : 151.8 \\ x_4 < 0 : 185.6 \end{array} & 0 \\ 0 & 300 & 0 & -1.5 & 0 & 149 \end{pmatrix} \tag{6.6}$$

and $\bar{D}$ to a diagonal matrix with translational elements of 300 and rotational ones of 1. All the translational forces are measured in $Ns/m$, and all the rotational ones are measured in $Ns/rad$.

More detailed approaches of the biological system are described in literature, e.g. [AL94], [MOR95], but they are too complex for the level of detail we want to discuss here. For this first step in parameter estimation of bio–mechanical models, we restrict ourselves to this simpler case.

### 6.2.3 Wobbling Masses

To better describe the soft tissue of the human body, like muscles and fat, we introduce wobbling masses as described in [GDSR87]. An additional body represents the soft tissue, which is connected by a nonlinear force to the original body, i.e. the skeleton. This additional mass is needed for a proper description of the motion of bio–mechanical models of humans.

The corresponding force–law is given as

$$f_i = \begin{array}{ll} c_t(p_i - p_{i0})^3 + d_t v_i & i = 1, 2, 3 \\ c_r(p_i - p_{i0}) + d_r v_i & i = 4, 5, 6, \end{array} \tag{6.7}$$

where the first three equations correspond to translational degrees of freedom, and the last three equations correspond to rotational degrees of freedom based on Cardan angles as described above. $c_t, c_r$ and $d_t, d_r$ are the associated stiffness of the spring and the coefficients of the damper.

## 6.3 Ground Contacts

When we model humans, there is an additional difficulty because of various contacts between the human and the surrounding environment. A fixed contact, which can be modelled using constraints, does not represent the reality appropriately. Therefore, we conventionally introduce soft constraints, i.e. force–elements and not kinematic constraints.

These forces are typically point–area forces. We simplify them using a small number of point–point Hertz type forces with certain properties as described in [Gue97].

This force model assumes that the size of the areas having contact increase up to a maximal value. Then a linear force law applies. Equation (6.8) represents the force law of a sphere with radius $r_0$ perpendicularly intruding into a surface. Parallel movement leads to a force against the motion (6.9).

$$f = \begin{cases} 0 & p > 0 \\ -a_0(p^2 - \frac{p^3}{3r_0^2}) & 0 > p > -r_0 \\ -a_0(p - \frac{r_0}{3}) & p < -r_0 \end{cases} \tag{6.8}$$

$$f^f = -f c_3 v. \tag{6.9}$$

Here $p$ denotes how much the sphere intrudes into the surface, $v$ is the component of the relative velocity between sphere and surface, which is parallel to the surface, $a_0$ is the stiffness of this contact and $c_3$ is the friction coefficient.

## 6.4 Muscles

Spherical or tensorial forces as described in the previous section are a very good first approximation to the real biological system, and they are a marginal extension of classical force–elements of technical mechanics, but they are not motivated from the biological system. In reality, the driving motion is produced using muscles. We will see that the modelling of this kind of elements is qualitatively different from previous force–elements. A thorough examination is given by Hatze [Hat81], van Soest [Soe92] and Günter [Gue97]. Here we provide an adaptation to Natural Coordinates.

Each biological muscle is composed of the muscle fibre with surrounding tissue and tendons connecting the whole system to the skeleton. These elements are connected in series, with the tendons as a first and last part. Abstractly speaking, a contractile element (CE) together with a parallel elastic element (PEE) is connected in series to two serial elastic elements (SEE).

### 6.4.1 Elastic Elements

Tendons, i.e. the SEE, are modelled as elastic deformable elements. They are important for human movement because they store kinetic energy and as such drastically influence the behavior of the human body especially in the case of running or walking. The exact behavior of tendons is studied in several experiments which get a negligible viscosity and only a small amount of tendon strain. As a result, tendons or, to be more exact, all connective tissue in series with the muscle fibre can be modelled as a nonlinear spring. Van Soest [Soe92] models the tendons using a second order polynomial. For elongations greater than zero we get

$$F_{SEE} = K_{SEE}\left(L_{SEE} - L_{SLACK}\right)^2 \quad . \tag{6.10}$$

$F_{SEE}$ is the resulting force of a SEE, $K_{SEE}$ the stiffness constant, $L_{SEE}$ the length of a serial elastic element, and $L_{SLACK}$ the maximal length of the tendon, where the force is zero. $K_{SEE}$ can be calculated from

$$K_{SEE} = \frac{F_{MAX}}{(U_{MAX} L_{SLACK})^2} \quad ,$$

where $F_{MAX}$ is the maximal force and $U_{MAX}$ the relative elongation at maximal isometric force.

From a structural point of view, the SEE and PEE are similar. As such, the PEE is modelled as a second order polynomial, too.

### 6.4.2 Contractile Elements

The muscle fibres are built out of smaller elements, which are called sakromeres. Out of the microscopic properties of these elements, one is able to propose an equation describing the force–length relation of a fibre

$$F_{isom}(l) = -cl^2 + 2cl - c + 1 \tag{6.11}$$

with

$$c = \frac{1}{width^2}$$

and

$$l = \frac{l_{CE}}{L_{CE_{opt}}} \quad ,$$

where $width$ is a parameter to describe the geometrical structure of the fibre, and $L_{CE_{opt}}$ the optimal operation length of the fibre if $v_{CE} = \dot{l}_{CE} = 0$. Out of $F_{isom}$ it is possible to compute the force, considering the activation $q$ and the maximal force $F_{max}$

$$F_{CE}|_{v_{CE}=0} = qF_{max}F_{isom}(l_{CE}) \quad . \tag{6.12}$$

The activity $q$ dependends on the concentration of $Ca^{2+}$ and other microscopic details of sakromeres that will not be discussed further. Models based on Hatze [Hat81] and Zajac [Zaj89] have already been developed.

A connection to moving muscles, i.e. $v_{CE} \neq 0$, was given by Hill [Hil38] using experimental knowledge. As we are only interested in passive muscles in the context of this work we do not want to discuss this further but refer to Günther [Gue97].

### 6.4.3 Geometry and Topology of Muscles

Until now have we discussed the length–force relation of muscles and tendons. Besides the fact that these are more complex w.r.t. previously discussed force–elements including additional differential equations to describe the internal structure in detail, we also have to consider the qualitatively more complex geometric structure of muscles. The muscles not only connect two "neighboring" bodies expressing forces depending on the length of a straight line between two body–fixed points or the corresponding equation on velocity level, but they typically connect non–neighboring bones, while the connection depends on the intermediate bones and is not a direct connection at all.

In the case of a two–dimensional model, the path of the muscle is easy to

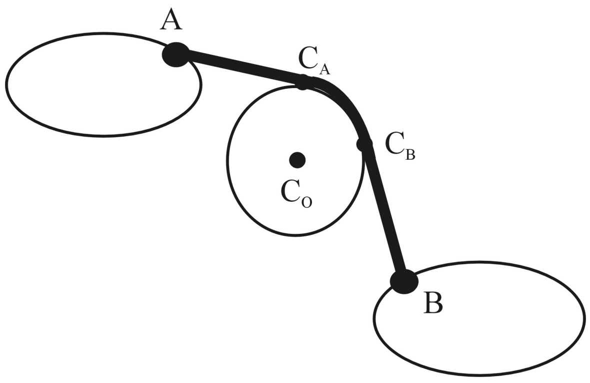


Figure 6.2: Muscle–path in 2D

compute. The length of the path only depends on the angles at the contact $C$, i.e. the angle between $\overline{C_A C_O}$ and $\overline{C_B C_O}$ (see figure 6.2). The relative position between $C_A$ and $A$ and $C_B$ and $B$ is of the same type as in previous models. An additional force has to be supplied at $C_O$, with a direction that is easy to calculate.

This changes when we model in 3D and when we consider the fact that the muscle need not have contact to $C$.

We introduce additional virtual points ($p_A, p_B$ corresponding to the contact points $C_A, C_B$) that define the path of the muscle. The position of these points is defined by constraints of the following form (here for $p_A$):

$$(p_A - o(p))^2 = r^2 \tag{6.13}$$

$$(p_A - o(p))^T (p_A - a(p)) = 0 \tag{6.14}$$

$$(p_A - o(p))^T d(p) = 0 \quad , \tag{6.15}$$

with the global position $o(p)$ of the origin of the contact–cylinder, the global direction $d(p)$ of its axis and the contacts on the bones $a(p)$. We want to stress that $o$, $a$ and $d$ are fixed positions respectively the axis w.r.t. to the moving frame of a body.

In order to retain the structure of the mechanical DAE, we define additional dynamic variables and additional kinematic constraints, where the dynamic variables $p_A$, $p_B$ correspond to zero mass. This leads to the following linear system

$$\begin{pmatrix} M & G^T & & \\ G & & & \\ & & \mathcal{O} & G_{A,B}^T \\ & & G_{A,B} & \end{pmatrix} \begin{pmatrix} a \\ \lambda \\ \ddot{p}_{A,B} \\ \lambda_{A,B} \end{pmatrix} = \begin{pmatrix} f \\ \gamma \\ 0 \\ \gamma_{A,B} \end{pmatrix} \quad , \tag{6.16}$$

where $G_{A,B}$ is the corresponding constraint matrix and $\gamma_{A,B}$ the acceleration independent part of the additional constraints of (6.13, 6.14, 6.15) for each virtual point. This is in addition to the original Augmented system.

Equations (6.13, 6.14, 6.15) and their first derivative w.r.t. time can be treated as invariants and are used to project to the consistent manifold.

**Part III**

# Optimization of Mechanical Systems

We now apply the parameter estimation algorithms described for ODEs in the first part to the mechanical DAEs as described in the second part. We will see that we have to cope with the high number of variables arising from using Natural Coordinates but again are able to use the structure and simplicity of the equations to gain an efficient and flexible algorithm. To summarize, this chapter discusses three aspects of the combined methods of the first two parts:

- The efficient calculation of the information necessary for a derivative–based method like the Generalized Gauß–Newton method.
- The treatment of arbitrary initial values of the underlying initial value problem in the context of an iterative scheme like the Generalized Gauß–Newton method.
- The parameterization of the model in a suitable and user–friendly way.

The first topic is of special importance because the generation of derivatives – or, to be more exact, the sensitivity of the solution of the underlying differential equation w.r.t. variations in some directions – is the most time–consuming part of the overall optimization algorithm. Due to the high dimension of models described which are based on Natural Coordinates, we have to apply reduced methods that calculate derivative information w.r.t. a minimal number of directions only. This can either be just one direction w.r.t. dynamic states, if one uses constraints which fix initial dynamic variables, or a number of directions equal to the degrees of freedom of the system and the number of parameters by using the constraints arising from the invariants of the mechanical DAEs.

Besides the reduction of the number of directions, we use the structure originating from the use of Natural Coordinates to efficiently calculate the right–hand side of the variational differential equation w.r.t. each direction. Because of the simplicity of the equations – the kinematic constraints are only quadratic equations w.r.t. the dynamic variables, and the mass matrix is sparse and constant – we are again able to achieve *linear complexity w.r.t. the number of bodies* with a very small scalar factor. Using internal numerical differentiation, we are able to reuse the decomposition of the Augmented system already done for the main trajectory. The evaluation of one directional derivative is approximately *two to three times faster* compared to the evaluation of the right–hand side of the differential equation.

Considering mechanical DAEs formulated based on the Full Descriptor Form, arbitrary initial states may violate the position– and velocity constraints, i.e. the invariants. They may be inconsistent. Besides using a non–redundant subset of the states as optimization variables, it is possible to relax, i.e. modify the differential equation such that arbitrary initial states are consistent and introduce additional constraints in a way that the states are consistent w.r.t. the original equations in the solution.

We present standard relaxation and non–stiff Baumgarte relaxation for mechanical systems. Moreover, we discuss singularities due to relaxation and reexamine non–stiff Baumgarte relaxation. The new idea is to take this approach to the limit by theoretically applying a damping factor of infinity. This is equivalent to an ideal projection of initial values and the use of standard drift avoidance techniques. This version of the projection combines the advantages of non–stiff Baumgarte relaxation, i.e. low error due to a drift from the non–relaxed manifold, without encountering the difficulties arising from certain singularities. We discuss several ideal projections with an increasing adaption to mechanical systems. The numerical tests of chapter 13 suggest that our combination is favorable in comparison to methods choosing a non–redundant subset of variables and to methods based on other types of relaxation.

Besides efficiency in terms of speed, we achieve efficiency in terms of flexibility and usability due to our version of Natural Coordinates. We show that some physical properties of the moments of inertia, which are difficult to guarantee if they are modelled based on, for example, Eulerian angles, are automatically fulfilled, if the mass matrix, which contains this information, is formulated based on Natural Coordinates.

Furthermore, our version of Natural Coordinates results in a fully parameterized model, which regains all design information in an explicit form that is easy to access. This state of the art property, which is regained for the software package due to the software approach we choose, distinguishes this approach from most others.

# Chapter 7

# Variational Differential Equation of Mechanical Models based on Natural Coordinates

The Generalized Gauß–Newton method requires the evaluation of derivatives. As already discussed in section 2.2, each row of the Jacobian matrix is calculated by the directional derivative of a component of the right hand side $r(y_0, q)$

$$\begin{aligned} \tfrac{\partial r_i(y_0,q)}{\partial y_0,q} D_0 &= \tfrac{\partial r_i}{\partial y(t_1)} \tfrac{\partial y(t_1)}{\partial (y_0,q)} D_0 + \tfrac{\partial r_i}{\partial y(t_2)} \tfrac{\partial y(t_2)}{\partial (y_0,q)} D_0 \\ &+ \ldots + \tfrac{\partial r_i}{\partial y(t_k)} \tfrac{\partial y(t_k)}{\partial (y_0,q)} D_0 + \tfrac{\partial r_i}{\partial q} D_{0,q}, \end{aligned} \tag{7.1}$$

where $q$ are the parameters and $y_0$ are the initial values. The columns of $D_0$ contain the initial directions, and $D_{0,q}$ is the part of $D_0$ corresponding to the parameters. In the case of multiple shooting, we have to calculate the derivative information w.r.t. the initial values $s_i$ of the corresponding multiple shooting interval. While the first part of each term is rather easy to compute, the second term

$$\frac{\partial y(t_i)}{\partial (y_0, q)} D_0$$

requires the derivative of the solution of the differential equation w.r.t. some initial directions $D_0$, i.e. the solution of the corresponding variational differential equation (VDE) w.r.t. these initial directions. This typically requires most of the computational time of the complete optimization algorithm.

Therefore, we study the efficient evaluation of the right–hand side of the VDE in general and in particular for models formulated in the Full Descriptor Form (section 2.1.1) based on Natural Coordinates. We will see that exploiting the structures, which originate from the use of Natural Coordinates and which were already mentioned in the simulation context, and the advantages of internal numerical differentiation allows for an efficient calculation of the solution of the VDE. The evaluation is of linear complexity w.r.t. the number of bodies and linear complexity w.r.t. the number of directions. This bilinear complexity is even better than a mere matrix–matrix multiplication, which is needed for a naive evaluation of the right–hand side of the VDE.

To further clarify the advantages of Natural Coordinates, we focus on presenting the derivative of the Augmented system of the Full Descriptor Form

$$\begin{pmatrix} M & G^T \\ G & \end{pmatrix},$$

which consists of the constraint matrix and the mass matrix. Because the constraints are quadratic equations, a *directional* derivative is very simple to compute, although the mathematical formula includes derivatives of matrices. The vectorial terms, like the forces or the constraints, follow the same principle and are as easy to evaluate. We will see that the solution of a linear system, based on the same matrix as in the simulation context, is the most time–consuming task. The efficient methods described for simulation are reused.

To do parameter estimation based on a dynamic model, one has to parameterize this model, i.e. define some basic properties of the model by parameters which are then optimized. In the simplest case, these parameters are formerly constants. Otherwise, if, for example, the structure of the model equations is used to reduce the dimension of the variables, or some constant values, which are defined by a function of constants corresponding to such basic properties, are pre–computed, it may be not longer possible to parameterized every property. As this is not desirable, we avoid this by providing a *fully parameterized model*. This is easily possible, because we do not use the structure to reduce the dimension, but exploit it. Furthermore, we thoroughly know the structure of our equations, and, therefore, are able to parameterize every property, even if, for performance reasons, we pre–compute constants. Together with the possibilities supported by the implementation shown in part IV parameters are just added to previously defined model description. This of special importance, if the model is already testes in the simulation context, because then it is guaranteed that we do not introduce additional implementation errors in an already tested model.

## 7.1 Evaluation of the Variational Differential Equations

We consider the variational differential equation, as in equation (2.20),

$$\begin{pmatrix} \dot{W}(t,t_0) & \dot{W}^q(t,t_0) \end{pmatrix} = \begin{pmatrix} \frac{\partial f}{\partial y} W(t,t_0) & \frac{\partial f}{\partial y} W^q(t,t_0) + \frac{\partial f}{\partial q} \end{pmatrix}$$

corresponding to a differential equation

$$\dot{y} = f(t,y,q)$$

and with some initial direction

$$W(t_0,t_0) = D_0,$$

where $D_0 \in \mathcal{R}^{n \times n_{Dir}}$ and $n$ the number of differential equations and $n_{Dir}$ the number of directions one is interested in. The huge computational effort arises from the fact that the variational differential equation is of dimension $n \cdot n_{Dir}$.

The main part of the VDE can be computed by

- first computing $\frac{\partial f}{\partial y}$ and then multiplying with $W(t, t_0)$ respectively $W^q(t, t_0)$
- or computing the directional derivative $\frac{\partial f}{\partial y} W(t, t_0)$ , $\frac{\partial f}{\partial y} W^q(t, t_0) + \frac{\partial f}{\partial q}$ directly.

If we look at the first approach, we see that the multiplication of the two terms, while neglecting the computation of the terms themselves, already has the complexity of a matrix–matrix multiplication and is $\mathcal{O}(n^3)$. We will see that we are able to compute the full directional derivative in $\mathcal{O}(n^2)$. For this reason, the first possibility is not considered further.

One possibility to calculate the derivative information is to approximate by using finite differences

$$\frac{\partial}{\partial x} f(x)|_{x=x_0} \Delta x \approx \frac{f(x_0) - f(x_0 + \epsilon \Delta x)}{\epsilon}. \tag{7.2}$$

The correct derivative is generated if we take the limit of $\epsilon \to 0$. But in this case, we encounter numerical difficulties arising from the cancellation error. Vice versa, a huge $\epsilon$ arises in discretization errors. Even an optimal choice leads to an accuracy of only the square root of the accuracy of the function evaluation. When considering second derivatives, it gets even worse. At the latest, then the achieved accuracy is not sufficient.

The other possibility is to provide the derivative explicitly in some way. We give the hand–optimized derivatives of the elementary terms, like the linear transformation (4.3), directly to provide upmost performance at the basic level, which is the inner loop of the evaluation, and use methods of automatic differentiation to calculate the derivative information for a combination of these elements. Therefore, we are able to automatically and efficiently provide a right–hand side of the VDE for arbitrary topology and elements.

## 7.2 Mechanical Systems

We have to consider the structures introduced by the index 1 form of mechanical DAEs, the so–called Full Descriptor Form as given in equations (2.14), (2.15) and (2.16).

The algebraic variables are the solution of a linear system. Therefore, we are able to compute them at every consistent state and eliminate these variables directly. Furthermore, we add a simple differential equation for parameters $q$ for notation purposes. This results in the following set of ODE's

$$\begin{aligned}\dot{p} &= v \\ \dot{v} = a &:= \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} M(q) & G(p,q)^T \\ G(p,q) & \end{pmatrix}^{-1} \begin{pmatrix} f(p,v,q) \\ \gamma(v,q) \end{pmatrix} . \\ \dot{q} &= 0\end{aligned} \tag{7.3}$$

The structure of these equations directly corresponds to a special structure of the corresponding variational differential equation

$$\begin{pmatrix} \dot{W}_p \\ \dot{W}_v \\ \dot{W}_q \end{pmatrix} = \begin{pmatrix} 0 & \mathcal{I} & 0 \\ \frac{\partial a}{\partial p} & \frac{\partial a}{\partial v} & \frac{\partial a}{\partial q} \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} W_p \\ W_v \\ W_q \end{pmatrix} . \tag{7.4}$$

The only non–trivial part is the derivative of the algebraic variable $a$. This variable is computed as the solution of a linear system. Using relation

$$\frac{\partial A(x)^{-1}}{\partial x} d = -A(x)^{-1} \frac{\partial A(x)}{\partial x} A(x)^{-1} d \quad \forall x \tag{7.5}$$

we are able to calculate the derivative of the inverse of a matrix using only the derivative of a matrix. Furthermore, we reuse the solution of the linear system

$$\begin{pmatrix} M & G^T \\ G & \end{pmatrix} \begin{pmatrix} a \\ \lambda \end{pmatrix} = \begin{pmatrix} f \\ \gamma \end{pmatrix}, \tag{7.6}$$

which is already available, and get

$$\frac{\partial a}{\partial x} = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} M & G^T \\ G & \end{pmatrix}^{-1} \left[ \frac{\partial}{\partial x} \left\{ \begin{pmatrix} M & G^T \\ G & \end{pmatrix} \right\} \begin{pmatrix} a \\ \lambda \end{pmatrix} + \frac{\partial}{\partial x} \begin{pmatrix} f \\ \gamma \end{pmatrix} \right], \tag{7.7}$$

which only includes derivatives of vectors and matrices.

A derivative of a matrix is a tensor of third order. To clarify the formula we introduce tensor notation

$$\frac{\partial}{\partial x^k} \left\{ \begin{pmatrix} M & G^T \\ G & \end{pmatrix}^i_j \right\} \begin{pmatrix} a \\ \lambda \end{pmatrix}^j \tag{7.8}$$

The direct calculation of this derivative is impractical not only due to memory usage. To circumvent this problem, we remember that we are not interested in this tensor itself. We multiply the derivative with one direction, and then multiply the resulting matrix with the solution of the linear system

$$\frac{\partial}{\partial x^k} \left\{ \begin{pmatrix} M & G^T \\ G & \end{pmatrix}^i_j \right\} d^k \begin{pmatrix} a \\ \lambda \end{pmatrix}^j, \tag{7.9}$$

where $d^k$ is a column of $W$ or $W^q$.

The result of this operation is a vector that can be treated very easily. We will see in the next section that Natural Coordinates and their simple structure allow for an efficient way to compute this vector directly.

At this point, we want to stress that, after computing the result from the derivative of the system matrix and the derivative of the force and $\gamma$-terms, we only have to solve a linear system. We are able to reuse the decomposition computed for the main trajectory if we use IND (section 2.2.1). This combination provides the high performance we need.

## 7.3 Special Structures due to Natural Coordinates

We now focus on the efficient computation of the directional derivative (term 7.9). Due to the simplicity of Natural Coordinates, we are able to compute this term with a very low computational effort.

The use of Natural Coordinates leads to a formalism based on local coordinate systems. That is, every body has its own set of variables and every joint connects only two bodies. Therefore, the system matrix is very sparse, and every entry in this matrix depends only on a small subset of the whole variable space.

### 7.3.1 Constraint Matrix

The constraint matrix is the derivative of the kinematic constraints w.r.t. the position variables. The kinematic constraints of typical joints are either linear or quadratic equations w.r.t. dynamic variables. In the first case we have:

$$g_p^1(p) = Ap \tag{7.10}$$

with some sparse matrix $A$ composed of transformation matrices $C_i$ and zero matrix blocks. For example, equality of two points, which defines a spherical joint, is given by

$$C_1 p_1 - C_2 p_2 = 0 \tag{7.11}$$

with small and sparse transformation matrices $C_i$ and the position vectors of only one body each.

In the case of constraints modelled using quadratic equations, we are able to provide a slightly more complex form

$$g_p^2(p) = p^T A_1^T A_2 p \tag{7.12}$$

with again very sparse matrices $A_i$ of similar structure.

#### Derivatives w.r.t. States

The derivative w.r.t. position, that is one row of the constraint matrix $G$, is then

$$G_i^1(p) = A \tag{7.13}$$

resp.

$$G_i^2(p) = p^T(A_1^T A_2 + A_2^T A_1). \tag{7.14}$$

Term (7.9) w.r.t. position is zero in the case of constraints based on linear equations or

$$d_p^{iT}(A_1^T A_2 + A_2^T A_1)a \tag{7.15}$$

respectively

$$d_p^{iT}(A_1^T A_2 + A_2^T A_1)^T \lambda \tag{7.16}$$

in the case of quadratic equations, where $d_p^i$ is the position part of $d^i$.

We arrive at a form where it is obvious that the calculation of the interesting term is very cheap. It is only slightly more expensive than the computation of the position constraints – which has a negligible computational effort especially if compared to several other approaches – or it generates no cost at all in the case of a linear kinematic constraints. As the acceleration–independent part of the acceleration constraint $\gamma(v)$ is either zero or a quadratic equations w.r.t. the dynamic variables, this term has a negligible computational effort compared to methods based on other formalisms, too.

#### Derivatives w.r.t. Parameters

At last we have to consider derivatives w.r.t. parameters which define the kinematic design of the model, i.e. define the position and orientation of the joints. As the joints are defined by a set of constraints, a parameterization of the joints is equivalent to a concerted parameterization of the constraints.

Reexamining the constraints as described above, we see that, depending on the exact type the constraint, the the directional derivative is a linear or quadratic equation w.r.t. the transformation matrices $C$. As we model based on Natural coordinates, this is equivalent to linear or quadratic equations w.r.t. the local definition of a point or an axis as described in equation 4.3. Therefore, directional derivatives w.r.t. parameters have the same nice property as directional derivatives w.r.t. dynamic variables: they are very easy and cheap to compute.

### 7.3.2 Mass Matrix

We compute the derivative of the mass matrix by directly computing the directional derivative as shown before. Compared to the derivative of the constraint matrix it is even less complex. When using Natural Coordinates, the mass matrix is very sparse and constant, i.e. independent of the state variables. For this reason the derivative w.r.t. state variables is zero. Moreover, the derivative of one mass matrix block corresponding to a body is only non–zero w.r.t. a parameter of the dynamic properties of this body. Therefore, the derivative of the mass matrix w.r.t. parameters is very sparse.

In the case of the mass matrix we encounter the additional difficulty that the entries of this matrix must fulfill additional conditions to be physically meaningful. If we first look at the the moments of inertia of one body, we encounter the following well–known conditions:

- no principal moment of inertia is bigger than the sum of the other two
- the matrix of the moments of inertia has to be positive definite and symmetric.

If we look at the matrix of the moments of inertia w.r.t. Eulerian angles (see equation 4.10)

$$\Theta_{ik} = \int \varrho(\tilde{r}_l^2 \delta_{ik} - \tilde{r}_i \tilde{r}_k) dV,$$

where $\delta_{ik}$ is the Kronecker symbol, we see that the first condition is expressed in the term $\tilde{r}_l^2 \delta_{ik} - \tilde{r}_i \tilde{r}_k$.

When modelling in Natural Coordinates, this term just does *not* appear (see equation 4.12)

$$M = \int_V \varrho \, C_r^T C_r \, dV = \begin{pmatrix} m & \\ & \frac{\operatorname{tr}\Theta}{2} - \Theta \end{pmatrix} \otimes \mathcal{I}_3,$$

but is introduced by the transformation between the two representations of the matrix of Inertia. Therefore, the first condition is *automatically* fulfilled in the case of the mass matrix formulated based on Natural Coordinates.

To guarantee a positive definite, symmetric matrix is more complex. One possibility is to use these conditions to guarantee the existence of a Cholesky decomposition

$$M = L^T L. \tag{7.17}$$

The six free entries of the lower left matrix $L$ are used as parameters. This choice automatically guarantees a symmetric and positive semi–definit mass matrix. To guarantee positive definiteness it is possible to apply simple inequality bounds to these parameters.

To summarize, the computational cost for the evaluation of (7.9) is negligible compared to a solution of the overall system. Furthermore, we were able to reuse the previously computed decomposition of the system matrix. As a result of this, our method is more accurate than methods based on finite differences. Furthermore, we were able to use the special structure of Natural coordinates to effectively compute derivative information, where the major computational effort just lies in the solution of the system. The calculation of one directional derivative is approximately *two to three times faster* than the evaluation of the main trajectory, as shown in part V.

# Chapter 8

# Generation of Consistent Initial Values for Mechanical DAE and Corresponding Variational Differential Equation

The initial values of the differential algebraic equation (DAE) must fulfil the algebraic conditions and the invariants. The same is true for the initial values of the corresponding variational differential equation (VDE). We now study the treatment of initial values which do *not* fulfil these conditions, i.e. are *not consistent*.

**Definition 7 (Consistency)**
*A state $x$ of a model, described by a DAE of index 1, is called* consistent, *if the algebraic conditions are fulfilled.*

In the case of the Full Descriptor Form, these algebraic conditions not only include the index–1 condition (2.15), but also the position and velocity constraints as invariants (2.16). The mathematical model only describes the physical reality correctly if the initial values and all intermediate states are consistent.

Arbitrary initial values are typically not consistent. To avoid in–consistent states, two methods can be applied:

The first approach partitions all the variables of the differential algebraic equation into a maximal set of non–redundant variables and the other ones. When considering this in the optimization context, we apply the optimization algorithm only to the non–redundant set, i.e. the optimization variables are a sub-set of all variables of the differential equation, while the other variables are calculated using the constraints and invariants. This method obviously does *not* lead to inconsistent initial values. The disadvantage of this approach is a typically higher nonlinearity, because we restrict ourselves to a nonlinear manifold of the flat complete space.

When studying models used in chemical engineering and in the context of optimization, we find that another approach leads to better convergence properties [LBBS02]. We observe that the solution of the consistency condition for the IVP is the inner loop of a non–linear optimization problem. As we are only interested in the optimal *solution* w.r.t. the correct physical model, we are free to work with modified models during the optimization process. To be more precise, we solve the consistency condition and the optimality condition together. Following this idea, we modify the model such that the IVP–problem is now formulated using consistent initial values, i.e. we *relax* it, and impose additional BVP–constraints, such that we get consistent initial values w.r.t. the original formulation in the solution.

After studying relaxation strategies applied to the general class of DAE with index 1, we adapt them to mechanical systems, where one has to take into account that, for example, the velocity constraint is the time–derivative of the position constraint. In the case of mechanical systems, we present several simple examples where the thoughtless use of relaxation may lead to problems. Singularities may appear. Therefore, to avoid such difficulties in the general case, we reexamine the existing relaxation techniques in this context.

Because of this difficulty, we modify one of the relaxation strategies, the non–stiff Baumgarte relaxation, by taking it to the limit w.r.t. damping, which is equivalent to an adapted projection. As we will see in part V, this method avoids the difficulties arising from singularities and provides the better results than the other relaxation techniques in the test set we examined numerically.

After considering consistent initial values of the differential equation, we study methods for the computation of initial values of the corresponding variational differential equation and the corresponding invariants they have to fulfil. We exploit the structure and our knowledge of the system again to provide a fast method of linear complexity to calculate each initial direction. This is especially important because solving an parameter estimation problem without using the structure is of $\mathcal{O}(n^3)$ and has to be applied at every multiple shooting node. For an increasing dimension this would dominate the cost of the complete optimization process.

## 8.1 Relaxation Strategies

We first study different relaxation strategies for DAEs of index 1. A specialization to mechanical DAEs in the Full Descriptor Form including the treating of invariants is given in the next section.

By applying relaxation we modify the original IVP–constraints $g$ such that the state $x$ of the model is consistent w.r.t. the new equations. The simplest form is

$$\hat{g}(x, x_0) := g(x) - g(x_0). \tag{8.1}$$

We subtract the initial inconsistency $g(x_0)$ calculated w.r.t. the initial values $x_0$.

Besides this simple form, we describe more sophisticated versions in the following, which try to improve the convergence of the optimization algorithm.

**Non–Stiff Baumgarte Relaxation** If we start with an initial value far off the solution–manifold, this alters the derivative information we get by solving the variational differential equation, due to the fact that we evaluate the variational differential equation based on a trajectory which is far of a consistent trajectory. With this in mind, it is opportune to reduce this effect using a adequate relaxation technique. This method should treat arbitrary initial values, and it should reduce the inconsistency, i.e. the distance from the non–relaxed manifold, during integration. We choose to reduce it exponentially

$$\hat{g_B}(x, x_0) := g(x) - g(x_0)e^{-\alpha\frac{t-t_o}{t_e-t_0}}, \tag{8.2}$$

with the damping constant $\alpha$, and we examine the integration interval $[t_0, t_e]$. The inconsistency $g(x(t))$ is exponentially damped with a factor $e^{-\alpha}$ at the end of the integration interval. This approach follows Baumgarte stabilization used to avoid drift in the simulation context, but it does not impose additional numerical stiffness for damping factors $\alpha$ that are not too high. As we know the "error", i.e. $g(x_0)$, and the time interval, we are able to construct the reduction accordingly. Therefore, it is called non–stiff Baumgarte relaxation [BEJ88], [SBS98].

## 8.2 Invariants of Mechanical Systems

In the case of mechanical systems and index reduction, we additionally have to consider the position and velocity invariants. It is important that the velocity constraints are the time–derivative of the position constraints, which has to be retained if the constraints are relaxed.

The Augmented system (2.15), which contains the algebraic constraints, is linear. Therefore, it is adequate to eliminate the algebraic variables including the acceleration directly. In this case we implicitly use the first method of treating ”in–consistent” initial variables. Due to the linear nature of the system, there is no underlying non–linearity.

This is different in the case of invariants. Considering a velocity invariant in simple form

$$\hat{g}_{vel}(p, v, p_0, v_0) := g_{vel}(p, v) - g_{vel}(p_0, v_0), \tag{8.3}$$

the proper position invariant is

$$\hat{g}_{pos}(p, p_0) := g_{pos}(p) - g_{pos}(p_0) - (t - t_0)g_{vel}(p_0, v_0). \tag{8.4}$$

This form of a relaxed velocity constraint retains the property to be the first time–derivative of the position constraint. A corresponding modification of non–stiff Baumgarte relaxation

$$\begin{aligned}\hat{g}_{pos}(t, p) := \;& g_{pos}(p) \\ - \;& \Big[g_{pos}(p_0) + (g_{vel}(p_0, v_0) + \alpha g_{pos}(p_0))(t - t_0) \\ + \;& (\alpha g_{vel}(p_0, v_0) + \tfrac{\alpha^2}{2} g_{pos}(p_0))(t - t_0)^2\Big]\, e^{-\alpha(t-t_0)}\end{aligned} \tag{8.5}$$

$$\begin{aligned}\hat{g}_{vel}(t,p,v) := \;& g_{vel}(p,v)\\ -\;& \Big[g_{vel}(p_0,v_0)+\alpha g_{vel}(p_0,v_0)(t-t_0)\\ -\;& (\alpha^2 g_{vel}(p_0,v_0)+\tfrac{\alpha^3}{2}g_{pos}(p_0))(t-t_0)^2\Big]\,e^{-\alpha(t-t_0)}\end{aligned} \tag{8.6}$$

$$\begin{aligned}\hat{\gamma}(t,p,v) := \;& \gamma(p,v)\\ +\;& \Big[-(3\alpha^2 g_{vel}(p_0,v_0)+\alpha^3 g_{pos}(p_0))(t-t_0)+(\alpha^3 g_{vel}(p_0,v_0)\\ +\;& \tfrac{\alpha^4}{2}g_{pos}(p_0))(t-t_0)^2\Big]\,e^{-\alpha(t-t_0)}\end{aligned} \tag{8.7}$$

is given in [SBS98]. Further studies about relaxation strategies can be found in [Sto04].

## 8.3 Singularities

The difficulties arising from singularities which are inherent in the model (section 5.3) or introduced by the mathematical description (section 5.2) in the simulation context must be treated in the optimization context, too. Additionally, we are confronted with singularities arising from the relaxation strategy.

### 8.3.1 Redundant Constraints due to Relaxation

Using the Full Descriptor Form, we have to solve for the acceleration (and Lagrange multipliers), which includes a decomposition of the system matrix. A unique, non–trivial solution is only possible if this matrix is non–singular, which implies full rank of the constraint matrix (see (2.1.1)). We have to ensure that the constraints are not redundant.

If one arbitrarily changes the state of the model, the kinematic design of the model is changed, too. Besides various other effects, we first want to study an extreme case. A constraint introduces a singularity into the system matrix because the derivative is zero.

Let us study an example of a simple one–dimensional kinematic constraint fixing the distance of some coefficient $p_i$ of the position vector to the origin

$$g_{pos}(p) = p_i^2 - l^2 \tag{8.8}$$

with a (simple) relaxed form of

$$\hat{g}_p(t,p,t_0,p_0,v_0) = p_i^2 - l^2 - p_{i,0}^2 - 2p_{i,0}v_{i,0}(t-t_0) \tag{8.9}$$

where $p_i, v_i$ are components of $p$ repectively $v$ with initial values $p_{i,0}$ and $v_{i,0}$. The constraint matrix row is

$$G_j(p) = p_i\delta_j^i. \tag{8.10}$$

This is exactly zero if $p_i$ is zero, i.e. an initial value of zero leads to a model that is not defined properly, and a modification of the model in either direction will lead to models with completely different physical properties. We are not able to do derivative–based optimization, if we reach or cross this point.

One might say it is very unlikely that exactly this state is chosen as the initial value. But let us consider the more realistic case that the model is not in a singular position at the beginning, but on one side of the singularity with an initial inconsistent velocity propelling it into the singularity.

As we have seen in this example, even simple constraints like a length condition can lead to unsolvable problems if an arbitrary modification of the kinematic design of the model is allowed. This includes modifications of the initial state in the context of optimization. A practical relaxation technique must pay attention to these problems.

One possible solution will be introduced in section 8.4. A possible mathematically inspired solution is given by Stossmeister in [Sto04] using a reformulation based on Gröbner bases.

### 8.3.2 Redundant Constraints

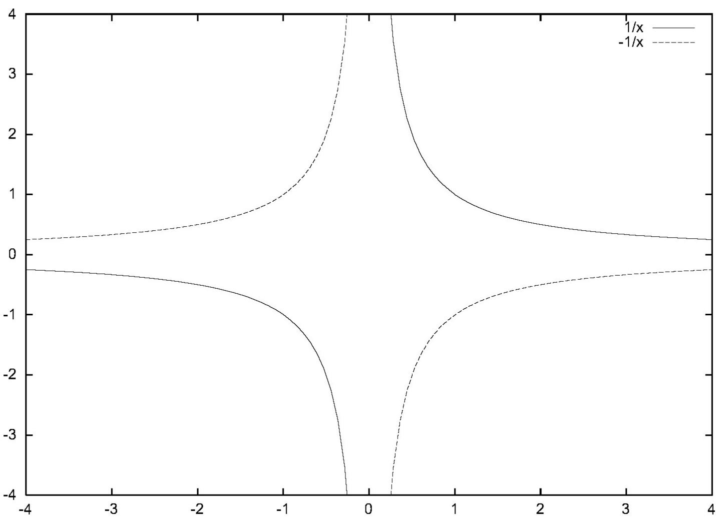


Figure 8.1: Two relaxed manifolds due to different initial values

We have seen in section (5.3) that the mechanical model itself can introduce kinematic singular positions. To study this behavior, we first want to examine a simple problem that is typical for this kind of singularity.

We study a sub–manifold of a 2–dimensional model which is defined by the following constraint

$$g_{pos}(p_1, p_2) = p_1 p_2. \tag{8.11}$$

Now we use inconsistent initial values and relax this constraint (ignoring velocity inconsistency)

$$\hat{g}_p(p_1, p_2, p_1^0, p_2^0) = p_1 p_2 - p_1^0 p_2^0. \tag{8.12}$$

Looking at graph (8.1) of this equation w.r.t. the inconsistency $p_{1,0}p_{2,0}$, we see that the solution of a differential equation w.r.t. similar initial values

$$\begin{pmatrix} p_{1,0} & p_{2,0} \end{pmatrix} = \begin{pmatrix} p_{1,0} & 0 \pm \epsilon \end{pmatrix} \tag{8.13}$$

discontinuously depends the small variation $\epsilon$. Furthermore, the critical point of the optimization is for the not varied value with $\epsilon = 0$.

Moreover, the method of detecting redundant constraints described in section (5.2) depends on exact kinematic design information. An arbitrary change of the initial state, resulting in a changed kinematic design, leads to completely changed properties. This includes a change in the number of degrees of freedom. Discontinuous models of this kind can not be treated with classical Gauß–Newton methods; therefore we want to avoid discontinuous models.

As a result of this, it is worth reexamining the question if relaxation techniques which produce excellent results with index–1 models of e.g. chemical engineering are as appropriate in mechanical problems.

## 8.4 Taking non–stiff Baumgarte Relaxation to the Limit

Due to the difficulties arising from the singularities discussed in the previous section, we now reconsider non–stiff Baumgarte relaxation. We will see that taking a damping factor to the limit enables us to treat the difficulties while probably the performance of the relaxation in terms of convergence of the parameter estimation problem is improved for certain situations.

Allowing arbitrary initial values leads to inconsistent initial values, which are treated using a relaxation technique. We study the effect of relaxation based on the assumption that consistency is an information that should be used to increase the performance of the complete algorithm. In this sense it follows one of the motivations of multiple shooting – to start with an initial trajectory as near to the solution trajectory as possible. A special case, where the following line of thought does not hold, is when the manifold that is defined by the kinematic constraints depends on parameters. On the other hand, we do not assume that this approach will hamper in that situation.

Let us examine the following nonlinear least squares problem

$$\begin{array}{rl} \min_{x_0,q} \sum_{i=1}^{n_{ls}} f_1^i(x(t_i; x_0, q, \mathcal{R}(x_0)), q)^2 & \\ f_c^i(x(t_i; x_0, q, \mathcal{R}(x_0)), q) = 0 & i = 1, ..., n_c \\ h^i(x(t_i; x_0, q, \mathcal{R}(x_0)), q) = 0 & i = 1, ..., n_h \\ g^i(x_0) = 0 & i = 1, ..., n_{kin}, \end{array} \tag{8.14}$$

where $f_1$ are the least squares conditions, $f_c$ are associated to the boundary conditions of the underlying DAE, $h$ correspond to the matching conditions which are formulated w.r.t. directions in the tangent space of the manifold, and $g$ are the consistency conditions which are formulated w.r.t. directions perpendicular to the tangent space of the manifold. $\mathcal{R}$ is the relaxation technique used to modify the "original DAE" such that it is possible to treat arbitrary initial state variables. We want to add, that $f_1, f_c$ and $h$ typically depend on the solution $x(t_i)$ of the differential equation at certain time–points $t_i$, which depends on the initial values $x_0, q$ and the relaxation $\mathcal{R}$. In contrast to that, we assume that the consistency conditions $g$ do not depend on the relaxation (and, as mentioned above, not on the parameters), because they give conditions for the initial values of the underlying IVP. We want to stress here that the solution of the nonlinear problem does not depend on the relaxation, because the last condition $(g(x))$ is fulfilled in the solution, i.e. the optimization variables are consistent w.r.t. the invariants of the underlying DAE.

### 8.4.1 Motivation for the Limit Form of non–stiff Baumgarte Relaxation

To give an impression of the effect of a specific relaxation method, we want to consider a DAE of index 1. The situation is more complex for mechanical systems, as there is a relation between position and velocity invariants, but the general line of thought remains the same. If we apply the simplest form of relaxation (see equation (8.1))

$$\hat{g}(x, x_0) := g(x) - g(x_0),$$

the distance of the relaxed manifold is constant and equal to the initial inconsistency $g(x_0)$, where $g$ is the original invariant of the underlying DAE. In the case of non–stiff Baumgarte relaxation (8.2)

$$\hat{g_B}(x, x_0) := g(x) - g(x_0)e^{-\alpha \frac{t-t_o}{t_e-t_0}},$$

we damp this initial inconsistency exponentially with a damping factor $\alpha$. Thus, the distance of the trajectory using non–stiff Baumgarte relaxation to the consistent manifold is less than in the case of applying simple relaxation. As the consistent manifold does not depend on the parameters, i.e. is already known during the optimization, it is better to keep the relaxed trajectory as near to the consistent manifold as possible, because this should result in a trajectory near to the solution trajectory. In that sense, a perturbation perpendicular to the manifold can be seen as a worse initial guess compared to the consistent value.

Therefore, it seems to be suitable to damp the effect of such a perturbation as much as possible. Unfortunately, there seems to be a contrary effect, because an increasing damping factor $\alpha$ increases the (numerical) stiffness of the differential equation, and, thus, a medium–sized damping factor should be used to balance these effects. This is a result described in [Lei95] for chemical DAEs of index 1.

Fortunately, this effect does not appear if we choose a theoretical damping factor which goes to infinity, which is equivalent to projecting the initial values to the manifold and then using standard drift avoidance techniques, i.e. invariant projection, during the simulation. This effect is comparable to not using Baumgarte *stabilization* but invariant projection in order to avoid drift, which avoids the introduction of (mechanical) stiffness as it is well known for using Baumgarte stabilization in the simulation context.

Applying this form of non–stiff Baumgarte relaxation, which we call the limit form, thus eliminates the (bad) effect of inconsistent initial values to the trajectory, if the manifold is not defined by unknown parameters. Besides the effect of better sensitivity information due to a trajectory which should be near to the solution trajectory, the integration is done on the consistent trajectory, which obviously avoids the difficulties discussed in section 8.3. Therefore, the limit form of non–stiff Baumgarte relaxation is an option to consider. The numerical tests shown in chapter 7 suggest that it is better than classical relaxation techniques and consistent methods for parameter estimation based on mechanical DAE that are formulated using Natural Coordinates.

### 8.4.2 Comparison to Methods using Consistent Initial Values

This form of relaxation seems to be comparable to consistent methods which only use a non–redundant set of variables or always keep the initial values of the underlying differential algebraic equation, i.e. the dynamic part of the optimization variables, consistent w.r.t. the original invariants. This is not the case because the latter corresponds to the following problem

$$\begin{array}{ll} \min_{\pi(x_0),q} \sum_{i=1}^{n_{ls}} f_1^i(x(t_i;\pi(x_0),q),q)^2 & \\ f_c^i(x(t_i;\pi(x_0),q),q) = 0 & i=1,...,n_c \\ h^i(x(t_i;\pi(x_0),q),q) = 0 & i=1,...,n_h \\ g(\pi(x_0)) = 0 & i=1,...,n_{kin}, \end{array} \tag{8.15}$$

where the the minimization is done w.r.t. projected optimization variables $\pi(x)$, i.e. the invariant conditions $g$ are fulfilled in a nonlinear sense during the optimization based on linearized auxiliary problems. As algorithms based on relaxation use optimization variables which are inconsistent w.r.t. the underlying DAE with invariants, the methods are not equal.

## 8.5 Different Types of Projection for the Limit Form of Non–Stiff Baumgarte Relaxation

We have showed that the limit form of Non–Stiff Baumbarte stabilization is equivalent to an ideal projection to the allowed manifold. In the following we compare several types of ideal projections.

### 8.5.1 Projection During Integration – Avoiding Drift

Discussing how to project according to the limit form of Non–Stiff Baumgarte Relaxation, one has to distinguish between the initial projection at the start of the integration interval and the projection done to avoid drift away from the manifold during the integration. The latter projects values which are almost consistent. Therefore, the exact type of projection is not important and we prefer the most efficient one. We use the invariant projection w.r.t. Energy norm as described in the following (as in e.g. integration package MBSSIM [Sch99], [Win01]).

First, we generate a consistent initial position by fulfilling the position constraints. This is equivalent to the following algebraic constrained minimization problem

$$\begin{array}{c} \min_p \|p - p^0\|_M^2 \\ g_p(p) = 0. \end{array} \tag{8.16}$$

The distance of the optimized position $p$ and the original position $p^0$ is minimal in some norm – here the mass matrix norm – w.r.t. the position constraint. As a second step, we fix this position and state the following problem

$$\begin{array}{c} \min_v \|v - v^0\|_M^2 \\ g_v(p, v) = 0, \end{array} \tag{8.17}$$

which projects on the velocity level using the original velocity $v^0$ and the already projected position $p$.

The usage of the mass matrix norm is adapted to mechanical systems, because in the case of velocity projection it asks for a a minimal energy difference [Ali92]. Furthermore, if we project the position and velocity variables independently, we are able to reuse the decomposition already done to evaluate the right–hand side of the differential equation. The linearized form of the projection on the position level is

$$\begin{pmatrix} M & G(p^i)^T \\ G(p^i) & \end{pmatrix} \begin{pmatrix} \Delta p^i \\ \lambda \end{pmatrix} = - \begin{pmatrix} p^i - p^0 \\ g_p(p^i) \end{pmatrix}. \tag{8.18}$$

The same matrix is used for the projection on the velocity level.

### 8.5.2 A First Algorithm to Project Initial Values

Although the exact type of projection is less important during the simulation, it does have an impact in the case of the first projection of the initial values. As the solution of the differential equation is in the inner loop of an iterative scheme, which ”arbitrarily” changes the initial values, these values can be far off the consistent manifold, at least compared to the situation when projection is used to avoid drift during the integration.

A first approach uses the same idea already proposed for the treatment of inconsistent initial values to avoid relaxation completely. We select a subset of independent variables $x_n$ and calculate the other variables $x_d$, such that

$$\begin{array}{c} \min_{x_n,x_d} \|x_n - x_n^0\|^2 \\ g(x_n, x_d) = 0, \end{array} \tag{8.19}$$

where $g$ are all invariants $g_p$, $g_v$ and $\frac{\partial g}{\partial x_d}$ is regular. This is equal to fixing some variables and calculating the other variables, such that the constraints are fulfilled. As we will see in the next chapter, it is sometimes important to compare results of a projection at different states. Using this method, we apply a pivoting strategy to find the variables $x_n$, which leads to difficulties if we use different pivot vectors for the two states. As we are no longer able to do such a projection in the extreme case, where no common pivot vector can be found such that $\frac{\partial g(x_n^1,x_d^1)}{\partial x_d}$ and $\frac{\partial g(x_n^2,x_d^2)}{\partial x_d}$ are regular at two points $(x_n^1, x_d^1)$ and $(x_n^1, x_d^2)$, a better approach is needed.

### 8.5.3 Full Euclidean Projection for Initial Values

A better approach in terms of differentiability and adaption to the local manifold uses an Euclidean projection on the full space. Reusing the definition of $g$, we write

$$\begin{array}{c} \min_x \|x - x^0\|_2^2 \\ g(x) = 0. \end{array} \tag{8.20}$$

In this case we project w.r.t. a direction perpendicular to the allowed manifold. Therefore, the result of the ideal projection is differentiable w.r.t. the initial values.

The Augmented system of the linearization of this non–linear projection is

$$\begin{pmatrix} \mathcal{I} & & G^T & G_v^T \\ & \mathcal{I} & & G^T \\ G & & & \\ G_v & G & & \end{pmatrix} \begin{pmatrix} \Delta p^i \\ \Delta v^i \\ \lambda_p \\ \lambda_v \end{pmatrix} = - \begin{pmatrix} p^i - p^0 \\ v^i - v_0^i \\ g_p(p^i) \\ g_v(p^i, v^i) \end{pmatrix}, \tag{8.21}$$

where $G_v := \frac{\partial g_v}{\partial p}$ and, as we recall, $\frac{\partial g_v}{\partial v} \equiv G$.

### 8.5.4 Projection Adapted to Mechanical Systems

The last approach, which is discussed here, does use two independent projections. It was already used for the projection to avoid drift during the simulation (see equations 8.16 and 8.17). The main difference is that the position is fixed during the second projection, i.e. the position and velocity variables are more independent, than in the full Euclidean case shown in the previous subsection. This version is more adapted to mechanical systems and the results of numerical experiments, shown part V, suggest that this approach leads to better convergence properties of the overall iterative scheme.

To compare to the Euclidean projection as presented in the previous subsection, we combine the linearized forms of both ideal projections.

$$\begin{pmatrix} M & & G^T & 0 \\ & M & & G^T \\ G & & & \\ G_v & G & & \end{pmatrix} \begin{pmatrix} \Delta p^i \\ \Delta v^i \\ \lambda_p \\ \lambda_v \end{pmatrix} = - \begin{pmatrix} p^i - p^0 \\ v^i - v_0^i \\ g_p(p^i) \\ g_v(p^i, v^i) \end{pmatrix}. \qquad (8.22)$$

The only difference is the zero matrix block in the upper right corner.

### 8.5.5 Projection of the Initial Direction of the Variational Differential Equation

Besides the projection of the initial states, we have to study the projection of the initial directions, i.e. the initial states of the corresponding variational differential equation.

The invariants for initial directions $d$ are

$$\frac{\partial g}{\partial x} d = 0. \qquad (8.23)$$

We introduce the following abbreviations: $G = \frac{\partial g_p}{\partial p}$ is the constraint matrix of the mechanical system, $G_v = \frac{\partial g_v}{\partial p}$, $G_p^q = \frac{\partial g_p}{\partial q}$ and $G_v^q = \frac{\partial g_v}{\partial q}$. Furthermore, we remember that $\frac{\partial g_v}{\partial v} \equiv G$.

In the case of mechanical systems, we specialize these invariants to

$$\begin{pmatrix} G & & G_p^q \\ G_v & G & G_v^q \end{pmatrix} \begin{pmatrix} d_p \\ d_v \\ d_q \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}. \qquad (8.24)$$

Because of the structure of the derivatives, we want to keep $d_q$ constant:

$$\begin{pmatrix} G & \\ G_v & G \end{pmatrix} \begin{pmatrix} d_p \\ d_v \end{pmatrix} = \begin{pmatrix} -G_p^q d_q \\ -G_v^q d_q \end{pmatrix}. \qquad (8.25)$$

To give a unique solution, we additionally state a minimization of the difference between initial and projected direction using some norm. Depending on this norm, the method is compatible with the corresponding projection of the initial state. In the case of Euclidean projection, we get the following linearized system:

$$\begin{pmatrix} \mathcal{I} & & G^T & G_v^T \\ & \mathcal{I} & & G^T \\ G & & & \\ G_v & G & & \end{pmatrix} \begin{pmatrix} d_p \\ d_p \\ \lambda_p \\ \lambda_v \end{pmatrix} = - \begin{pmatrix} d_p^0 \\ d_v^0 \\ -G_p^q d_q \\ -G_v^q d_q \end{pmatrix}. \qquad (8.26)$$

We see that the system matrix is the same as for the projection of the states. This property is regained in principle for the other variants. Therefore, we do not repeat the formulas.

In the case of the mechanically adapted version, we are again able to reuse the highly efficient decomposition and solution algorithms developed to solve the Augmented system corresponding to the acceleration constraints of the mechanical DAE.

## 8.6 Computing Initial Directions

Finally, we want to discuss the very important step of generating the necessary initial directions for the invariant based reduction methods as described in the next chapter. These directions must fulfil the consistency conditions adapted to the corresponding VDE, i.e. they must be consistent w.r.t. the according invariants. Furthermore, they should be unique because we want to have a continuous formulation of the basis of the tangent space (see relation 9.30). Besides this, the computation should be as fast as possible. Furthermore, we want to reuse existing code to minimize the amount of the additional implementation.

This work is done in two steps. First we generate a suitable set of good initial guesses which is inspired by Relative Coordinates of the open loop case to avoid the treatment of the large basis of the full space. Then we generate a basis using this generating system.

### 8.6.1 Generating Initial Guesses

A first guess of initial directions would be to use the unity matrix, i.e. a basis of the full space. This selection already fulfils one of the conditions we aim at. It is guaranteed that we do not miss a direction. Unfortunately, there are too many initial directions, which leads to an increased computational cost. The redundancy of the resulting basis (after projection) will not lead to a methodic problem, but again to an increased computational cost.

Therefore, we look for a better approach with less directions. In the case of an open–loop system the minimal set of variables is, for example, provided by Relative Coordinates. Comparing this approach to a formalism with local coordinate systems, like when applying Natural Coordinates, we see that these variables correspond to directions not restricted by constraints in the our case. Therefore, if we use the directions corresponding to constraints not formulated for a specific joint, we get a basis of the tangent space of the consistent manifold. Talking of corresponding directions, we use the derivative of the constraints w.r.t. position variables. This is equivalent to the fact, that one row of the constraint matrix corresponds to a direction perpendicular to the manifold.

The resulting initial directions are then projected to fulfill the corresponding invariants according to section (8.5.5).

### 8.6.2 Treatment of Redundancy

In the case of automatically generated models, which include closed kinematic loops, redundant constraints may occur. This does not impede the computation of consistent initial directions as described in the previous sub–section, because we use the linear algebra method which is able to treat such situations as introduced in the integration context (see section 5.2).

The approach stated above uses all directions that correspond to the degrees of freedom of the open–loop system. Consequently, we may have calculated the sensitivity information w.r.t. too many initial directions. One possibility to avoid this is to orthogonalize the generating system. Based on this, it is possible to provide a basis of the tangent space. Thus, we are able to calculate sensitivity information w.r.t. a minimal number of directions.

# Chapter 9

# Various Reduced Methods for Mechanical DAEs

The most time–consuming task of the optimization algorithm, is to calculate the solution of the VDE w.r.t. some directions. The methods of this chapter try to minimize the number of directions one has to calculate explicitly. This complements the topics of chapter 7, which discussed the efficient solution w.r.t. one direction. We examine different methods to use various information already present in the optimization problem.

- initial or interior point constraints of the boundary value problem (BVP) (see equation (1.2))
- algebraic constraints respectively invariants of the mechanical DAEs (see (2.9,2.15) and (2.16))

In the first case we use BVP–constraints, to fully specify a linear combination of our increment without calculating the corresponding derivative of the underlying dynamic system. This is the so–called Fixfit–approach of Schlöder [Sch88]. In the special case of fixed initial states, the number of derivatives one has to compute is the number of parameters plus one.

The second approach, which is based on [SBS98], recognizes the connection of invariants of the initial value problem (IVP) to implicitly or explicitly given BVP–constraints. As, in the solution, invariants and acceleration constraints must be fulfilled at every state, this is especially true for the start of each multiple shooting interval. Therefore, one has to specify corresponding BVP–constraints to ensure a consistent solution of the optimization problem, if the differential equations are relaxed. These BVP–constraints can be used to eliminate the need to generate derivative information w.r.t. these directions according to the Fixfit–approach. Due to the number of invariants and, for this reason, the number of BVP–constraints per multiple shooting node, one has to generate dynamic derivatives w.r.t. a number of directions equal to the degree of freedom of the mechanical model plus the number of parameters.

A major difference of the second approach compared to the Fixfit–approach is that the BVP–constraints originating from invariants are formulated at every multiple shooting node independently. This allows for an independent calculation of directional derivatives of the solution of the differential equation and as such for a parallelization of this method w.r.t. the multiple shooting discretization.

For the latter approach we present two versions. Both use the BVP–constraints to avoid the need for generating dynamic derivative information w.r.t. the corresponding directions. The first version then computes sensitivity matrices $H = \frac{\partial h(s_i, s_{i+1})}{\partial s_i}$, which have the same dimension as the original full system. The second approach additionally uses the structure of the BVP–constraints w.r.t. the invariants to reduce the amount of work which has to be done in the context of the condensing algorithm.

All these methods have to take into account the treatment of inconsistent initial values and the corresponding relaxation strategy with the eventual projection algorithm. We will see that not all the projection methods of the limit form of the non–stiff Baumgarte relaxation can be applied to every reduction method.

## 9.1 Using BVP–Constraints to reduce the computational effort

The basic idea of the Fixfit–approach of Schlöder [Sch88] is to use initial value constraints to implicitly eliminate some variables, and thus to reduce the computational effort of derivative generation *and* condensing, i.e. decomposing and solving the part of the system originating from the multiple shooting discretization.

We start with the condensed system (2.36) and partition the vector of increments

$$\begin{pmatrix} \Delta s_0 \\ \Delta p \end{pmatrix} = \begin{pmatrix} \Delta s_{01} \\ \Delta s_{02} \\ \Delta p \end{pmatrix}, \tag{9.1}$$

such that $D_{21}$ of the following system has full rank

$$\min_{\Delta s_{01}, \Delta s_{02}, \Delta p} \|D_{11}\Delta s_{11} + D_{12}\Delta s_{12} + D_{p1}\Delta p + d_1\|_2^2 \tag{9.2}$$

$$D_{21}\Delta s_{01} + D_{22}\Delta s_{02} + D_{p2}\Delta p + d_2 = 0 \tag{9.3}$$

$$D_{31}\Delta s_{01} + D_{32}\Delta s_{02} + D_{p3}\Delta p + d_3 = 0. \tag{9.4}$$

Here $d_1$ are least square conditions, $d_2$ are constraints fixing initial values, while $d_3$ are constraints w.r.t. intermediate points or end–points.

$\Delta s_{01}$ will be formally eliminated, which results in a condensed system of $\Delta s_{02}$ and $\Delta p$ only:

$$\min_{\Delta s_{02}, \Delta p} \|\tilde{D}_{12}\Delta s_{02} + \tilde{D}_{p1}\Delta p + d_1\|_2^2 \tag{9.5}$$

$$\tilde{D}_{32}\Delta s_{02} + \tilde{D}_{p3}\Delta p + d_3 = 0. \tag{9.6}$$

To calculate this subdivision, we need a basis of the space perpendicular to the tangent space defined by the constraints $d_2$. We define

$$\begin{array}{rcl} {}^{-1}H_E & := & \begin{pmatrix} -D_{21}^{-1}D_{22} \\ \mathcal{I} \end{pmatrix} \\ {}^{-1}H_P & := & \begin{pmatrix} -D_{21}^{-1}D_{p2} \\ \mathcal{I} \end{pmatrix} \\ {}^{-1}H_R & := & \begin{pmatrix} -D_{21}^{-1}d_2 \\ \mathcal{I} \end{pmatrix}, \end{array} \tag{9.7}$$

where $\begin{pmatrix} {}^{-1}H_E & {}^{-1}H_P \\ 0 & \mathcal{I} \end{pmatrix}$ is this basis.

If we recursively define

$$\begin{array}{rcl} {}^kH_E & := & H_k \cdot {}^{k-1} H_E \\ {}^kH_P & := & H_k \cdot {}^{k-1} H_P + H_k^p \\ {}^kH_R & := & H_k \cdot {}^{k-1} H_R + h_k, \end{array} \tag{9.8}$$

we are able to compute the condensed system.

We initialize ($l = \{1, 3\}$)

$$\begin{array}{rcl} \hat{D}_{l2}^0 & := & D_l^0 \cdot {}^{-1} H_E \\ \hat{P}_l^0 & := & D_l^0 \cdot {}^{-1} H_P + {}^0 D_l^p \\ \hat{d}_l^0 & := & D_l^0 \cdot {}^{-1} H_R + \hat{r}_l^0 \end{array} \tag{9.9}$$

and calculate

$$\begin{array}{rcl} \hat{D}_{l2}^i & := & \hat{D}_{l2}^{i-1} + D_l^i \cdot {}^{i-1} H_E \\ \hat{P}_l^i & := & \hat{P}_l^{i-1} + D_l^i \cdot {}^{i-1} H_P + {}^o D_l^p \\ \hat{d}_l^i & := & \hat{d}_l^{i-1} + D_l^i \cdot {}^{i-1} H_R + \hat{r}_l^i. \end{array} \tag{9.10}$$

By this, we get

$$\begin{array}{rcl} \tilde{D}_{l2} & = & \hat{E}_{l2}^N \\ \tilde{P}_l & = & \hat{P}_l^N \\ \tilde{d}_l & = & \hat{d}_l^N \end{array} \tag{9.11}$$

as the condensed system.

After solving the condensed system, we are able to compute the increments. A fast version uses stored matrices ${}^kH_\alpha(\alpha = E, P, R)$

$$\Delta s_j =^{j-1} H_E \Delta s_{02} +^{j-1} H_P \Delta p +^{j-1} H_R. \tag{9.12}$$

Further information and a derivation of the formulas can be found in [Sch88].

An important difference to the other methods shown in this chapter is the efficient mix of derivative generation and condensing saving computational work, but restricting this method to the class of serial algorithms.

**Remark 5 (Using all constraints)**
*To reduce the number of directional derivatives needed, it is possible to use all the constraints, not only the constraints w.r.t. initial values of the first multiple shooting interval. At the time–point a constraint depends only on this and previous time–points, a corresponding linear combination of increments is defined and can be formally eliminated with the same approach as shown above.*

## 9.2 Invariant Based Reduction Method

The methods in this section examine BVP–constraints originating from invariants of the mechanical system. Eliminating the need to generate derivative information in directions already specified by invariants, we solve the VDE w.r.t. a number of directions equal to the degrees of freedom of the underlying model. Especially in the case of Natural Coordinates with many redundant variables, this is of highest importance. We want to remind that algebraic variables and algebraic constraints are *not* used in the optimization context.

To specify the interesting directions, we provide a *unique* decomposition of the tangent space of the full Euclidean vector space. The first part is a basis of the tangent space of the manifold that fulfils the invariants, and the second part is a basis of the orthogonal space and is given by the derivatives of the invariants. For simplicity reasons we omit a possible dependency of the constraints w.r.t. parameters and non–holonomic constraints in this presentation. A generalization is easily done

$$\hat{G}(p,v) = \begin{pmatrix} \frac{\partial g_{pos}(p)}{\partial p} & 0 \\ \frac{\partial g_{vel}(p,v)}{\partial p} & \frac{\partial g_{vel}(p,v)}{\partial v} \end{pmatrix}. \tag{9.13}$$

The basis of the tangent space $S(p,v)$ is fully orthogonal to this subspace, such that

$$B(p,v) := \begin{pmatrix} S(p,v) \\ \hat{G}(p,v) \end{pmatrix} \tag{9.14}$$

has full rank and the scalar products $S_i(p,v)\hat{G}_j^T(p,v)$ and $\hat{G}_j^T(p,v)S_i(p,v)$ are zero, where $S_i(p,v), \hat{G}(p,v)_i$ are rows of the matrices and a basis of the Euclidean vector space at $(p,v)$. A computation of $S_i$ is given in section 8.6.

To give a separation of the Jacobian of the parameter estimation problem into derivatives in the manifold and orthogonal to it, we transform the linear system with block matrices of the form $B_i^T := B(s_i)^T$ from the right–hand side.

$$\begin{pmatrix} D_0 B_0^T & D_1 B_1^T & \dots & D_N B_N^T & D^p \\ H_0 B_0^T & -B_1^T & & & H_0^p \\ & & \ddots & & \vdots \\ & & H_{N-1} B_{N-1}^T & -B_N^T & H_{N-1}^p \end{pmatrix} \begin{pmatrix} B_0^{-T}\Delta s_0 \\ B_1^{-T}\Delta s_1 \\ \vdots \\ B_N^{-T}\Delta s_N \\ \Delta p \end{pmatrix}; \begin{pmatrix} l \\ h_0 \\ \vdots \\ h_N \end{pmatrix}$$

Looking at the Jacobian, we have two sets of directional derivatives. Derivatives in direction of the manifold $(H_i S_i^T), (D_i S_i^T)$ are fully specified by the dynamics of the underlying differential equations. The other directions perpendicular to the manifold $(H_i \hat{G}(s_i)^T), (D_i \hat{G}(s_i)^T)$ are implicitly given by the relaxation technique as described in the last section.

Using the limit form of non–stiff Baumgarte relaxation, we get sensitivity information w.r.t. the matching conditions

$$H_i \begin{pmatrix} S_i \\ \hat{G}_i \end{pmatrix}^T \equiv \begin{pmatrix} H_i S_i^T & 0 \end{pmatrix} =: \left( \hat{H}_i 0 \right) \tag{9.15}$$

$\hat{H}_i$ is calculated directly using directional derivatives and matching initial directions.

In the case of least square conditions or constraints, we get

$$D_i \begin{pmatrix} S_i \\ \hat{G}_i \end{pmatrix}^T \equiv \begin{pmatrix} D_i S_i^T & D_i \hat{G}_i^T \end{pmatrix} =: \left( \hat{D}_i \hat{D}_i^G \right). \tag{9.16}$$

If we assume that we first apply the projection of the limit form of Non–Stiff Baumgarte relaxation, we have $\hat{D}_i^G = 0$.

To summarize, we only need to compute the solution of the VDE w.r.t. a reduced set of directions. These are specified by a set of initial directions $S_i$. This procedure is equivalent to multiplying the complete sensitivity matrix $H_i$ with $B_i$ from the left–hand side. In the following two sub–sections, we present two different methods to apply the condensing algorithm based on this efficient generation of the derivative information.

## 9.2.1 Invariant–Based Reduction Method I

The first method reverses this transformation after the sensitivity has been calculated w.r.t. a reduced set of directions.

$$\bar{H}_i = \left( \hat{H}_i 0 \right) \begin{pmatrix} S_i \\ \hat{G}_i \end{pmatrix}^{-T} \tag{9.17}$$

$$\bar{D}_i = \left( \hat{D}_i 0 \right) \begin{pmatrix} S_i \\ \hat{G}_i \end{pmatrix}^{-T} \tag{9.18}$$

We get singular sensitivity matrices $\bar{H}_i$ and $\bar{D}_i$, but we want to stress that this singularity does not transport itself to the Jacobian of the optimization problem because of the term $-\mathcal{I}$ in each matching conditions block–row, except for the first multiple shooting node. At the first node we formulate BVP–constraints which guarantee consistent initial value of the underlying IVP in the solution and by the continuation condition of the complete trajectory

$$\begin{aligned} g_{pos}(s_0) &= 0 \\ g_{vel}(s_0) &= 0. \end{aligned} \tag{9.19}$$

The resulting system is of the same form and dimension as the one in standard formulation and can be treated with the same algorithms.

This method was developed by the author and first implemented by Stossmeister.

### 9.2.2 Invariant–Based Reduction Method II

While method I uses the invariants to reduce the computational effort of solving the VDE, method II additionally exploit the structure to reduce the effort of the condensing algorithm.

Although we know by experience, that the computational effort of the condensing algorithm is negligible, this might not be the case in our context. This is due to the high dimension originating from the complexity of the (bio–mechanical) model and the use of Natural Coordinates, which introduce 24 variables per body. Furthermore, we saw that the generation of derivative information is quadratic w.r.t. the number of bodies[1], while the decomposition is of $\mathcal{O}(n^3)$. Therefore, there is a size of the model at which the decomposition algorithm is more expensive. We estimate that this break even point is at approximately several hundred bodies. Although, we do treat models with a smaller number of bodies, it is useful to keep future developments in mind.

This huge amount of work during the condensing algorithm can be avoided or at least postponed to higher dimensional models by eliminating variables in a similar way to section 9.1. This is achieved by transforming the system from the left–hand side in addition to the transformation from the right–hand side.

We want to take a first look at the linearized system.

$$\begin{pmatrix} D_0 & D_1 & \dots & D_N & D^p \\ G_0 & & & & G_0^p \\ H_0 & -\mathcal{I} & & & H_0^p \\ & & \ddots & & \vdots \\ & & H_{N-1} & -\mathcal{I} & H_{N-1}^p \end{pmatrix} \begin{pmatrix} \Delta s_0 \\ \Delta s_1 \\ \vdots \\ \Delta s_N \\ \Delta p \end{pmatrix} ; \begin{pmatrix} l \\ g_0 \\ h_0 \\ \vdots \\ h_{N-1} \end{pmatrix}, \tag{9.20}$$

The transformed system is

$$\begin{pmatrix} \tilde{D}_0 & \tilde{D}_1 & \dots & \tilde{D}_N & D^p \\ \tilde{\mathcal{G}}_0 & & & & \tilde{\mathcal{G}}_0^p \\ \tilde{H}_0 & -\tilde{\mathcal{I}}_1 & & & \tilde{H}_0^p \\ & & \ddots & & \vdots \\ & & \tilde{H}_{N-1} & -\tilde{\mathcal{I}}_N & \tilde{H}_{N-1}^p \end{pmatrix} \begin{pmatrix} \Delta \tilde{s}_0 \\ \Delta \tilde{s}_1 \\ \vdots \\ \Delta \tilde{s}_N \\ \Delta p \end{pmatrix} ; \begin{pmatrix} l \\ \tilde{g}_0 \\ \tilde{h}_0 \\ \vdots \\ \tilde{h}_{N-1} \end{pmatrix}, \tag{9.21}$$

[1]For this assessment we assume that the number of degrees of freedom is approximately proportional to the number of bodies. This is the typical case except for models with a high number of closed kinematic loops.

where we defined

$$\begin{array}{rcl}
\tilde{g}_0 & := & \begin{pmatrix} g_{pos}(s_0) \\ g_{vel}(s_0) \end{pmatrix} \\
\tilde{h}_i & := & B_i^{-T} h_i \\
\tilde{D}_i & := & D_i B_i^T \\
\tilde{D}_i^p & := & D_i^p \\
\tilde{G}_0 & := & G_0 B_0^T \\
\tilde{G}_0^p & := & G_0^p
\end{array}$$

$$\begin{array}{rcll}
\tilde{H}_i & := & B_{i+1}^{-T} H_i B_i^T & \\
\tilde{H}_i^p & := & B_{i+1}^{-T} H_i^p & \\
\tilde{\mathcal{I}}_i & := & B_i^{-T} \mathcal{I} B_i^T & = \mathcal{I} \\
 & & & \\
\Delta\tilde{s}_i & := & B_i^{-T} \Delta s_i . &
\end{array}$$

Examining $B^{-T}$, we see

$$\begin{array}{rcl}
\begin{pmatrix} S^{T+} \\ \hat{G}^{T+} \end{pmatrix} \begin{pmatrix} S^T & \hat{G}^T \end{pmatrix} & = & \begin{pmatrix} S^{T+}S^T & S^{T+}\hat{G}^T \\ \hat{G}^{T+}S^T & \hat{G}^{T+}\hat{G}^T \end{pmatrix} \\
 & = & \begin{pmatrix} (SS^T)^{-1}SS^T & (SS^T)^{-1}S\hat{G}^T \\ (\hat{G}\hat{G}^T)^{-1}\hat{G}S^T & (\hat{G}\hat{G}^T)^{-1}\hat{G}\hat{G}^T \end{pmatrix} \\
 & = & \begin{pmatrix} \mathcal{I} & \mathcal{O} \\ \mathcal{O} & \mathcal{I} \end{pmatrix},
\end{array} \tag{9.22}$$

because $\hat{G}S^T = 0$ and $S\hat{G}^T = 0$ as of the construction of $S$, and therefore we get a better suited representation of the the left–hand side inverse $B^{-T}$.

Using this, we write

$$\tilde{H}_i = B_{i+1}^{-T} H_i B_i^T = \begin{pmatrix} S_{i+1}^{+T} \\ \hat{G}_{i+1}^{+T} \end{pmatrix} \begin{pmatrix} \hat{H}_i & 0 \end{pmatrix} = \begin{pmatrix} S_{i+1}^{+T}\hat{H}_i & 0 \\ \hat{G}_{i+1}^{+T}\hat{H}_i & 0 \end{pmatrix} =: \begin{pmatrix} \hat{H}_i^S & 0 \\ \hat{H}_i^G & 0 \end{pmatrix} \tag{9.23}$$

with

$$\hat{H}_i^G = \hat{G}_{i+1}\hat{H}_i \simeq 0 \tag{9.24}$$

and zero in the solution. A detailed discussion of this term is shown at the end of this section. It is possible to give an equivalent form for derivatives w.r.t. parameters.

According to equation (9.23), we write the increment as

$$\begin{array}{rcl}
B_i^{-1}\Delta s_i & = & B_i^{+T}\Delta s_i \\
 & = & \begin{pmatrix} S_i^{+T}\Delta s_i \\ \hat{G}_i^{+T}\Delta s_i \end{pmatrix} \\
 & =: & \begin{pmatrix} \Delta\tilde{s}_i^S \\ \Delta\tilde{s}_i^G \end{pmatrix}
\end{array} \tag{9.25}$$

and the matching conditions as

$$\begin{aligned} B_i^{-1} h_i &= B_i^{+T} h_i \\ &= \begin{pmatrix} S_i^{+T} h_i \\ \hat{G}_i^{+T} h_i \end{pmatrix} \\ &=: \begin{pmatrix} \tilde{h}_i^S \\ \tilde{h}_i^G \end{pmatrix} \end{aligned} \tag{9.26}$$

This results in two decoupled systems if one neglects $\hat{H}_i^G$. One system corresponds to increments *in* the manifold

$$\begin{pmatrix} \hat{D}_0 & \hat{D}_1 & \dots & \hat{D}_N & D^p \\ \hat{H}_0^S & -\mathcal{I} & & & \hat{H}_0^{S,p} \\ & \ddots & \ddots & & \vdots \\ & & \hat{H}_{N-1}^S & -\mathcal{I} & \hat{H}_{N-1}^{S,p} \end{pmatrix} \begin{pmatrix} \Delta\tilde{s}_0^S \\ \Delta\tilde{s}_1^S \\ \vdots \\ \Delta\tilde{s}_N^S \\ \Delta p \end{pmatrix} ; \begin{pmatrix} l \\ \tilde{h}_0^S \\ \vdots \\ \tilde{h}_{N-1}^S \end{pmatrix} . \tag{9.27}$$

We have reobtained a system in the *same* form but a *reduced* dimension.

The other system corresponds to increments *perpendicular* to the manifold[2]:

$$\begin{pmatrix} -\mathcal{I} & & & \\ 0 & -\mathcal{I} & & \\ & \ddots & \ddots & \\ & & 0 & -\mathcal{I} \end{pmatrix} \begin{pmatrix} \Delta\tilde{s}_0^G \\ \Delta\tilde{s}_1^G \\ \vdots \\ \Delta\tilde{s}_N^G \end{pmatrix} ; \begin{pmatrix} -\tilde{g}_0 \\ \tilde{h}_0^G \\ \vdots \\ \tilde{h}_{N-1}^G \end{pmatrix} . \tag{9.28}$$

Every term except the first reads like

$$\hat{G}_i^{+T} \Delta\tilde{s}_i^G = \hat{G}_i^{+T} h_i$$

or

$$\hat{G}_i \Delta\tilde{s}_i^G = \hat{G}_i h_i.$$

This is equal to one linearized projection step to the consistent manifold using the pseudo inverse $\hat{G}_i^{+T}$. The increment of the first multiple shooting node can be re–formulated to an equal form.

As a result, we get a modified system of much lower dimension as in reduction method I. Furthermore, the decomposition and solution of the second system can be done in parallel, too.

At last we examine if to neglect

$$\hat{H}_i^G = \hat{G}_{i+1} \hat{H}_i = \hat{G}_{i+1} H_i S_i^T \tag{9.29}$$

is a valid approximation.

[2]Here shown, if the manifold is not parameterized, i.e. the constraints do not depend on the parameters

The columns of $S_i$ are a basis of the tangent space of the consistent manifold at $s_i$. As we integrate along the consistent manifold, $H_i S_i^T$ is a basis of the tangent space at $x_i := y(t_{i+1})$, i.e. the state at the end of the integration interval. As we have a non–linear manifold, this tangent space is not equal to the tangent space at $s_{i+1}$.

As we use a unique and continuous generation of the basis in the following sense

$$\lim_{(x-y)\to 0} S_x - S_y = 0, \tag{9.30}$$

where $S_x$, $S_y$ are the basis of the tangent space at states $x$ and $y$, the tangent spaces are equal in the solution of the parameter estimation problem, where the matching conditions are fulfilled.

To estimate the error of neglecting $\hat{H}^G$ we compute

$$\begin{aligned}\tilde{H}_i &= B_{i+1} H_i B_i^T = \begin{pmatrix} S_{i+1} \\ \hat{G}_{i+1} \end{pmatrix} \begin{pmatrix} \hat{H}_i & 0 \end{pmatrix} =: \begin{pmatrix} \hat{H}_i^S & 0 \\ \hat{H}_i^G & 0 \end{pmatrix} \\ &= \begin{pmatrix} S_{i+1} \\ \hat{G}_{i+1} \end{pmatrix} \begin{pmatrix} S_{x_i} \\ \hat{G}_{x_i} \end{pmatrix}^{-1} \begin{pmatrix} S_{x_i} \\ \hat{G}_{x_i} \end{pmatrix} \begin{pmatrix} \hat{H}_i & 0 \end{pmatrix} \\ &= T_{x_i}^{s_{i+1}} \begin{pmatrix} S_{x_i} H S_{s_i} & 0 \\ 0 & 0 \end{pmatrix},\end{aligned} \tag{9.31}$$

where $T_{x_i}^{s_{i+1}}$ is the transformation between the tangent space at $x(\tau_{i+1}; s_i)$ and $s_{i+1}$. If the consistent manifold would be an Euclidean space, the transformation matrix $T_{x_i}^{s_{i+1}}$ would the unity matrix. According to previous considerations, in the non–linear case there is an additional term proportional to the curvature of the manifold and the difference of the two points, i.e. the matching conditions. For this reason, the approximation leads to an error of the same quantity.

The nonlinearity of the manifold is fixed. Therefore, to bound the error introduced by this approximation, the norm of the matching conditions $h_i$ has to be bounded. These values correspond to the gap between the state at the end of one multiple shooting node and the initial state of the next node. Refining the multiple shooting discretization, i.e. distributing more multiple shooting nodes to the same integration interval, will decrease the norm of each matching condition, while keeping the norm of all norms of the matching conditions almost equal.

Due to this, the error of neglecting $\hat{H}_i^G$ can be bounded by choosing a corresponding granularity of the multiple shooting grid and suitable initial guesses for the states at the multiple shooting nodes. Therefore, it is valid to approximate the Jacobian by neglecting $\hat{H}_i^G$.

## 9.3 Theoretical Comparison of the Reduction Methods

We now present a theoretical comparison of the three methods described in previous sections. A numerical comparison will be shown in part V.

The Fixfit–approach is especially useful if all (or at least most) initial dynamic states are fixed. Then, it is only necessary to compute directional derivatives w.r.t. the parameters and one additional direction corresponding to the violation of the matching condition. This number of directions is much smaller than the number of degrees of freedom of mechanical models, especially in the case of biomechanical systems, where the motion of the bodies is less restricted by joints than in the case of technical mechanics. In this case, some joints typically restrict a less number of degrees of freedom and some bodies are only connected to other bodies via forces. A special example for the latter is the modelling of wobbling masses.

On the one hand, the reduction methods based on invariants only reduce to the dimension of the degrees of freedom. On the other hand, they use constraints which are formulated at each multiple shooting node. Therefore, a parallelization w.r.t. the multiple shooting discretization is possible.

Furthermore, we want to look at the different types of relaxation which can be applied to the different reduction methods. Fixfit and the first reduction method based on invariants are not restricted, but this is not the case in the second reduction method, which implicitly formulates the projection by transforming one part of the sensitivity matrix of the matching conditions

$$\hat{H}_i^S = S_{i+1}^T H_i S_i^T.$$

While applying the condensing algorithm one has to compute terms of the form

$$S_{i+1}^T S_{i+1}^T H_i. \tag{9.32}$$

The term $S_{i+1}^T S_{i+1}^T$ corresponds to a projection that is equivalent only to an Euclidean projection as shown in the following.

The Augmented system of the linearized form of the Euclidean projection for initial directions is

$$\begin{pmatrix} \mathcal{I} & \hat{G}^T \\ \hat{G} & \end{pmatrix} \begin{pmatrix} d \\ \lambda \end{pmatrix} = \begin{pmatrix} d_0 \\ 0 \end{pmatrix}, \tag{9.33}$$

with

$$\hat{G} = \begin{pmatrix} G & \\ G_v & G \end{pmatrix}, \tag{9.34}$$

and initial respectively projected directions $d_0$ and $d$. According to this equation, the projected direction is

$$d = \left\{ \mathcal{I} - \hat{G}^T \left( \hat{G}\hat{G}^T \right)^{-1} \hat{G} \right\} d_0. \tag{9.35}$$

Because

$$\begin{aligned} \mathcal{I} &= \begin{pmatrix} S^T & \hat{G}^T \end{pmatrix} \begin{pmatrix} S^{T+} \\ \hat{G}^{T+} \end{pmatrix} \\ &= S^T S^{T+} + \hat{G}^T \hat{G}^{T+} \\ &= S^T \left( S S^T \right)^{-1} S + \hat{G}^T \left( \hat{G}\hat{G}^T \right)^{-1} \hat{G}, \end{aligned} \tag{9.36}$$

we reformulate (9.35) to

$$d = S^T S^{T+} d_0,$$

which proves the connection to equation (9.32). As there does not exist a comparable simple reformulation of the projection adapted to mechanical systems, we are not able to use this type of projection for the second invariant–based reduction method. Because will see that this projection method has the best properties, this is a drawback of the second method. In part V, we study the severity of this drawback by means of numerical experiments.

All methods need initial directions for the variational differential equation at each multiple shooting interval, but provide different approaches to calculate these directions. For the following considerations we restrict ourselves to initial direction w.r.t. states, fixed initial values to simplify the formulation w.r.t. the Fixfit–approach and do not consider the first multiple shooting interval.

Fixfit–approach uses the directions at the end of the previous multiple shooting interval added to the matching condition: ${}^kH_R = H_k \cdot^{k-1} H_R + h_k$ (see equation 9.8). As the manifold is non–linear, ${}^kH_R$ is not consistent at $s_k$ and will be projected due to the limit form of non–stiff Baumgarte relaxation at the start of the integration of the next multiple shooting interval. In the case of the invariant–based methods, we calculate the initial directions based on the tangent space of the manifold at the multiple shooting node with basis $S_k$ which is consistent w.r.t. the manifold.

As the solution trajectory is continuous and, therefore, $H_k S_{k-1} = S_k$ in the solution, this effect does not effect the solution. As we assume a suitable multiple shooting discretization as discussed at the end of the previous section, we can assume that for typical situations, the difference of the two methods does not effect the convergence properties of the GGN significantly. This assumption is approved by numerical tests as shown in part V; no significant advantage of one of the two approaches is seen.

# Part IV

# Software Engineering

The first three parts of this work study the theoretical aspects of the numerical algorithms that are used to solve parameter estimation problems applied to (bio–) mechanical models. This part studies the practical aspect, i.e. the implementation, of these algorithms and approaches. Mechanical systems are ”models that consist of bodies that interact with each other and the environment by joints and forces”. Therefore, it is natural to think in terms of objects which implement these bodies, joints and forces and, therefore, use an object–oriented approach. We think that object–oriented programming (OOP) is a suitable software design also for the parameter estimation part, because it is able to treat the complexity originating from various options and extensions one has to consider when implementing all methods presented in the numerical part of this work. As not all possible options are considered in this work, it is even more important to provide a flexible, maintainable and especially extendable program. These are exactly the difficulties object–oriented programming tries to treat.

We first present a brief introduction to OOP by discussing the three major paradigms: encapsulation, inheritance and polymorphism. By exploiting these paradigms, OOP promises the advantages cited above. Afterwards, we discuss the prejudice of a lack of performance of OOP compared to procedural approaches and refer to guidelines which help to avoid this disadvantage. Furthermore, we focus on problems of OOP as discussed in the middle of the 90’s. Due to the increase of design options and the increase of interactivity between different parts of the program, the complexity of the program increases. This leads to a *decrease* of flexibility, usability and maintainability because individuals or groups of humans are no longer able to treat the complexity. This is the opposite of what OOP promises. Moreover, this disadvantage gets worse with an increasing size of the program. Considering the size of the implementation presented in this work, some 80’000 lines of source–code, this would be a major drawback.

In the second chapter of this part, we discuss and apply various methods invented to avoid this problem. We first provide a theoretical description and then apply the methods directly to one or both of the two major packages that form this work: Parfit++ (PARameter FIT using C++), which implements the parameter estimation tool as described in part I, and MBSNAT (MultiBody System simulation based on NATural coordinates), which provides the implementation of the (bio–) mechanical model as described in part II. The theory of part III is represented in both packages. All these methods successfully try to increase the structure, and thus reduce the design possibilities and complexity of the package, by splitting it into more manageable parts.

We first discuss the three–tier model, which separates the presentation layer and database access from the main computational task, i.e. the numerical algorithms. This has the additional effect that parts of the program which are highly dependent on the hard– and software environment are separated from the main computational task.

The other way round, the core of the program, i.e. the numerical algorithms, is still implemented in one part. Therefore, we introduce additional structure originating from levelization or hierarchy. In OOP we have to distinguish object–hierarchy, like a model consisting of bodies, and class–hierarchy, like spring and damper elements inheriting from a general force–element. Both hierarchies can and must be applied such that every feature is located exactly in one particular part of the program: the dynamic properties of a body are distinguished from that of a force (object–hierarchy) and from the methods of the body used during the decomposition of the system (class–hierarchy). This approach is well suited for the package MBSNAT.

A further approach recognizes the problem of physical and logical coupling between different parts of the program. Whereas the latter method tries to reduce the complexity of the program by organizing the location of one feature, it does nothing to reduce the coupling. To avoid this, each class inherits from one or more interfaces, i.e. classes which only provide a minimum of information. Furthermore, different classes communicate only by such lightweight interfaces. Thus, the classes are not coupled directly, i.e. strongly, knowing the complete definition of the other class. A drawback of this approach is the increase of indirection, which slightly decreases the performance, originating from the use of interfaces. Therefore, it is suitable to strongly couple parts of the program, which heavily communicate with each other, and only use light coupling otherwise. This results in the so–called interface– or component–oriented programming, which decomposes the whole program in smaller component, which are strongly coupled internally and loosely coupled to other components. This approach is used in Parfit++. Due to the decrease of performance in the case of MBSNAT, which is the inner loop of the whole package, we implement it as one component.

Therefore, we avoid the drastic increase of complexity originating from a naive application of OOP by using a combination of all three approaches, while the special demands of Parfit++ and MBSNAT are considered by the question how strongly we apply them. Thus we get a highly structured and efficient set of components which, in different combinations, amount to either Parfit++ or MBSNAT. Efficiency in this context is achieved in terms of speed *and* flexibility, maintainability and usability.

# Chapter 10

# Object–Oriented Programming

Object–oriented programming (OOP) provides developers with a powerful concept to cope with the growing complexity of today's software. Mostly because of the high flexibility and extensibility of this concept, OOP is widely accepted, which includes the community of numerical mathematics in recent years. This is due to the fact that especially programming in C++ has reached such a mature state that one obtains the speed of Fortran 77 while retaining the good properties of the object–oriented concept. In order to be competitive with Fortran 77 in terms of speed, we have to keep in mind several guidelines that will be discussed briefly in section 10.1 together with a short introduction into the basic concepts of OOP.

Unfortunately, this richness of features of C++ can lead to incomprehensible and overly complex programs. This typically is not a problem with small software projects, but with programs growing in size it can lead to unbearable difficulties. In order to prevent this, several restrictions have to be accepted, which inhibits an uncontrolled growth of software. During the first years, several solutions were developed [Lak96], [Dat97], concluding in the industry–standard solution: *Component–Oriented Programming* [Gri98]. This is a concept to further improve structure on a higher level. The paradigm is one of the basics of (D)COM(+) [Box98], Corba [Bor01],[Lin98] or Enterprize Java Beans (EJB) [Ste98]. Although these concepts can be used to support distributed computing, we will apply a subset on this stand–alone project. We supply our own implementation in order to retain an easy support of different programming environments. Furthermore, in chapter 11 we will take a closer look at how this is implemented .

## 10.1 Introduction to Object–Oriented Programming

In order to understand efficient object–oriented programming, one first has to know the paradigms. The fundamental abstraction unit is the *class*, which corresponds to some (abstract) physical object such as a body, or some more abstract numerical entity like a matrix or an iteration scheme. An instance of a class is called an *object*. The class is a collection of *methods*, that is a set of functions providing actions the class *can do*, and *member variables* which represent the *properties* or *states* of the class. A special form of class which only defines methods, but does not provide any implementation or member variables is called an *interface*. In procedural languages, data move ”through” a series of procedures, which results in the classical calling–tree. In contrast to this, data are fixed to one object in object–oriented languages, and operations on data are done by requesting them from the corresponding object, which then administers its own data. Therefore, the communication structure is more flexible.

The philosophy of OOP is best outlined by three paradigms associated with it – encapsulation, inheritance and polymorphism – which are described in the following.

### 10.1.1 Encapsulation

The paradigm *encapsulation* means that the internal design is hidden from the user of this class, and only the minimum of necessary information is presented to him. In practice, this is equivalent to only knowing the class declaration that defines the *public* interface to the rest of the program, and not the implementation. As a result, the user knows *what* the class does but not *how*. Furthermore, data are typically accessible by *get-* and *set-* methods so that the internal data–structure is hidden from the user. This allows for additional code to implement dependencies between data and to guarantee a valid internal state of the object. E.g. when changing the dimension of a vector–object, the variable storing the dimension and the internal array holding the data are both changed. This results in a less complicated and more flexible usage. Furthermore, one gains the possibility of modifying the internal data structure without a need to change the rest of the program. Part of this *information hiding* is already applied in procedural languages, but object–oriented languages realize this benefit in a more flexible way and better support this paradigm.

### 10.1.2 Inheritance

The paradigm of *inheritance* is one of the most powerful and well known concepts in object–oriented programming. When thinking of animals, it is possible to define a tree classifying them. So, for example, mammals and birds are animals while bats and horses are mammals (figure 10.1).

If one implements this example, it is possible to consider the similarities and distinctions between these types. Inheritance allows for *reusing* code – not just copying it – if the behavior is equal, while it is possible to *overwrite* it if the behavior is different. In the above example, one feature of mammals is that they typically use four–legged movement. As a result, it is suitable to implement a method MOVE() in the class MAMMAL based on four–legged movement. As horses use this type of movement, we can reuse this code, which is not possible for bats because their type of movement is different. Therefore, we have to override the corresponding implementation in this case. This is strongly connected to encapsulation

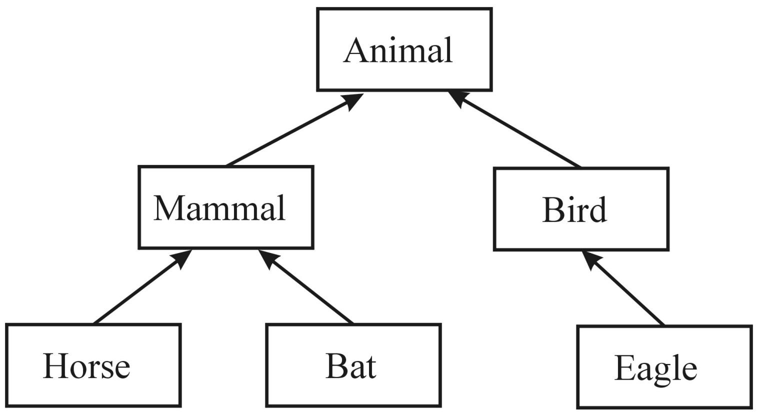


Figure 10.1: Inheritance Tree

because a bat still implements the method MOVE(), which still results in movement, but uses a different implementation. The exact internal detail is hidden.

Another example for the use of inheritance is the implementation of spring and damper elements. Each one is a specialization of a general force element, and they share much of their principle behavior, like connecting to bodies and applying some force. They are distinguished by the force law, though.

A further improvement of this concept is *multiple inheritance*. Looking at bats and birds, you can see that they are both animals and have the same principle behavior. But the structure we used until now is not able to represent the additional similarity which is shared by these two types, namely that they are able to fly. It is desirable to represent this connection, based on the same reasons to get inheritance at all. Therefore, we introduce another "base–class" (CANFLY) to provide it (figure 10.2).

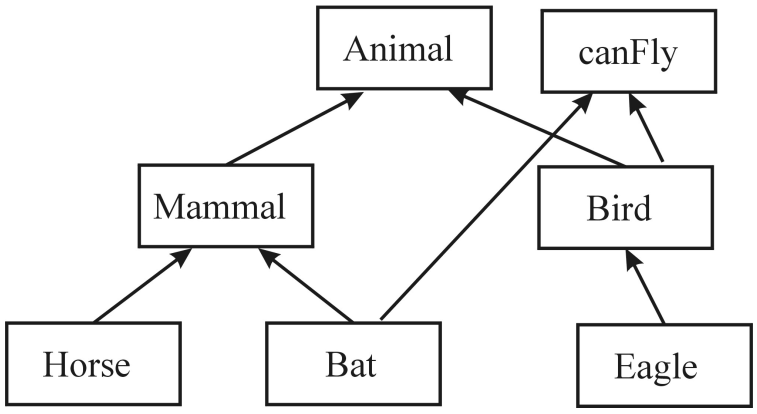


Figure 10.2: Multiple Inheritance

As a result, the bird and the bat inherit methods of flyer like FLYHIGHER().

By using multiple inheritance, it is possible to easily extend the functionality of a class by adding further interfaces, which improves the flexibility and reusability of a class. Furthermore, a class is not a *black box*, but one is able to interact in a very adapted way.

### 10.1.3 Polymorphism

The last paradigm we want to mention is *polymorphism*. As already said in the context of inheritance, we typically have class–trees implementing similar behavior, e.g. that horses and eagles are both animals and as such inherit from the corresponding class. As a result of this, we are able to define a collection of animals and call a method, i.e. MOVE(), of *all* animals. We define a *heterogenous* collection of objects which we can principally treat in the same way, as long as we use their shared interface. If we consider the concept of multiple inheritance, we are able to cast to another base–interface at runtime (e.g. CANFLY) and use this functionality if it is supplied, again gaining much flexibility.

## 10.2 Complexity of OOP

Although the properties of OOP improve the flexibility, reusability and maintainability, one has to consider several possible limitations or difficulties when using an object–oriented language like C++. Because it extends the functionality of C, C++ is harder to learn and provides much more design possibilities. This impairs the usability of source code by others. Furthermore, the code may be more complex because separate parts of a code have more possibilities of interaction. We will see that if one does not introduce structure, and by that narrow the design options, one looses properties of OOP like reusability and maintainability because of the limit of complexity a human being or a group is able to handle. A direct drawback in terms of performance at compile time is the topic of section 10.2.2, and the design aspects are presented in chapter 11. We first want to take a closer look at a prejudice regarding OOP, which says that there is a lack of numerical efficiency.

### 10.2.1 Efficient Numerics

In the beginning of using the object–oriented paradigm, the software often lacks the performance of procedural code written, for example, in C or Fortran. As numeric applications are typically very time–consuming, this was a major drawback. But OOP in itself does not lead to excessively slow programs if one follows several rules. A thorough examination is done in [Yan01]; Here we do not want to go into details, but restrict ourselves to the most important topics:

- Machine–optimized BLAS- and LAPACK-libraries should be used for the most time–critical parts. Thus, one achieves upmost efficiency independent of the programming language.

- Arguments in calling functions, which are not of basic data type, should be passed by reference and not by value. The latter would lead to copying of internal data and may be to time–consuming memory allocation and deallocation.

- Constant qualifiers should be used wherever possible to allow for optimizations of the compiler.

- Inline functions avoid the offset of a function call – they are similar to macros but use a thorough type check – and should be used in very time–critical parts of the code.

- Virtual functions, which allow polymorphic behavior, should be avoided in time–critical parts, if they are not needed.

All these requirements can be met without impairing the functionality of OOP. The measures above lead to a performance of programs almost as good as in procedural languages. If one stresses this approach, which is done in programs like ATLAS [ATL], one is even faster. The disadvantage of programs like ATLAS is a less readable code. As a result, one has to balance between performance in terms of speed and performance in terms of flexibility, reusability and readability. As we will see later, the decision which method to use depends on the exact environment of a specific part of the code.

### 10.2.2 Complexity and Possible Consequences

Classes typically export several interfaces with several methods. Furthermore, the objects interact with each other in various ways, which leads to a more complex calling structure compared to the classical calling–tree of procedural programs. An object may access quite a number of other objects of different types. Therefore, the class must know the definition of several other classes.

This logical complexity of coupling between the classes results in physical complexity. A file containing references to various classes needs to include several header–files that contain the definition of these classes. If one includes the indirect connections – a class includes a header file, which again includes another header – this may lead to the inclusion of all the header files of a complete package. Because every class or a small group of classes defines a header file with quite a lot of entries, this may lead to a bottleneck at *compile time*.

Let us look at a test case done in [Lak96]. We define a number of arbitrary header files of fixed length. Furthermore, we supply an equal number of source files which include all the header files and, besides from that, are empty. This will lead to the following compile–times done on a SunSPARC 20 with 32MB:

| System Size | 100 lines headers | 1000 lines headers |
|---|---|---|
| 1 | 0.1s | 0.4s |
| 2 | 0.1s | 1.0s |
| 4 | 0.2s | 1.0s |
| 8 | 0.4s | 3.4s |
| 16 | 0.8s | 11.0s |
| 32 | 2.4s | 32.2s |
| 64 | 8.2s | 137.7s |
| 128 | 26.5s | 497.5s |
| 256 | 98.1s | > 1 day |
| 512 | 397.6s | |
| 1024 | > 1day | |

Considering the fact that the program package we are talking about in this work includes some 280 header files, this is at least an annoyance. This annoyance may lead to avoiding a modification of the code if this would result in a recompilation of the whole package, and as such may leads to inferior code. The solution of this problem is the same as in the case of design complexity: to impose additional structure to the software design.

# Chapter 11

# Structure of the Software Implemented in the Packages MBSNAT and Parfit++

As we have seen in the previous chapter, using an object–oriented approach in the first moment seems to lead to overly complex programs annihilating the advantages of this approach. In the case of the software packages Parfit++ (**PAR**ameter **FIT**ting based on C**++**) for parameter estimation and MBSNAT (**M**ulti **B**ody **S**ystems modelled using **NAT**ural coordinates) to model mechanical systems, which were developed in this work, we are talking about some 80'000 lines of code and additional Fortran libraries implemented in this work group. Therefore, it is of upmost importance to design the implementation of the software efficiently.

To solve the problem of too complex code, it is necessary to restrict the design possibilities and by that drastically increase the structure of the design. This will reduce the complexity to a manageable level.

In this chapter, we want to present three possibilities we used in the software–packages : the (three–)tier model, the hierarchy or levelization and the interface- or component–oriented programming. We will show that every option is suitable in different parts of the program, resulting in a well–designed structured software package.

## 11.1 Three–Tier Model

In modern software, the overall package is typically divided into several reasonable parts, also called tiers. In the case of two–tier models, this is the well–known client–server approach. In the three–tier model [Blo99], the program is broken up further.

| | |
|---|---|
| First Tier | Presentation (GUI) |
| Second Tier | Business Logic (Middleware) |
| Third Tier | Backend (Databases) |

Figure 11.1: Three Tier Model

### 11.1.1 First Tier

This tier represents the graphical user interface, including menus, dialogs and graphical output. This is highly dependent on the environment i.e. the operating system (like Windows or Unix), the framework (like for example MFC, OpenGL or Qt) or if the program is used via the web. This presentation tier typically includes very simple input/outpout checking, leaving the more complex ones to the second tier.

In the case of our program, we supply several packages which form this tier.

- In the case of MBSNAT, we provide an interactive graphical user interface based on Microsoft Foundation Classes (MFC). This interface allows for an interactive graphical build–up of the model based on mouse–clicks, menus or dialogs.
- We support 2D-graphics for diagrams. To adapt to different platforms, we need several different implementations based on file or Gnuplot output or online Matlab linkage based on ActiveX.
- It is possible to show a 3D–representation of the mechanical model based on OpenGL. This drastically increases the usability and debug–ability of the package.
- By using the picking and selection capabilities of OpenGL and additional dialogs we are able to interactively add, edit or remove arbitrary elements of the model.

As can be seen at the examples shown above, the implementation of this tier highly depends on the exact platform and software environment.

### 11.1.2 Second Tier

The second tier, which implements the middleware, deals with the actual computation or ”business–logic”. In the case of numerics, this level includes the complete actual numerical computation. In contrast to the first tier, this part typically does not depend on the environment. Therefore, it is suitable to include more complex routines concerning the user interface, too. A more detailed discussion of the middleware, which points out its significance, is done later on.

### 11.1.3 Third Tier

This third tier supplies access to the data, i.e. implements a connection to the database. In this case, several high–performance implementations are at disposition. Different realizations are suitable, depending on the amount of data and the environment. In the case of this package we do not expect a high number of input–data compared to other applications. Therefore, we choose the industry standard XML.

XML is based on human readable and structured ASCII text similar to HTML, and therefore is easy to learn and handle. XML allows for syntax definition and checking using DTD (document type declaration) or similar methods. It is also easily presented in a more graphical form using XSLT (eXtensible StyLesheet Transformation) and a common web browser. Furthermore, modern industry standard databases are able to supply the result of a request in XML, such that one is able to cope with an increasing number of input data if this is necessary.

Although this approach is very suitable, the main problem of reducing the complexity is not solved. The type of program we discuss here typically focuses on the second tier which is also represented in the amount of code. It is still by far too much, thus it is necessary to introduce additional structure.

## 11.2 Hierarchy or Levelization to Structure Software

A second possibility of introducing structure is to use a fixed hierarchy with certain levels of a fixed ordering. Because of inheritance and the focus on classes or objects, we must differentiate between object–hierarchies and class–hierarchies.

Class–hierarchies originate from the paradigm of inheritance. A class typically inherits from a super–class, or it is this super–class. This leads to a tree–structured system. The introduction of multiple inheritance destroys the tree–structure and may introduce loops, while being still hierarchical. We may introduce a numbering of the classes, such that each class belongs to one particular level.

The second hierarchy originates from the fact that some objects hold a collection of several other objects. A suitable example of this is the mechanical model which is defined by the bodies, forces and joints (see figure 11.2). The model contains them. This association between different objects is representable in a tree–structured system, which is called the object hierarchy.

It is possible to implement an association using both methods, but after this has been determined, these hierarchies are orthogonal to each other. In the example of figure (11.2) we see a model containing (by pointers) a body and a joint. This is the ”vertical” object–hierarchy. The ”horizontal” hierarchy is due to inheritance, where the class NMODEL inherits from DMODEL and the other classes accordingly. To increase the structure, it is important that the implementations done in level ”D” are of one category and different to level ”N”. The application of this segmentation throughout the program drastically increases the readability because

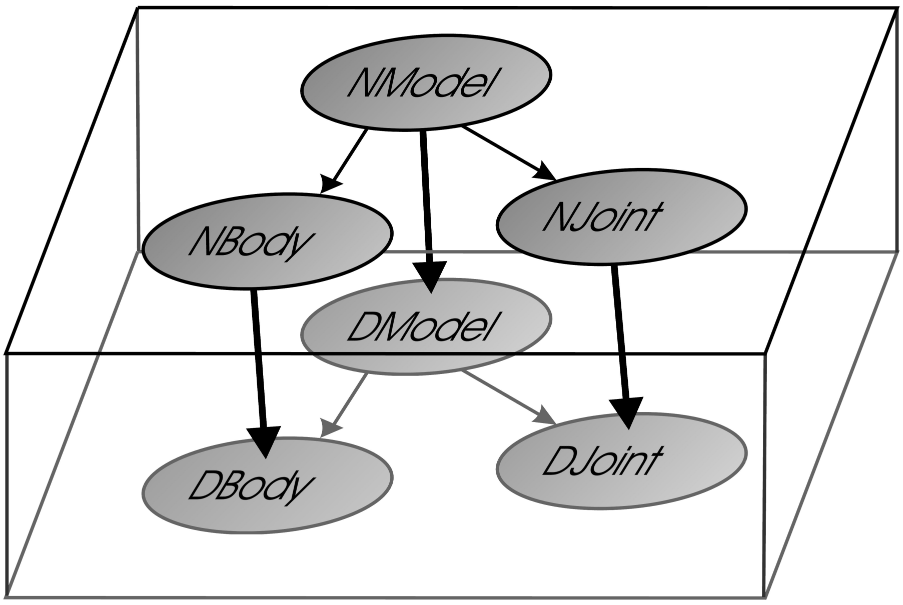


Figure 11.2: Object- and Class–Hierarchy

it is more easy to locate a specific code–segment. Adding functionality only leads to an extension of one level in the class–hierarchy or an additional element in the object hierarchy. This improves the flexibility of the overall program.

### 11.2.1 Object–Hierarchy of MBSNAT

The main object–hierarchy of MBSNAT is shown in figure (11.3). It represents the fact that the model is defined by a collection of bodies, joints and forces (and additionally events and parameters), while the joints themselves are defined using constraints.

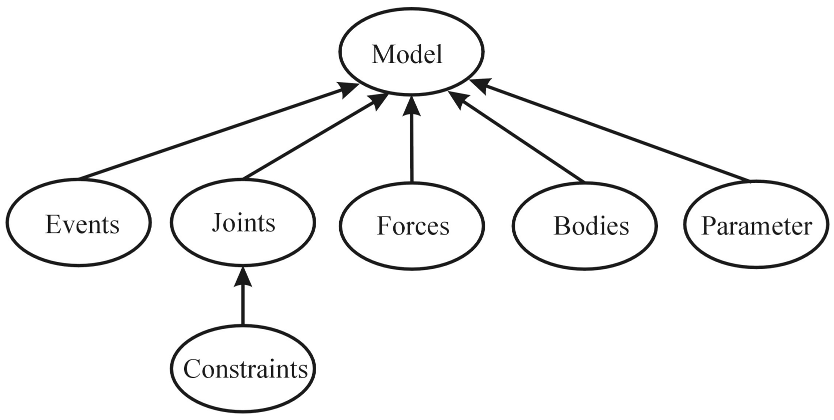


Figure 11.3: Object–Hierarchy of MBSNAT

In the software package, several other elements which represent discontinuities or parameters are included. Nevertheless, applying this approach leads to a better structure of the overall system and a better manageability of the code and the representation of the model in the computer.

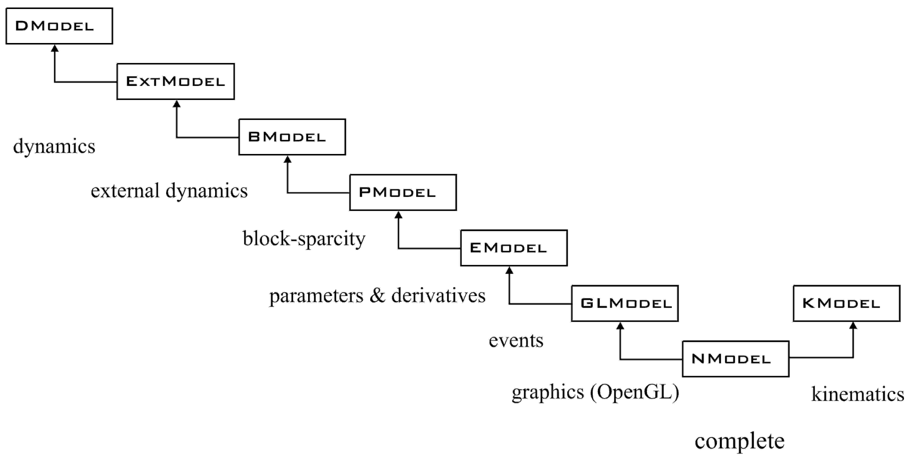


Figure 11.4: Class–Hierarchy for Model

### 11.2.2 Class–Hierarchy of MBSNAT

This hierarchy originates from the different features of a class, each level implementing one of these features. In the case of mechanical systems these are:

D: a dynamic description of the elements, like the mass of a body, contact points and force–law.

K: the same structure originating from bodies and joints seen from a kinematic point of view. We have implemented a fast way of kinematic movement based on relative coordinates.

EXT: additional handling of external dynamics, which is not part of the framework of mechanical DAE of index 3.

B: code implementing the block–sparse decomposition described in chapter 4.7.

P: parameters as elements that parameterize previously defined elements and code to automatically generate derivative information w.r.t. dynamic variables or these parameters.

E: events are elements that modify previously introduced elements to implement discontinuous behavior and to detect the exact time–point at which they have to be modified.

GL: extensions holding pointers to shapes, implemented in a component handling 3D–Graphics (see first tier; section 11.1.1).

N: the final level of the general implementation and the basis for specialized elements.

The last layer implements the complete shared functionality of classes of one type, be it a force, a joint or a parameter. Furthermore, the corresponding classes of level N are the base–classes for every specialized class of that type. The classes originating from that level are discussed further in appendix C, which shortly describes the elements implemented in MBSNAT.

Using both of these structures, it is easy to locate a certain functionality of an object by selecting the corresponding class–type, like forces, and the correct layer of the class–hierarchy this feature belongs to. Therefore, although we are working with a huge package, the code is still easily manageable, because we only have to search a rather small amount of code in every case. Furthermore, in the case of extending the functionality of the model, other parts of the package are modified as little as possible, because most of the new functionality is located in only a small specific part of the program.

## 11.3 Concepts of Interface–Oriented Programming

Looking at the previous section, we see that there has been introduced a form of structure that reduces the complexity of the design. Every logical entity corresponds to one part of the program, while familiar ideas are located nearby. Although this is good in terms of design, it does almost nothing to reduce the coupling between different parts.

When we look once more at figure (11.2), we see that NMODEL holds a pointer to NJOINT and as such ”knows” this class and all the base classes (e.g. DJOINT). The class NMODEL or, to be more exact, an instance of it, is *indirectly* coupled, physically and logically, to a huge portion of the overall program. This may lead to difficulties discussed in section 10.2.2 for increasing program size.

Due to this insight, the coupling is reduced if an object points to an instance of a base class – in this case DJOINT. Following this line of thoughts, the coupling is minimized by using the most basic class, which defines only the absolute minimum in order to access it. This minimum is provided by the definition of the methods, omitting any implementation of these methods and any member variables. This type of ”class” is fully abstract and therefore an *interface*. We define an interface for every feature of a class, such that every class inherits from a number of interfaces using multiple inheritance. Applying this concept to every class, we see that an object is only associated to another object via one (or more) of its interfaces. This drastically reduces the coupling between these parts of the program. These parts are *loosely coupled*, in contrast to *strong coupling* when pointing to the class directly or even inheriting from it. Furthermore, this loose coupling enhances the idea of information hiding, only providing the minimum of information defined in the interface while hiding internal details and extensions which are not needed in this context. In contrast to a black box approach, it is always possible to use the concept of dynamic casts to get a pointer to another aspect of an object which is defined by another interface. But there is also a drawback of loose coupling: only a subset of the complete functionality is given by one pointer, so that we must use virtual functions and the corresponding indirection. This leads to a slight performance decrease.

These interfaces can be used to further increase the logical structure, by grouping a set of objects to a so–called *component*[1]. Members of the same group are strongly coupled, that is they communicate directly in a fast way, while members of different components only know each other via interfaces, i.e. are loosely coupled.

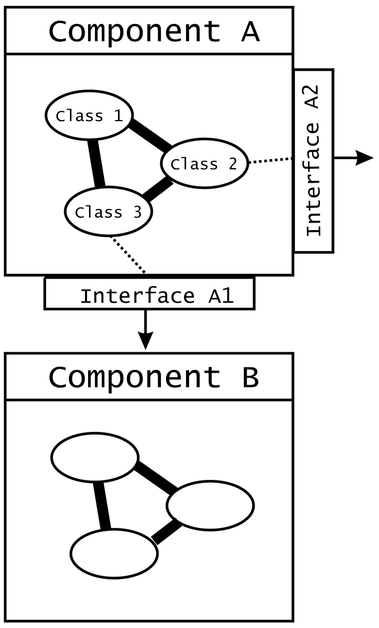


Figure 11.5: Strong and Loose Coupling

As a conclusion, interfaces are used to decouple different parts of a program. This gives the possibility to further structure a program, leading to components or a better structure inside such a component. This is valid in a logical sense, as the classes only need to know a couple of interfaces, as well as in a physical sense, because a file includes only header files containing interface definitions. This loose coupling is bought with a slightly decreased performance depending on the granularity of the decomposition. Usually, one has to balance complexity or

[1]The term ”component” is typically used in the context of distributed computing. The complexity in our context is drastically reduced omitting especially the problem of threading models.

readability and performance. This balancing leads to different design decisions in the two major packages (Parfit++ and MBSNAT). In the case of MBSNAT, which heavily uses a hierarchic structure and levelization as described before, we do not use interface–oriented programming to the previously defined high extent. Instead we keep the *core* of MBSNAT in *one* component, to achieve the necessary high performance of the linear algebra solver implemented by MBSNAT.

In the case of Parfit++, the time–consuming calculations are big matrix–vector or matrix–matrix operations, which are calculated using machine–optimized BLAS and LAPACK routines as described in section 10.2.1. Therefore, we are free to apply the concept of interface–oriented programming to further increase the flexibility and maintainability of Parfit++.

## 11.4 Applying Interface–Oriented Programming to Parfit++ and MBSNAT

In this section, we first describe how the parameter estimation algorithm is devided into several smaller parts following the structure already theoretically introduced in part I. These parts are associated with components. We discuss the basic interfaces which show

- the connection of the different parts / components,
- the functionality of the component itself.

This is done in detail at a very simple version of the iterative solution of a nonlinear least squares problem. Doing this we want to give an impression of the structure of the implementation and how interfaces are used. In this overview we will present only a small selection of over 100 interfaces used in the two packages. After this introduction we will present the implemented components, which in combination either amount to Parfit++ or to MBSNAT.

### 11.4.1 Basic Structure

To illustrate the component structure of this part, we look at a simple example (compare to equations 1.12)

$$\min_x \|r(x)\|_2^2.$$

This problem can be solved using a QR–Decomposition, and the solution is found as a fixpoint of the iteration scheme:

$$x_{i+1} = x_i - (Q(x_i)R(x_i))^+ \, r(x_i)$$

When we look at this problem in a more abstract point of view, we have

- a nonlinear problem at some state $x_i$, which should be able to supply the linearized form: $J(x_i), f(x_i)$. This is our first component, the PROBLEM.
- The linear form is solved by applying a QR–decomposition: the second component which is called SOLVER.
- The solution is used to apply an iterative scheme: the ITERATOR. This scheme iteratively sets the PROBLEM to a new state until some tolerance is fulfilled and the corresponding nonlinear problem is solved.

This is shown in figure (11.6).

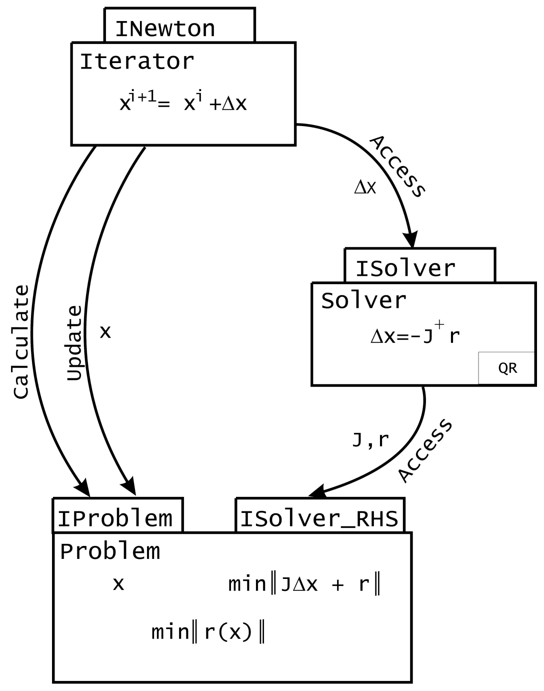


Figure 11.6: Component model of an algebraic problem

Besides the decomposition in three components, we also specify a number of interfaces such that the components are able to communicate with each other: INEWTON, ISOLVER, ISOLVER_RHS and IPROBLEM. A slightly simplified version of these interfaces is shown in appendix B. To illustrate the basic functionality of these interfaces, we want to take a closer look at INEWTON and IPROBLEM.

IPROBLEM holds the problem definition. The class is specified as a state machine holding its own state $x$. Thus, a procedural function call $f(in, out)$ is split into three parts: UPDATE(), CALCULATE() and ACCESS(). This structure is represented in the interface. We want to stress here that the state is stored inside the problem because of the state–machine or object–oriented approach. This has the advantage that the state may be structured, which remains hidden from the rest of the program, or that there is some other method to set or modify the state, which does not impair this principle calling structure.

INEWTON is the basic interface to the iteration scheme. As this is the topmost component, it must know about the other components. Furthermore, the initialization is done in the first part. The second part of the interface deals with the internal state of the iteration scheme and provides functionality to set the tolerance or the maximum number of steps, and also to ask for information. The last method of the interface is the actual main routine OPTIMIZE(). We want to point out that the procedure has no argument, because all the information, in particular the state as an internal detail of the problem, is already known. The interface provides access to an iterative scheme with minimal functionality. As prerequisites, we provide a problem (which should be solvable with an iterative scheme) and a solver using information of the problem to provide an increment.

Although this framework may be disproportionate for such a simple problem, it is valid in a much more complex environment, and the principle properties of the approach can be seen in this simple example. Furthermore, we typically need to extend only one component, using another interface and multiple inheritance, thus a complex model is built by small extensions of the simple ones.

Considering a constrained minimization problem (see equations 1.12 and 1.13), the situation changes slightly

$$\begin{aligned} \min_x \|r_1(x)\|_2^2 \\ r_c(x) = 0. \end{aligned}$$

In this case, we must distinguish between least squares conditions and equality constraints. As a result, the problem component supplies an additional interface: IPROBLEM_LEASTSQUARES. Moreover, we have to adapt the solver component to be able to solve constrained minimization problems, but this is just an implementation detail, as in the latter case. The topic we want to stress here is that we do *not change* the original interface but only *add* another interface. Therefore, other components working with the original version of this component are still fully functional in contrast to the situation in classical procedural languages.

Until now, we have restricted ourselves to algebraic problems. In the case of dynamic problems, it is useful to extend the component model by adding an INTEGRATOR and a MODEL which supply the differential equation and a method to solve the corresponding IVP. This is shown in figure (11.7).

The basic connections between the components are that the optimization variables of the problem are the start values of the integration, i.e. the initial state of the model. The model supplies the right–hand side evaluation of the differential equation, while the integration scheme modifies the state of the model according to the differential equation. This is reflected in the methods of the corresponding interfaces: IINTEGRATOR and IMODEL with MODE, as the ODE–extension of a simple storage for variables.

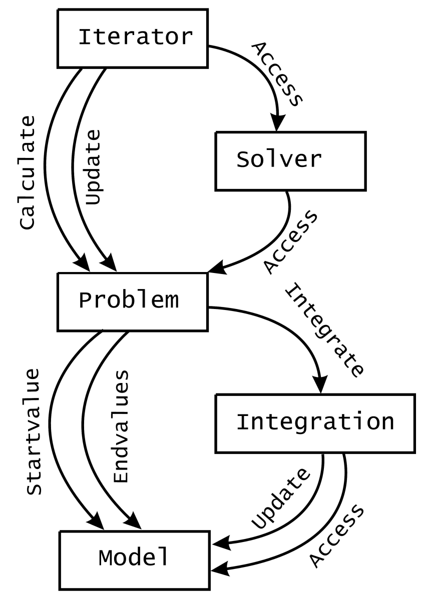


Figure 11.7: Component model of a dynamic problem

### 11.4.2 Description of Components

In the following, we want to present a short description of the components of which Parfit++ consists. The description is enriched by several references to the theoretical part, which are used to describe the exact content of the implementation.

**Iterator** The component is accessed via the basic interface INEWTON and the basic functionality is described above (see appendix B). There are three implemented classes, which are both inside the same DLL: a simple full–step Gauß–Newton method, a damped Gauß–Newton method based on line search and natural level functions, and one method based on the restrictive monotonicity test as described in section 1.3.2.

**Solver** This component is used to solve the linearized form of constrained least squares problems (see section 1.4). All the classes in this component inherit from ISOLVER (see appendix B) and provide the corresponding functionality. There are several implementations which are distinguished from each other by the type of the linear algebra solver used to treat the constraints and the unconstraint minimization. Furthermore, as the constraint qualification (definition 2) must be fulfilled and the system should be positive definite (definition 3), we apply regularization to guarantee the latter (see section 1.4).

The classes in this component do not only provide a solution of constraint minimization problems, but they also give statistical information about the solution, i.e. the decompositions are used to calculate the corresponding covariance matrix and to provide the standard deviation (see section 3.4).

An extension of this component is the solution of multiple experiments problems (see section 3.1). The structure is implemented as a list of problems originating from the component PROBLEM_DYNAMICS or PROBLEM_ALGEBRAIC. From the point of software engineering, it is necessary to administrate this list and prevent the elimination of "global parameters" during the solution of local problems. This is again done using additional interfaces.

**Problem_Dynamics** This component implements the dynamic problems based on multiple shooting including the corresponding condensing algorithm.

The component PROBLEM_DYNAMICS uses an INTEGRATOR component to solve a differential equation specified in MODEL in order to evaluate the least squares conditions and constraints at different time–points, and to generate derivative information. The optimization variables correspond to initial values of the DE. For the corresponding mathematical theory, see chapter 2.

The implementation of multiple shooting leads to an extensive internal structure. As described in section 3.1, the time–interval is discretized (see equation 2.23). This is realized as a list of multiple shooting nodes. Apparently, not using multiple shooting discretization is equivalent to only one multiple shooting node. Moreover, we defined conditions w.r.t. several time–points (see equation 2.4) that need not coincide with the start of the multiple shooting intervals. Therefore, we implement another list with elements of type STOPPOINT which hold the terms of the equations w.r.t. one time–point. These objects are associated to the corresponding node. The two separate lists allow for a flexible treatment of the multiple shooting discretization and are used to exploit the structure of the derivative of the conditions. This includes the possibility to automatically generate initial values of the multiple shooting node by using arbitrarily distributed least squares conditions or constraints (see section 3.2).

The data concerning the multiple shooting discretization are stored in a class of this component. Because the condensing algorithm uses these structures to allow for an efficient decomposition and solution of the corresponding part, it is evident to apply these steps inside the component. As a result, classes of this component inherit from ISOLVER, too. Furthermore, it is possible to apply reduced methods as described in chapter 9.

**Problem_Algebraic** This component implements the formulation of algebraic problems originating from the identification of a parameterized manifold by measuring several states as described in section (3.1.1). Due to the fact that the states are different and the parameters are the same at different measurement–points, we automatically get a multiple experiment structure. An efficient specialized treatment is implemented in this component. If we formulate several series of measurements with partly different parameters, we get an additional level of the multiple experiment structure which has to be realized using the multiple experiment implementation of component SOLVER.

**Equation** Looking at the formulation of all these problems, we are able to interpret them as a set of functions $r^i : R^n \to R^1$. Each of these functions has a specific meaning, i.e. they are part of a constraint or least squares condition. Although the meaning may vary, the basic structure is the same. This is a good example of the use of polymorphism and information hiding. These terms are implemented in this component and the corresponding classes inherit from IEQUATION (see appendix B).

**Integrator** This component provides integration methods to solve differential equations of different types. Several up–to–date methods are available, but they are written in Fortran. Because of this, we provide wrapper classes to neatly fit these methods in this framework. These libraries are ODESIM and ODEOPT for ODEs, MBSSIM and MBSOPT for mechanical DAE of index 3 (see section 2.1.1; [vW97], [Sch99], [Win01]) and DAESOL for stiff DAE of index 1 (see section 2.1; [BBS99], [Bau00]). ODEOPT, MBSOPT and DAESOL provide sensitivity information as described in section 2.2. All methods are able to treat non–discontinuous differential equations as described in section 3.3.

These wrapper classes provide the same interface for the same feature of an integration package, independent of the exact underlying package or scheme. If a package can treat, for example, discontinuities, it has an appropriate interface, and it is the same interface for all the methods. The internal "detail" of how the interface is implemented is not important. This is a example to distinguish interface–oriented programming and object–oriented programming. While the latter means implementation inheritance when speaking of inheritance, the former only focuses on inheritance of the interfaces. In this case, the code base is surely very different.

**Model** This component implements a differential equation, be it an ODE, DAE of index 1 or a mechanical DAE. It contains classes that hold a state (IMODEL) and a time and are able to supply the right–hand side of a differential equation (e.g. MODE). Several further interfaces are defined supporting differential equations of different types. There exist two variations of this component: one supplies various differential equations used to test Parfit++, the other one is the complete core of MBSNAT described before.

Besides these core components, there are several utility components, which are typically associated with the first or third tier, i.e. presentation layer or database layer. This exemplifies that is is useful to combine several approaches in order to reduce the complexity of the software. The components associated with the first tier are the following:

- A graphical user interface based on MFC of Microsoft.
- A collection of dialogs, also based on MFC, to support the interactive generation and editing of mechanical elements of MBSNAT.
- An engine handling 3D–data based on OpenGL, which supports the interactive definition and removing of elements. This uses selection and picking support of OpenGL. The main loop of an GUI–based system always resides in the GUI. Therefore, we need an additional code to interpret several mouse–events to add another element.
- A wrapper interpreting data from the core routines to present them as 2D–diagrams. Depending on the environment, one can use a version that communicates online with Matlab via ActiveX, or a version that generates data–files and control–files, which can be read by Gnuplot.

The component associated with the last tier is

- an xml–wrapper handling of XML–based input independently of the platform. Depending on the environment, it is based either on MSXML 3.0 of Microsoft or on our development based on string handling.

Again we see that the components presented above are highly dependent on the platform and on the software environment. Therefore, it is very important to provide specialized versions of these components. This keeps the core routines platform independent, i.e. we do not need to change these routines when we move to another platform, but we can still use the tools provided especially for this platform. Furthermore, a porting of such functionality to a previously not supported platform is very simple because it is not necessary to case about the highly complex numerical core routines in this case.

## 11.5 Object-Instantiation – The Factory Concept

Previously, we have focused on the modularization of the code e.g. by using interfaces. We have stressed the importance of interfaces to regain the properties typically associated with OOP. When considering the allocation of the object, we have to expose the complete definition of the corresponding class to the calling component. Therefore, one has to change the way of doing the allocation of objects to avoid this strong coupling. The degree of flexibility this can be accomplished with is a major criteria for the usability and flexibility of the whole program.

The first and simplest dynamic form one can think of in C++ is to just type in the code for dynamically allocation of one specific object, let us say, an object **a** of a class **A**:

```
A *a = new A();
```

This object is allocated on the heap and at runtime, so that we can avoid some of the problems associated with languages like Fortran. But this form does not consider a choice: The type of **a** is hard coded into the program. If we want to allow user–interaction, it is necessary to select the correct class according to the user's wish. Let us say we have two additional classes **B** and **C** that inherit from **A**, and we want to initialize again the (polymorphic used) variable **a**:

```
switch (user_input) {
    case 'b': a = new B(); break;
    case 'c': a = new C(); break;
    default : a = NULL;
};
```

With this construct, the type of variable **a** is easily selectable by the user input. But when we take a closer look, we see that this has some major drawbacks: The type of the object can be selected, but only from a predefined set of classes that is hard–coded in this code segment. If one wants to broaden the set of classes, one has to modify the existing code at maybe several different places, which is obviously prone to errors.

One possibility to avoid this is not only to use the dynamic allocation of objects, but to define the set one can choose from dynamically. Therefore, we define global object, the so–called *finder*, which contains a map[2] of so–called factory–objects, such that every selectable class is included in form of one instance. By the specification of the key the finder returns a pointer to the selected element, which is an instantiation of the desired class. This object is able to CLONE() itself, that is to return an instance of the desired class. The corresponding code is:

```
A* p = dynamic_cast<A*>(finder->Get(key));
```

A new instance of the wanted type of class is selected by the "key" argument of the GET() method of the finder. The pointer is of a common type which we have to cast to the correct one. Other methods would generate code dependencies as seen above. This approach is known as the factory–concept [Lin98].

With this method, we are now able to dynamically request a new instantiation of a class specified by a key. This allows for an interactive build–up of arbitrary models or optimization problems using a GUI, without recompilation after every slight modification. Furthermore, we are able to extend existing environments with additional functionality, which is possible without recompilation of the existing code, either. As such, the user is able to code *plug–in*s.

This concept is well known in distributed computing and used by Microsofts technology COM [Box98], by CORBA or Sun's JAVA–RMI. Our implementation is kept much simpler, because topics like object–replication on a different machine

[2]This map is not to be confused with a mathematical map. It is a container holding pairs of keys and an element sorted by the key. This is a standard container of the Standard Template Library of C++.

are not needed here, but the factory–concept is one of the basics of *component–oriented* programming which has guided the overall design of this software.

# Part V

# Numerical Results

This part is dedicated to the presentation of the efficiency of the algorithms and implementations shown in this work. This concerns namely the (bio-)mechanical modelling tool MBSNAT with its numerical properties, and the parameter identification tool Parfit++ in combination with MBSNAT. As in the previous parts, efficiency means performance in terms of speed *as well as* flexibility and usability. In the first chapter, this is discussed in detail at models originating from technical mechanics, and, in the next chapter, it will be applied to a large bio–mechanical model of a human.

We start with simple examples like the single planar pendulum, and then incrementally increase the topological complexity. We append additional bodies and show the decomposition and solution of the Augmented system of the algebraic part of the DAE in linear complexity w.r.t. the number of bodies. By changing the type of the topology to tree–structured systems, we regain this linear complexity. If one extends the type to closed loop systems, it is possible to regain linear complexity w.r.t. a varying size of the models if one fixes the principle topology. The linear complexity is achieved by using the structure of the matrix and not by eliminating parts at the startup.

Afterwards we move on to the optimization context and show the performance of the derivative generation routine, which is the most time–consuming task of the overall optimization algorithm. We will see that, by using the structured decomposition algorithms and IND, the effort to evaluate one directional derivative is approximately two to three times faster than the evaluation of the main trajectory.

The next sections are dedicated to several more complex examples: the hexapod machine tool and the 6–bar–mechanism. The first model is characterized by a huge number of closed kinematic loops. The model consists of a load that is connected to the inertial system by six hydraulic actuators, such that the load can be moved in all six degrees of freedom of the free body. Each actuator is modelled using two rigid bodies, which results in a model of 13 bodies corresponding to 312 dynamic variables.

After this, we present the 6–bar–mechanism with only one closed kinematic loop but with the additional difficulty that no globally valid set of minimal coordinates exist. We will show a simulation of this model, using coordinate partitioning as introduced by us in section 5.2.

In the second chapter, we discuss various reduction methods introduced to use BVP–constraints to efficiently solve the linearized optimization problem. We discuss the effects of different combinations of the following categories:

- relaxation (chapter 8)
- projection in the case of the limit form of non–stiff Baumgarte relaxation (section 8.5)
- reduction methods (chapter 9)
- globalization strategy (section 1.3.2)

The best result is achieved by a combination that depends on the problem and on the question whether one wants to use a parallelized version.

Not considering parallel computers and focusing on problems with fixed initial dynamic values, the best results is achieved

- by applying RMT with back projection (see section 1.3.2)
- to the linearized problem reduced by applying the Fixfit approach (see section 9.1)
- where we use the limit form of non–stiff Baumgarte relaxation (see section 8.4)
- based on a projection adapted to mechanical systems (see section 8.5.4).

Using RMT with back projection, we better follow a *consistent* homotopy path associated with the solution of the non–linear parameter estimation problem. In the case of mechanical DAEs, this will be shown to be very beneficial. If the initial dynamic variables are fixed, then Fixfit–approach is optimal to reduce the number of directional derivatives needed while regaining the advantages of multiple shooting. Furthermore, our version of relaxation will prove to perform better compared to either other relaxation approaches or methods based on a non–redundant set of variables in the optimization context. These results emphasize the need to carefully adapt all parts of the optimization algorithm and modelling.

Because the Fixfit–approach does not allow for a parallelization on the multiple shooting level, it may then be advisable to use Reduction Method II (see section 9.2.2) and the same relaxation and globalization technique as presented before. Regrettably, because of the structure of the linear algebra, the previously favored projection method can not be applied. One has to use the Euclidean Form (see section 8.5.3).

After studying the algorithms for some test cases, we then apply the combination that we found to be the best one, to a huge bio–mechanical model.

As input data we use the position of markers stuck to the body and filmed by stereo–cameras, which is a typical situation found in modern gait laboratories. We will see, that the measurement data of just one such experiment is not enough to identify the parameters of the model accurately. Therefore, we combine the data of eight independent measurement series.

The model consists of 22 bodies for the bio–mechanical model of a human and 6 bodies for an additional seat model, summarizing to 672 variables. As parameters that are to be estimated we selected force–constants of intervertebral discs, forces connecting wobbling masses to the skeleton, and 3D–forces to realize passive muscles. Our approach tries to find these internal parameters of a model of a human based on a non–invasive examination, which is in heavy contrast to invasive surgery or x–ray examinations. Furthermore, we are not interested in the constants defining the force of the real body parts, but in parameters, such that the idealized model fits the real human.

# Chapter 12

# Numerical Properties of the Modelling and Sensitivity Generation Algorithms

We start to study the numerical properties of our approach by focusing on the linear algebra used to decompose and solve the Augmented system of the mechanical DAE in Full Descriptor Form (2.14). This is the most time–consuming part in the simulation context. Therefore, it is useful to achieve a minimal complexity w.r.t. the number of bodies and, therefore, a good performance in the case of huge models. Furthermore, we also want to be efficient in the case of less complex models. The approach shown here is competitive with other approaches in the case of medium and large automatically generated models.

Besides studying the performance in the simulation context, we also present the efficient evaluation of the right–hand side of the variational differential equation. Due to the structured solver, we are able to get a complexity of $\mathcal{O}(n_{Bodies} \cdot n_{Direction})$, where $n_{Bodies}$ is the number of bodies and $n_{Direction}$ is the number of directional derivatives. Due to using IND and exploiting the resulting structure, the calculation of one additional direction is faster than the evaluation of the right–hand side of the original differential equation by a factor of approximately three.

## 12.1 Modelling of Different Types of Pendulums

We start the study with increasingly more complex types of pendulums in the simulation context. The numerical results prove the linear complexity of the evaluation of the right–hand side for different types of models and topologies. For this first test, we choose to use small examples, so that we can study the behavior in a manageable environment.

We use SI–units throughout the test set.

### 12.1.1 N–Pendulum

The first model contains one body, which is connected to the inertial system by a revolute joint. An OpenGL representation of the model is shown in figure 12.1.

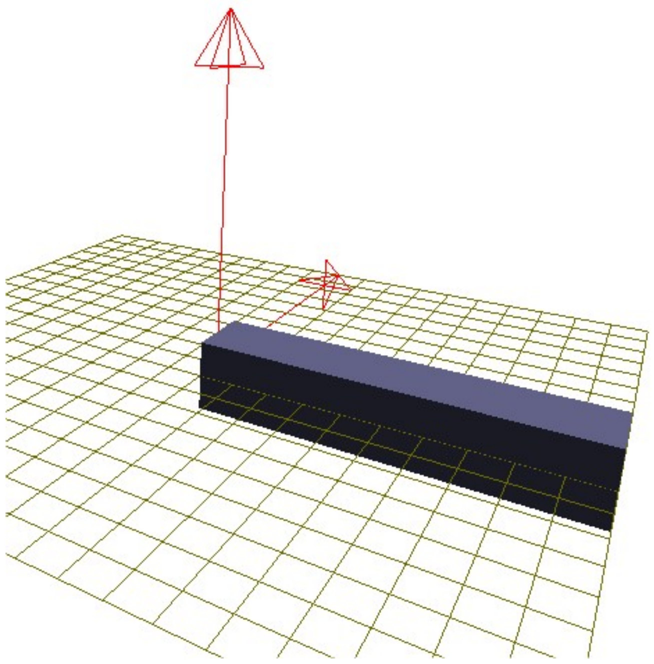


Figure 12.1: Single Pendulum (Initial State)

To exemplify the modelling process, we present the XML–Script of this model.

After an introductory part concerning the simulation and some remarks, we define the body ”Body1”

```
<body>
    <id> Body1 </id>
    <position>
        <o> 0.0 0.0 0.0 </o>
        <x> 1.0 0.0 0.0 </x>
        <y> 0.0 1.0 0.0 </y>
        <z> 0.0 0.0 1.0 </z>
    </position>
    <mass> 2.0 </mass>
    <center_of_mass> 0.5 0.0 0.0 </center_of_mass>
    <moment_of_inertia>
        <x> 2.5 0.0 0.0 </x>
        <y> 0.0 2.0 0.0 </y>
        <z> 0.0 0.0 2.0 </z>
    </moment_of_inertia>
</body>
```

which is located at the origin and has the same orientation as the inertial system. Furthermore, the body has a mass of 2.0 kg, the center of mass of the body is located at $\begin{pmatrix} 0.5 & 0.0 & 0.0 \end{pmatrix}$ w.r.t. the local body–fixed frame, and the moment of inertia $[Nm^2]$ in the classical form w.r.t. Eulerian Angles is given. We do not specify a node for the velocity, which is interpreted as zero velocity (and accordingly zero angular velocity). For simplicity reasons, we omit the node for the shape, which defines the optical look of the body as drawn by OpenGL.

The next step is to connect the body to the inertial system by using a revolute joint.

```
<joint key="Revolute">
    <id> Joint1 </id>
    <begin id="Inertial">
        <o> 0.0 0.0 0.0 </o>
        <x> 0.0 0.0 1.0 </x>
        <y> 1.0 0.0 0.0 </y>
        <z> 0.0 1.0 0.0 </z>
    </begin>
    <end id ="Body1">
        <o> 0.0 0.0 0.0 </o>
        <x> 0.0 0.0 1.0 </x>
        <y> 1.0 0.0 0.0 </y>
        <z> 0.0 1.0 0.0 </z>
    </end>
</joint>
```

The key attribute defines the type (”Revolute”) of the joint. The ID is ”Joint1”. Furthermore, there are two markers associated with the ”Inertial”–system and the body with ID ”Body1”. The joint fixes the local origin of ”Body1” w.r.t. the body–fixed coordinate system to the origin of the inertial system. The global representation of the z–axes of both markers define the rotational axis, which will be the y–axis of the global system. The definition of x– and y–axes of the markers are optional. As a result, the body is able to rotate only around the global y–axis.

At last, we define a driving force, which is the gravitation.

```
<force key="Gravitation">
    <id> Gravitation </id>
    <constants> -9.81 </constants>
</force>
```

This force naturally affects all bodies. The gravitational gravitational constant is $-9.81\frac{m}{s^2}$.

The script shown above provides the implementation of a simple single pendulum. Longer chains are modelled as easily by just adding bodies with different initial positions and joints which connect the respective neighboring bodies.

Different joints are modelled using different keys and suitable interpretations of the markers. Other forces connecting two bodies are modelled in the same way as in the case of the joints. A complete listing can be found in appendix C.

Now we want to look at the scaling of the performance w.r.t. the number of bodies. Each new body is added at the end of the chain and positioned along the x–axis. All initial velocities are zero.

The calculation has been done on an Athlon 1.2GHz using Windows XP and Microsoft Visual C++.Net. We simulate for 10 seconds and want to achieve an relative error of $10^{-6}$ and an absolute error of $10^{-8}$ using the Adams–Bashforth–Moulton method (MBSABM) of MBSSIM.

The table shows the number of bodies together with the cost of the evaluation, decomposition and solution in milliseconds. To better compare the costs we additionally provide the cost per body. All results we done by averaging the cost using 1000 samples.

| | Evaluation [ms] | | Decomposition [ms] | | Solution [ms] | | All [ms] |
|---|---|---|---|---|---|---|---|
| #Bodies | all | p. body | all | p. body | all | p. body | p. body |
| 1 | 0.077 | 0.077 | 0.047 | 0.047 | 0.016 | 0.016 | 0.140 |
| 3 | 0.124 | 0.041 | 0.187 | 0.062 | 0.047 | 0.016 | 0.119 |
| 5 | 0.171 | 0.034 | 0.328 | 0.066 | 0.078 | 0.016 | 0.115 |
| 7 | 0.249 | 0.036 | 0.468 | 0.067 | 0.109 | 0.016 | 0.118 |
| 10 | 0.344 | 0.034 | 0.672 | 0.067 | 0.172 | 0.017 | 0.119 |
| 12 | 0.407 | 0.034 | 0.828 | 0.069 | 0.203 | 0.017 | 0.120 |
| 15 | 0.484 | 0.032 | 1.032 | 0.069 | 0.250 | 0.017 | 0.118 |
| 20 | 0.656 | 0.033 | 1.391 | 0.070 | 0.328 | 0.016 | 0.119 |
| 25 | 0.829 | 0.033 | 1.750 | 0.070 | 0.407 | 0.016 | 0.119 |
| 30 | 0.954 | 0.032 | 2.109 | 0.070 | 0.500 | 0.017 | 0.119 |
| 40 | 1.312 | 0.033 | 2.954 | 0.074 | 0.703 | 0.018 | 0.124 |
| 50 | 1.812 | 0.036 | 3.985 | 0.080 | 1.015 | 0.020 | 0.136 |
| 100 | 6.125 | 0.061 | 8.219 | 0.082 | 2.547 | 0.025 | 0.169 |

An additional overhead can be seen for a very low number of bodies and for a very high number of bodies. The first is an always expected overhead, while the second is may be due to cache–management and –sizes. Nevertheless, the calculation time for one evaluation of the right–hand side grows approximately linearly w.r.t. the number of bodies for the range of dimensions we are interested in.

If one wants to compare these results with other modelling tools, it is necessary to keep in mind that the effort of our method increases with the number of constraints, while an approach using some kind of minimal coordinates scales with the number of degrees of freedom. Therefore, we want to present results from a system similar to the open loop case shown above, but with spherical joints instead of revolute joints.

The table shows the number of bodies together with the sum of the cost of one evaluation, decomposition and solution for both models divided by the number of bodies. Additionally we show the relative cost of the model with spherical joints compared to the model with revolute joints.

| | **Spherical** | **Revolute** | |
|---|---|---|---|
| # Bodies | cpu–time [ms] | cpu–time [ms] | Rel. Effort |
| 5 | 0.091 | 0.115 | 79% |
| 10 | 0.091 | 0.119 | 76% |
| 20 | 0.089 | 0.119 | 75% |
| 30 | 0.090 | 0.119 | 76% |
| 50 | 0.095 | 0.136 | 70% |
| 100 | 0.119 | 0.169 | 70% |

The cost of the evaluation of the model using spherical joints is approximately 20% less than the one using revolute joints. An ideal scaling w.r.t. the number of constraints is not achieved, because this would neglect the decomposition of the block corresponding to the Augmented system of one body.

We will see that free bodies or joints which restrict a low number of degrees of freedom, i.e. have less constraints per body, are more often used in the field of biomechanics in than technical mechanics. Therefore, a modelling technique that has a better performance for a lower number of constraints – as our approach – is better suited to treat the biomechanical models we are most interested in.

## 12.1.2 Tree–Structured and Closed Loop Systems

To study the performance w.r.t. different topologies (see section 4.7), we compare the following models:

- a *linear chain* of $n$ bodies (see figure 12.2),
- a *tree–structured system*, where one chain of $n$ bodies is connected to the ground and two other chains of $n$ bodies which are connected to the end of this chain (see figure 12.3),
- a system with *closed kinematic loop*, which consists of $5n$ bodies organized in a loop of $4n$ bodies, which are connected at one point to a chain of $n$, which is then connected to the inertial system (see figure 12.4).

Doing so, we are able to compare different types of topologies with the same number of bodies.

Figure 12.2: Linear chain of $n$ (here $n = 1$) bodies

All systems have an initial velocity of zero. The linear chain is has an initial position along the x–axis, each part of the tree–structured system is again positioned along the x–axis (in contrast to figure (12.3) where we positioned one part

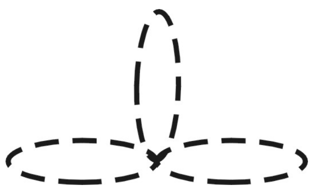


Figure 12.3: Tree–structured system of $3n$ (here $n = 1$) bodies

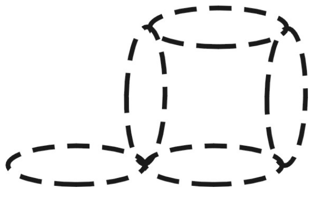


Figure 12.4: System of $5n$ (here $n = 1$) bodies with closed kinematic loop

parallel to the z–axis for presentation reasons), and the system with closed loop is positioned as shown.

First, we want to compare the tree–structured system to a linear chain. The table shows the number of bodies together with the complete cost of one evaluation, decomposition and solution of both models. Additionally the table presents the relative cost of the tree–structured system compared to the linear chain. The following tables again show the average cost based on 1000 samples.

| | **Tree–Structured** | **Linear Chain** | |
|---|---|---|---|
| #Bodies | cpu–time [ms] | cpu–time [ms] | Rel. Effort |
| 15 | 0.119 | 0.118 | 101% |
| 30 | 0.120 | 0.119 | 101% |
| 45 | 0.130 | 0.130 | 100% |

As expected, the decomposition and solution of the Augmented systems of tree–structured systems has a negligible overhead w.r.t. the linear chain. This is due to the fact that no additional fill–in is produced.

In the second example, we compare the linear chain with a corresponding closed loop system. We achieve the following results:

| | **Closed Loop** | **Linear Chain** | |
|---|---|---|---|
| #Bodies | cpu–time [ms] | cpu–time [ms] | Rel. Effort |
| 5 | 0.181 | 0.115 | 157% |
| 10 | 0.172 | 0.119 | 145% |
| 15 | 0.171 | 0.118 | 145% |

In contrast to the tree–structured system, there is an overhead because of fill–in that can not be avoided. This overhead is constant w.r.t. the number of bodies if the fundamental topology is unchanged, which is achieved by replacing every body by two or more bodies, i.e. by fixing the relative number of bodies of each part.

As a result, we are have linear complexity of the decomposition and solution algorithm w.r.t. the number of bodies in the case of linear chains and tree–structured systems. In the case of closed kinematic loops with only one loop, we need an additional effort which is proportional to the size of the loop. Depending on the exact configuration of more complex systems, we do not longer have linear complexity, but for the situation where we replace each body with two other bodies, i.e. fix the topology, we again achieve linear complexity.

### 12.1.3 Sensitivity Generation

Next, we evaluate the performance of the MBSNAT w.r.t. derivative generation. Therefore, we extend the previous models to include parameters. Due to the object–oriented approach and because we have a fully parameterized model, it is very easy to add a parameterization. To parameterize e.g. the value of the gravitational constant, we just add the following line to the XML–input file:

```
<parameter key="Constant_0"> Gravitation </parameter>
```

The first constant (which is the gravitational constant) of the force ”Gravitation” is now a parameter.

We repeat the numerical experiment for the linear chain as described in the previous section, but now we include the derivative w.r.t. a modification of the gravitational constant and all degrees of freedom of the system. As we assume a linear chain connected by revolute joint, we have $n$ kinematic degrees of freedom per body or $2n$ additional directions. Here, we again used 1000 samples. In the simulation case, we computed the effort as the sum of each part: computing the mass and constraint matrix together with the forces and $\gamma$, the decomposition of the Augmented system, and the solution of the linear system of the algebraic constraints.

The tables show the number of bodies and directions together with the sum of the cost of an evaluation, decomposition and solution of the differential equation and the cost of computing the directional derivatives. In the first case we divide as by the number of bodies while in the second case we additionally give the cost divided by the number of bodies times the number of directions to compare. Furthermore, we provide the relative effort to calculate one directional derivative compared to the cost to evaluate the differential equation.

| | | **Simulation** | **with Sensitivity** | | |
|---|---|---|---|---|---|
| | | per Body | complete | p. Body & Dir | |
| #Bodies | #Dir | cpu–time[ms] | cpu–time[ms] | cpu–time[ms] | Rel. Effort |
| 1 | 3 | 0.140 | 0.125 | 0.042 | 30% |
| 3 | 7 | 0.119 | 0.891 | 0.042 | 36% |
| 5 | 11 | 0.115 | 2.33 | 0.042 | 37% |
| 7 | 15 | 0.118 | 4.48 | 0.043 | 36% |
| 10 | 21 | 0.119 | 9.02 | 0.043 | 36% |
| 12 | 25 | 0.120 | 13.3 | 0.044 | 37% |
| 15 | 31 | 0.118 | 21.0 | 0.045 | 38% |
| 20 | 41 | 0.119 | 37.1 | 0.045 | 38% |
| 25 | 51 | 0.119 | 60.9 | 0.048 | 40% |
| 30 | 61 | 0.119 | 96.4 | 0.053 | 44% |
| 40 | 81 | 0.124 | 191 | 0.059 | 48% |
| 50 | 101 | 0.136 | 308 | 0.061 | 45% |
| 100 | 201 | 0.169 | 1325 | 0.066 | 39% |

Furthermore, we present the results for the pendulum using spherical joints.

| | | **Simulation** | **with Sensitivity** | | |
|---|---|---|---|---|---|
| | | per Body | complete | p. Body & Dir | |
| #Bodies | #Dir | cpu–time[ms] | cpu–time[ms] | cpu–time[ms] | Rel. Effort |
| 5 | 31 | 0.091 | 4.94 | 0.032 | 35% |
| 10 | 61 | 0.091 | 20.7 | 0.034 | 37% |
| 20 | 121 | 0.089 | 82.1 | 0.034 | 38% |
| 30 | 181 | 0.090 | 193 | 0.036 | 40% |
| 50 | 301 | 0.095 | 676 | 0.045 | 47% |

As we see, the overall complexity of the evaluation of the right–hand side of the variational differential equation is, as the solution of the Augmented system of the mechanical DAE, approximately of linear complexity w.r.t. the number of bodies and of linear complexity w.r.t. the number of directions. Furthermore, the evaluation of one direction is approximately two to three times faster than one for the main trajectory. A slight increase of the computational time is seen for an increasing number of bodies, which may be caused by an increased memory usage and the resulting worse cache and memory management. The same effect is seen in the simulation case for a larger number of bodies in the case of the pendulum with revolute joints. Thus, the relative effort has shows a maximum for approximately 40 bodies in this case.

## 12.2 Hexapod Machine Tool

After studying pendulums of different size, we now want to present a more complex example: the hexapod machine tool, which includes a high number of closed kinematic loops (see figure 12.5).

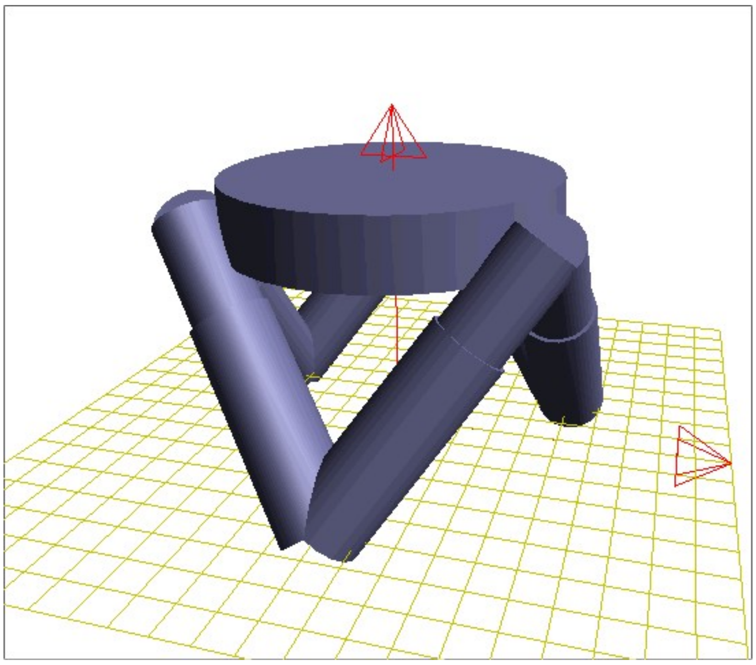


Figure 12.5: Hexapod Machine Tool

The model has a body, also called the load, which is connected to the ground by six hydraulic actuators. These actuators are placed in such a way that the load can be controlled to move in all six degrees of freedom of the free rigid body. Therefore, the model has a high number of closed kinematic loops. It is one of the most prominent examples in the field of parallel kinematic machines.

The weight of the controlling elements is not negligible compared to the load – in some cases it is even much bigger – and therefore, one has to model the actuators by using bodies. The model uses two bodies for each actuator, which are connected by a translational joint allowing one degree of freedom, and some force or control to drive the actuator. These actuators connect to the load by using spherical joints, and they connect to the ground by using cardan joints. This combination provides all the moveability that is needed, while disallowing a rotation of the actuator around its own symmetry axis. Therefore, the complete system has only the necessary six degrees of freedom and does not introduce any redundant constraints.

To study the behavior of our model within a simulation environment, we use springs with different stiffness constants to provide simple driving forces of the actuators. The initial velocity is zero and the initial position can be seen in figure (12.5). The load is 1.0 m above and parallel to the ground.

The simulation is done in the same environment as the previous examples: an Athlon 1.2GHz running Windows XP and using Microsoft Visual C++.Net as compiler. We use MBSABM of MBSSIM as an integration scheme to solve the provided differential equation with a relative tolerance of $10^{-6}$ and an absolute tolerance of $10^{-8}$. In this environment, we have simulated a model for 1.0 seconds, which has been done in 0.95 seconds. More detailed information can be found in [WK00].

Taking into account the number of evaluations of the right–hand side of the differential equation, we get a computational effort which is approximately two times higher than in the case of the linear chain. This is expected because of the high number of closed kinematic loops, each of them spanning three bodies.

Comparing the cost of evaluating one directional derivative to the cost needed in the simulation context, we need only 27%, i.e. are more than three times faster.

## 12.3 6–Bar–Mechanism

The 6–bar–mechanism is an example with ”only” one closed kinematic loop. Five bodies and the ground are connected by revolute joints to form one big closed loop without branches. The whole system has one kinematic degree of freedom, which appears only if exactly this orientation of the rotational axes of the revolute joints is chosen. A different orientation of the axes would lead to a rigid system with no degree of freedom even when the same topology is used. Therefore, although the model contains only one loop, classical formulations typically exhibit a high nonlinearity, and it is challenging for all modelling tools to close the loop.

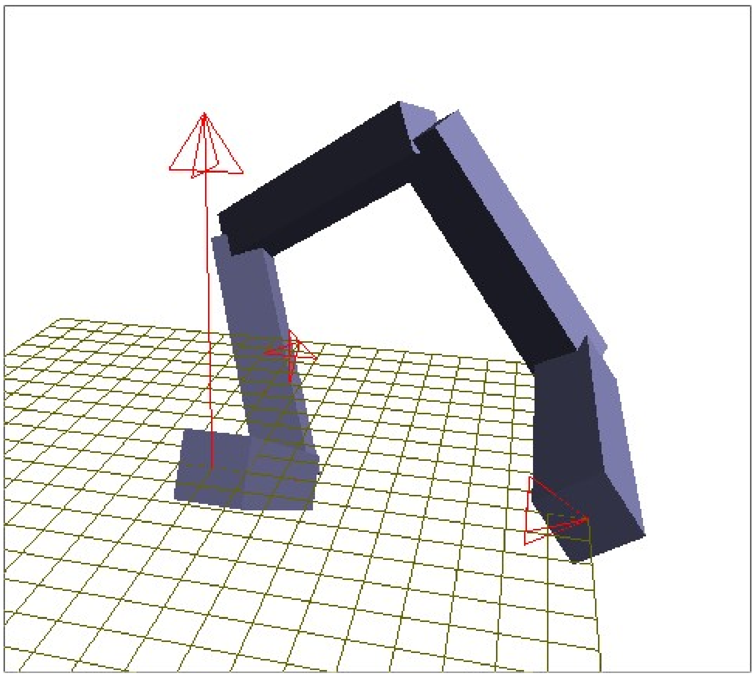


Figure 12.6: 6–Bar–Mechanism

The model has five free bodies, each of them with the six degrees of freedom of the rigid body. We include six revolute joints, each of them restricting five degrees of freedom. Because the whole system is supposed to move in this (and only this) configuration, one constraint must be redundant. In the case of modelling based on coordinate partitioning, it is known that no chosen set of variables is valid in the complete consistent space. This corresponds to the fact that in our case no global set of non–redundant *constraints* exists. Depending on the exact state, a different constraint is redundant. Therefore, the monitoring strategy introduced in section 5.2 has to be applied to detect the correct selection.

As theoretically expected and also seen in numerical tests, a change in the set of used constraints does not require a restart of the integration scheme. This is the case because a change only leads to discontinuous lagrange multipliers and not to a discontinuous acceleration of the right–hand side of the differential equation. This drastically increases the performance of the integration process.

We have used the same simulation environment as in the previous examples. To achieve a better approximation of the consistent manifold, the accuracy of the invariant projection was increased to $10^{-12}$. A consistent state is needed because only the configuration of the rotational axes we choose leads to a non–rigid system. In this sense, we need a consistent state to stay inside of a *singularity* of the model–description. It is possible to choose a medium value of $10^{-3}$ for the rank analysis to detect a change of the set of non–redundant constraints.

This example has been tested on a Athlon 1.2.GHz running the same software and integration scheme as the previous tests. We choose two sets of relative and absolute tolerances of the integration scheme to show the performance in the classical regimes, on the one hand, of low accuracy used for simulation and, on the other hand, of high accuracy needed in the optimization context.

| rel. tol. | abs. tol. | cpu–time [s] |
|---|---|---|
| $10^{-3}$ | $10^{-4}$ | 1.3 |
| $10^{-6}$ | $10^{-8}$ | 3.8 |

We simulate a time–interval of 10 seconds. In both cases an integration in more than real–time is easily possible. Figure (12.7) shows snapshots of the first of this integration interval, which should give an impression of the resulting movement of this model.

Again comparing the cost of evaluating one directional derivative to the cost needed in the simulation context, we need only 31%, i.e. are more than three times faster. Thus, the relative effort presented in section 12.1.3 is a conservative estimation.

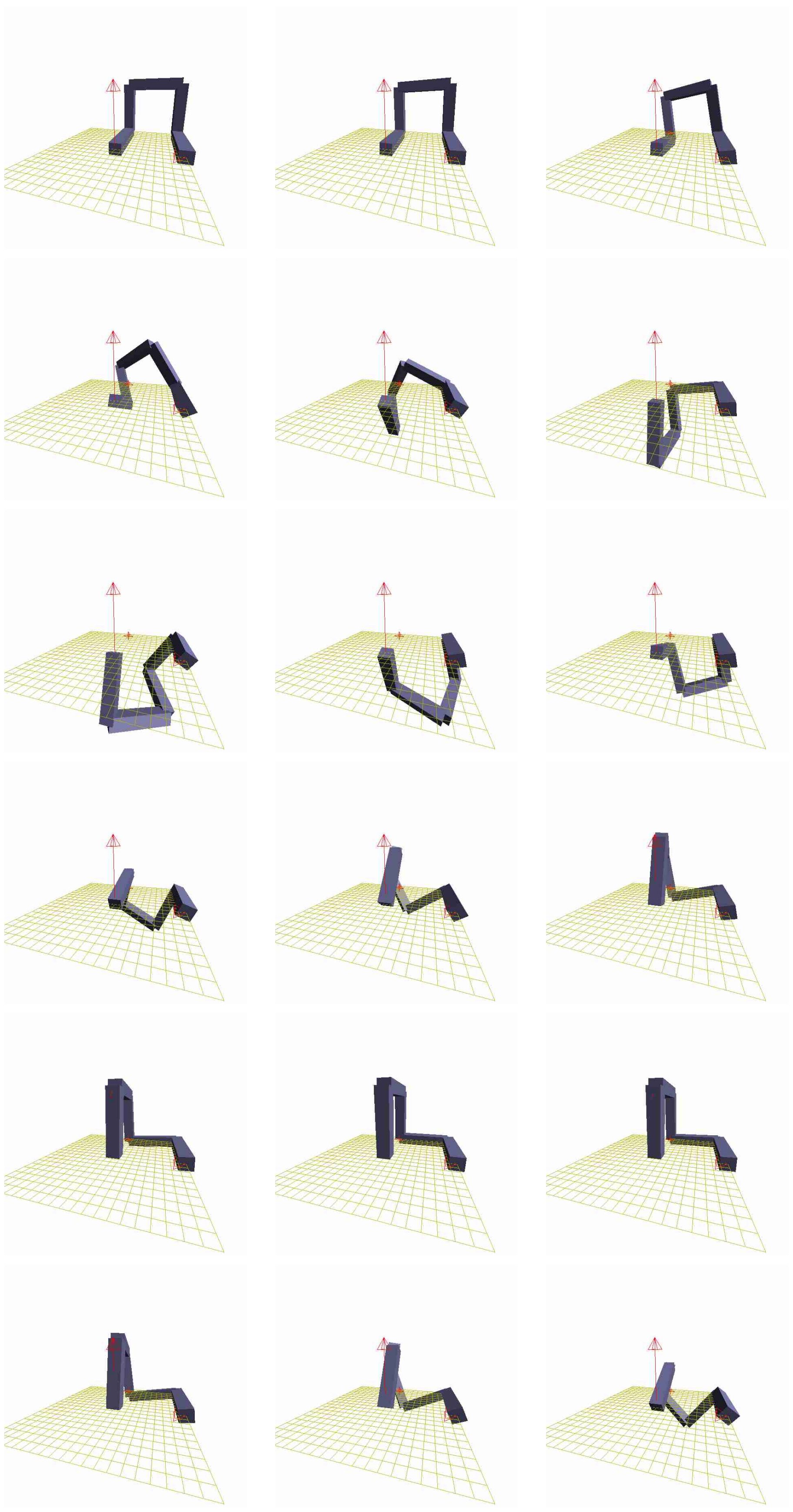


Figure 12.7: Snapshots of 6–Bar Mechanism

# Chapter 13

# Comparison of Different Methods to Achieve Optimal Performance of a Parameter Identification Tool applied to Mechanical Models

In the previous chapter, we have studied the efficiency of the evaluation of the right–hand side of the differential equation and the sensitivity differential equation. As a result, we are able to efficiently calculate one step of the generalized Gauß–Newton method of the parameter estimation problem. This chapter is dedicated to finding an optimal combination of options presented in part III to reduce the number of steps of the iterative method. Keeping in mind the theoretical results of this part, we now want to present several numerical comparisons.

For theoretical reasons, we choose to use an approach based on redundant coordinates, i.e. we use relaxation to cope with inconsistent variables. Therefore, we have to choose a relaxation method: standard relaxation (equations 8.3 and 8.4), non–stiff Baumgarte relaxation (equations 8.5 and 8.6) or the limit form of non–stiff Baumgarte relaxation (section 8.4). In the latter case, we have to choose a projection method, which either fixes a selection of variables and calculate the others according to the constraints (section 8.5.2), or which minimizes the change w.r.t. Euclidean norm under the restriction of the constraints (section 8.5.3) or which uses an projection method adapted to mechanical systems (section 8.5.4). Moreover, we study an approach which corresponds to using minimal coordinates in the optimization context. Hence, we are able to compare to more classical methods by modifying a relaxed method step by step until it is comparable to a method based on a set of non–redundant optimization variables.

The use of redundant variables is always associated with constraints where IVP–constraints of the mechanical DAE correspond to BVP–constraints of the parameter estimation problem. Furthermore, the initial values of the problem – or at least parts of them – are typically fixed resulting in additional constraints. The next section studies different approaches which use these constraints to reduce the number of directional derivatives needed to evaluate the linearized optimization problem. The Fixfit approach (section 9.1) uses fixed initial values to eliminate the need to calculate directional derivatives w.r.t. these directions. Another approach (section 9.2.1) uses the BVP–constraints associated with the invariants at the start of each multiple shooting interval to reduce the effort of derivative generation, while a second variant (section 9.2.2) additionally uses these constraints to reduce the effort needed during the condensing algorithm. We want to remark that only the first two options are able to apply all versions of relaxation. Therefore, we use the Fixfit–approach to compare the effect of different relaxation methods on the convergence of the parameter estimation problem.

At last we compare different globalization strategies of the iterative method (see section 1.3.2). Besides the possibility to apply a full–step method, there is also the possibility of using the classical monotonicity test or the restrictive monotonicity test (RMT), which are both based on natural level functions. For each damped method, an additionally back projection can be applied. As the numerical examples will show, these methods will have a strong influence on the convergence properties of the optimization algorithm.

We expect that only an ideal combination of these options provide the most efficient result. Moreover, we assume that one can find the optimal combination by modifying each property more or less separately. This will be done in the next sections. At last we want to remark that, especially because of the properties of the reduction methods, the optimal approach still depends on the exact environment one has, and also on the type of parameter identification problem one solves. Therefore, we give instructions for when to use which combination.

## 13.1 Description of the Test–Set

We study the properties of the different versions of our parameter identification tool at two small problems in detail and additionally at the hexapod machine tool.

All of the examples use the same environment. We use Windows XP and Microsoft Visual Studio.Net and an Intel Xeon 2.4GHz processor. To solve the differential equation, we use an extension of MBSABM, an Adams–Bashforth method, which integrates the sensitivity differential equation along the main trajectory using IND. To avoid problems due to discretization errors of the integration process, we choose a relative tolerance of $10^{-8}$ and an absolute tolerance of $10^{-10}$ as the integration tolerances.

The following examples are used with varying initial guesses for the parameters. The initial guesses for dynamic variables at the intermediate multiple shooting nodes are calculated by an initial integration, i.e. we start with a continuous initial trajectory, which is far off the solution trajectory. As a result we might get an initial trajectory far of the solution trajectory. This situation is well suited to study different approaches, which try to treat bad initial guesses for mechanical systems.

**Single Pendulum** The first example is the previously defined single pendulum with revolute joint and a gravitational force. As a parameter we have the constant of the gravitation. Therefore, we have one kinematic degree of freedom accounting to a two–dimensional consistent manifold and one parameter.

The time–interval is 0.5 seconds, and we measure the x– and z–coordinate of the center of mass of the body at ten time–points. The model is tested at generated data with a relative error of $1\%$, which are normally distributed.

We fix the initial dynamic variables according to the description in section 12.1. Furthermore, we use five equally distributed multiple shooting nodes. The tolerance of the GGN is 1.0e-3.

**Double Pendulum** The second model is a pendulum of two bodies which are connected by revolute joints. The bodies of this model have the same mass, center of mass and moments of inertia as the single pendulum. In addition to the gravitational force, we introduce a spring damper element which connects the point $\begin{pmatrix} 1.5 & 0.0 & 1.0 \end{pmatrix}$ of the inertial system and the center of mass of the second body. The normal length of the element is $1.0m$, the stiffness is $10.0\frac{N}{m}$ and the damping coefficient is $10.0\frac{Ns}{m}$. As in the last example we use the same initial positions and velocities as in the examples of section 12.1.

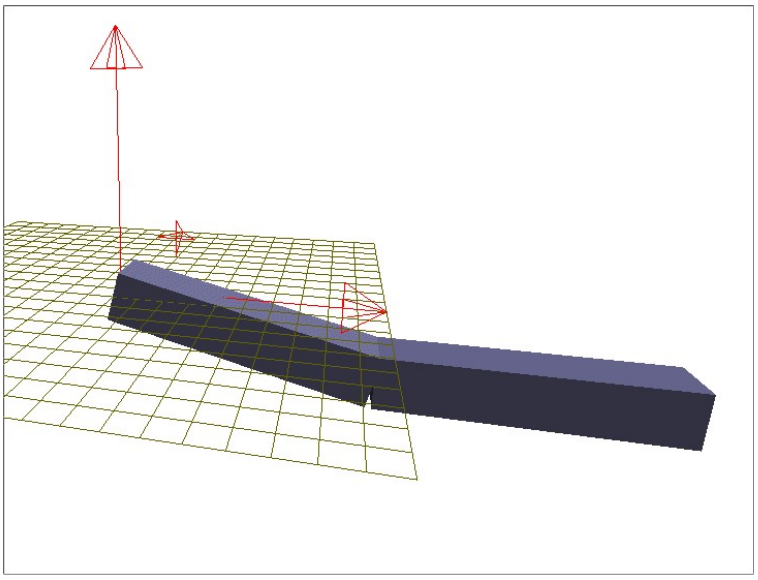


Figure 13.1: Double Pendulum

As parameters we have the three constants of the force–element, which leads to a four–dimensional consistent manifold and three parameters. The integration interval is the same as in the single pendulum case, but this time with 20 measurement points and a relative standard deviation of $1\%$. We measure the x– and z–coordinate of the center of mass of the second body. We again have five equally distributed multiple shooting nodes and a GGN tolerance of 1.0e-3.

**Hexapod Machine Tool** At last we consider the hexapod machine tool as described before. The hydraulic elements are simplified and modelled by springs of different stiffness. These stiffness constants are the parameters we want to identify. Together with the six kinematic degrees of freedom of the model, we get a 12–dimensional manifold and 6 parameters.

In the case of this highly complex model, we first use exact data to compare the different methods. Afterwards, we apply the best combination to a problem that uses generated data with normally distributed errors and a deviation of $1\%$. For all examples we study a time–interval of $0.5s$ and 20 measurement points. For this example, we measure the complete translational position of the center of mass of the load.

To solve this problem, we apply a multiple shooting discretization with 20 multiple shooting nodes, which are equally distributed. Additionally, we fix the initial dynamic states to the values shown in section 12.2. Furthermore, we ask for a GGN tolerance of 1.0e-3.

## 13.2 Comparison of Different Methods

The following subsections are used to study the effect of one set of variations of the parameter estimation algorithm on the convergence properties. In detail we examine:

1 different relaxation strategies

2 different reduction strategies

3 a comparison with consistent methods

4 different globalization strategies

5 initial state generation.

If not defined otherwise, we use

- the limit form of non–stiff Baumgarte relaxation with the projection adapted to mechanical systems
- apply the Fixfit–approach based on fixed initial dynamic variables
- use RMT with backprojection
- and do not apply initial value projection.

We assume the previously stated integration tolerance of $10^{-8}$ respectively $10^{-10}$ and a GGN tolerance of $10^{-3}$.

### 13.2.1 Relaxation Strategies

We first want to study different relaxation strategies. As shown above, there are three principle options:

(1) standard relaxation (see equations (8.3,8.4))

(2) non–stiff Baumgarte relaxation with a relaxation factor of 0.01 (see equations (8.5,8.6),

(3) the limit form of non–stiff Baumgarte relaxation (see section 8.4).

In the latter case, we have to choose between three different projection methods:

(3a) fixing a subset of variables, and calculating the others by using the constraints[1] (see section 8.5.2),

(3b) minimizing the difference between the original and the new variables w.r.t. the Euclidean norm while fulfilling the constraints (see section 8.5.3),

(3c) first projecting the position variables using minimization in mass matrix norm, and then projecting the velocity variables while fixing the position variables (see section 8.5.4).

These five options are tested at both pendulums for different initial guesses of the parameters. We want to stress that, on the one hand, we assume very low measurement errors and, on the other hand, we choose initial parameters which are far from the solution. This should provide us with good information about the influence of the manifold and the corresponding closing of the matching conditions.

The following table shows the number of Gauß–Newton steps for different initial guesses of the parameters of the singular and double pendulum problem. If no convergence is achieved, we write an ”x”. The correct parameter for the gravitational constant is $9.81\frac{m}{s^2}$ in the case of the single pendulum, and $1m$, $10\frac{N}{m}$, $10\frac{Ns}{m}$ for initial length, stiffness and damping factor of the double pendulum. In the latter case, we choose to change the initial guesses for the parameters proportionally, and apply a common factor to all parameters. This factor is shown in the tabular. All these tests have been done using the Fixfit–approach, Restrictive Monotonicity Test with back projection and no initial value projection. The corresponding comparisons are presented in the next subsections.

[1]To get a valid subset we apply a QR–decomposition with pivoting and use the first indices.

**Single Pendulum**

| | Relaxation Strategy | | | | |
|---|---|---|---|---|---|
| Parameter | 1 | 2 | 3a | 3b | 3c |
| -10 | 1 | 1 | 1 | 1 | 1 |
| -30 | 2 | 2 | 3 | 3 | 2 |
| -50 | 4 | 4 | 4 | 4 | 3 |
| -70 | 5 | 5 | 4 | 4 | 3 |
| -100 | 5 | 5 | x | x | 4 |
| -150 | x | x | x | 9 | 7 |
| -200 | x | x | x | 8 | 6 |

**Double Pendulum**

| | Relaxation Strategy | | | | |
|---|---|---|---|---|---|
| Factor | 1 | 2 | 3a | 3b | 3c |
| 1.1 | 2 | 2 | 2 | 2 | 2 |
| 1.5 | 3 | 3 | 3 | 3 | 3 |
| 2 | 3 | 3 | 4 | 4 | 3 |
| 2.5 | 5 | 4 | 6 | 6 | 4 |
| 3 | 6 | 6 | x | x | 5 |
| 3.5 | x | 5 | x | x | 3 |
| 4 | x | 14 | x | x | 4 |

Only the method using the limit form of non–stiff Baumgarte stabilization with the projection adapted to mechanical systems converges for all initial guesses in this broad region. Besides this, it needs a less or equal number of steps for initial guesses than other methods, if they converge, too. This result holds true for both examples.

### 13.2.2 Reduction Strategies

The same parameter estimation problem as above is solved for different reduction strategies:

(1) Fixfit–approach (see section 9.1),

(2) Invariant based reduction method for derivative generation only (see section 9.2.1),

(3) Invariant based reduction method for derivative generation and condensing (see section 9.2.2)

Due to the experiences from the previous section, we choose the limit form of the non–stiff Baumgarte relaxation with the projection adapted to mechanical systems (relaxation method $3c$) for the Fixfit approach and the first invariant–based method (method (2) of this study). Due to the structure of the projection, we have to choose a different relaxation technique for the second invariant–based method (method (2) of this study). We choose the limit form of the non–stiff Baumgarte with minimization as projection (relaxation method (3b)). We use RMT with back projection and again no initial value projection.

To compare the reduction methods, we give the results for two versions of the Fixfit approach (see section 13.2.1):

(1a) Fixfit with limit form of relaxation and mechanically adapted projection (previously method 3c),

(1b) Fixfit with limit form of relaxation and Euclidean projection (previously method 3b).

In contrast to the previous test, we will get a less distinct difference between the methods. Therefore, we have documented a larger sample on the next page.

We first want to compare the results of method (1a) and (2) of this section, i.e. Fixfit and the first invariant–based reduction method, each using the limit form of Baumgarte stabilization and the mechanically adapted projection. The only theoretical difference of these two methods is the choice of the initial directions for the VDE at the start of each multiple shooting interval (see 9.3. As the numerical results show, this does not lead to an eminent difference of the methods.

Next, we compare Fixfit to the second invariant–based method (3). Both methods are based on the limit form of non–stiff Baumgarte with a minimization in Euclidean norm. In this case, the invariant based methods achieve better results. It is more robust for the single pendulum and typically needs a smaller number of iterations. Here again we see the influence of the initial directions. Because we apply the less adapted method to generate initial directions for each interval independently in the case of second invariant based method, the performance is slightly better than in the case of Fixfit. Unfortunately, this gain in robustness is less if we compare it to the usage of the adapted version of the projection.

**Single Pendulum**

| | Reduction Method | | | |
|---|---|---|---|---|
| Parameter | 1a | 1b | 2 | 3 |
| -10 | 1 | 1 | 1 | 1 |
| -20 | 2 | 2 | 2 | 2 |
| -30 | 2 | 3 | 2 | 3 |
| -40 | 3 | 4 | 3 | 3 |
| -50 | 3 | 4 | 3 | 3 |
| -60 | 3 | 4 | 3 | 3 |
| -70 | 3 | 4 | 4 | 4 |
| -80 | 4 | x | 5 | 5 |
| -90 | 4 | 5 | 4 | 4 |
| -100 | 4 | x | 3 | 3 |
| -110 | 4 | 7 | 4 | 4 |
| -120 | 4 | x | 5 | 5 |
| -130 | 4 | x | 6 | 6 |
| -140 | 4 | 7 | 5 | 5 |
| -150 | 7 | 9 | 6 | 6 |
| -160 | 5 | 9 | 6 | 6 |
| -170 | 6 | 6 | 5 | 5 |
| -180 | 7 | 7 | 6 | 6 |
| -190 | 7 | 8 | 6 | 6 |
| -200 | 6 | 8 | 10 | 10 |

**Double Pendulum**

| | Reduction Method | | | |
|---|---|---|---|---|
| Factor | 1a | 1b | 2 | 3 |
| 1.1 | 2 | 2 | 1 | 2 |
| 1.3 | 2 | 2 | 2 | 2 |
| 1.5 | 3 | 3 | 3 | 3 |
| 1.8 | 3 | 4 | 3 | 4 |
| 2 | 3 | 4 | 3 | 4 |
| 2.3 | 4 | 5 | 4 | 4 |
| 2.5 | 4 | 6 | 4 | 5 |
| 2.8 | 5 | 6 | 5 | 5 |
| 3 | 5 | x | 3 | 6 |
| 3.3 | 3 | 8 | 3 | x |
| 3.5 | 3 | x | 3 | x |
| 3.8 | 3 | x | 4 | x |
| 4 | 4 | x | 4 | x |

As a result, we are able to state that using either the Fixfit–approach or an invariant–based reduction method does almost not change the robustness or efficiency of the optimization algorithm. The slight difference in the choice of the initial directions does not have a strong influence. Therefore, we propose to use Fixfit for problems with fixed initial values. In the case of free initial values, this advantage does not hold any more. A drawback of the Fixfit–approach on parallel machines is that one is not able to use the structure originating from the multiple shooting discretization for parallelization. Due to the limitations of the reduction method II with respect to the projection method, we only suggest to use this method for problems with a high number of bodies. In this case, the gain of performance due to the faster condensing algorithm may offset the disadvantage mentioned above.

### 13.2.3 Comparison to Consistent Methods

One of the differences between methods based on relaxation and methods which are consistent is the fact that the state variables at each multiple shooting interval are consistent during the complete optimization procedure. Even when using the limit form of non–stiff Baumgarte stabilization, where the states are consistent w.r.t. the original manifold during the integration, we have inconsistent initial values, i.e. inconsistent optimization variables.

Therefore, we apply the following additional modifications to compare to consistent methods:

- We use a nonlinear increment by projecting the calculated new state to the manifold. Thus, we always have consistent optimization variables.

  $$x^{k+1} = \pi(x^k + \alpha_k \Delta x_k)$$

  $\pi$ is the ideal projection to the manifold and $\alpha_k$ the steplength. This projection is done for the test step of the globalization strategy, too.

- The part of the increment corresponding to projection $\pi$ does not go into the estimation of the nonlinearity $\omega$ which is used for the globalization strategy of the GGN–method.

We compare this modification to all three projection types of the limit form of non–stiff Baumgarte. Therefore, we have to use the following methods:

(1) We fix a subset of variables and calculate the others using the constraints.

(2) We minimize the difference between the original variables and the new ones w.r.t. the Euclidean norm while fulfilling the constraints.

(3) First, we project the position variables using minimization w.r.t. the mass matrix norm, and then we project the velocity variables using the same method while fixing the position variables.

each either

(a) based on relaxation

(b) or using the consistent version.

Based on Fixfit and the restrictive monotonicity test with back projection, we get:

**Single Pendulum**

| | Method | | | | | |
|---|---|---|---|---|---|---|
| Parameter | 1a | 1b | 2a | 2b | 3a | 3b |
| -10 | 1 | 1 | 1 | 1 | 1 | 1 |
| -30 | 2 | 2 | 3 | 4 | 2 | 4 |
| -50 | 3 | 3 | 4 | 4 | 3 | 4 |
| -70 | 3 | 3 | 4 | 4 | 4 | 4 |
| -100 | 4 | 4 | x | x | 3 | x |
| -150 | 7 | x | 9 | x | 6 | x |
| -200 | 6 | 6 | 8 | x | 10 | x |

**Double Pendulum**

| | Method | | | | | |
|---|---|---|---|---|---|---|
| Factor | 1a | 1b | 2a | 2b | 3a | 3b |
| 1.1 | 2 | 1 | 2 | 1 | 2 | 1 |
| 1.5 | 3 | 3 | 3 | 3 | 3 | 3 |
| 2 | 4 | 3 | 4 | 4 | 3 | x |
| 2.5 | 6 | x | 6 | x | 4 | x |
| 3 | x | x | x | x | 5 | x |
| 3.5 | x | x | x | x | 3 | x |
| 4 | x | x | x | x | 4 | 4 |

The consistent method is less robust and needs slightly more iterations to converge, in case it converges in both examples. Therefore, we can say that it is advisable to use inconsistent optimization variables, i.e. relaxation.

### 13.2.4 Globalization Strategies

At last we study the influence of the globalization strategy, as described in section 1.3.2, on parameter estimation problems based on mechanical DAE using Natural Coordinates. Neglecting a full–step method, we have to choose between

(1) the Classical Monotonicity Test

(2) the Restrictive Monotonicity Test (RMT)

each

(a) without back projection

(b) with back projection

Numerical tests based on Fixfit with the limit form of non–stiff Baumgarte relaxation and the projection adapted to mechanical systems lead to the following results:

**Single Pendulum**

| | Globalization Method | | | |
|---|---|---|---|---|
| Parameter | 1a | 2a | 1b | 2b |
| -10 | 1 | 1 | 1 | 1 |
| -30 | 3 | 3 | 2 | 2 |
| -50 | 4 | 4 | 3 | 3 |
| -70 | 5 | 4 | 3 | 3 |
| -100 | 19 | 8 | 4 | 4 |
| -150 | x | 10 | 11 | 7 |
| -200 | 16 | 8 | 12 | 6 |

**Double Pendulum**

| | Globalization Method | | | |
|---|---|---|---|---|
| Factor | 1a | 2a | 1b | 2b |
| 1.1 | 3 | 2 | 2 | 2 |
| 1.5 | 4 | 5 | 3 | 3 |
| 2 | 5 | 5 | 4 | 3 |
| 2.5 | 6 | 6 | 4 | 4 |
| 3 | x | x | 6 | 5 |
| 3.5 | x | x | 5 | 3 |
| 4 | x | x | 4 | 4 |

First comparing the classical monotonicity test to the restrictive monotonicity test, we see that typically the RMT is better. The performance is further improved by a back projection step. According to the theory of the RMT, the solution of the parameter estimation problem is equivalent to the solution of the Davidenko differential equation. If the initial values of this differential equation, i.e. the initial values of the optimization problem, are consistent, using back projection better we better follow this trajectory, i.e. stay consistent during the optimization. This seems to drastically improve the performance of the algorithm. As in the cases before, this is seen in a smaller or equal number of steps and a larger convergence radius. This seems to emphasize the need for a good approximation of the consistent manifold while still using inconsistent variables during the optimization process.

### 13.2.5 Initial State Generation

At last we study the effect generated by the use of the algorithm to generate good initial guesses using measurement information as described in section (3.2), in contrast to initial guesses generated by pure integration. We study the major options:

(1) Fixfit with the limit form of non–stiff Baumgarte relaxation and projection adapted to mechanical systems

(2) Fixfit with the limit form of non–stiff Baumgarte relaxation and Euclidean projection

(3) First invariant–based method with the limit form of non–stiff Baumgarte relaxation and mechanically adapted projection

(4) Second invariant–based method with the limit form of non–stiff Baumgarte relaxation and Euclidean projection

either

(a) without initial value projection

(b) or with initial value projection.

Based on the same procedure as in the previous test, we get:

**Single Pendulum**

| | Method | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Parameter | 1a | 1b | 2a | 2b | 3a | 3b | 4a | 4b |
| -10 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| -30 | 2 | 2 | 3 | 2 | 2 | 2 | 3 | 2 |
| -50 | 3 | 3 | 4 | 3 | 3 | 3 | 3 | 3 |
| -70 | 3 | 3 | 4 | 5 | 4 | 4 | 4 | 4 |
| -100 | 4 | 4 | x | 6 | 3 | x | 3 | x |
| -150 | 7 | 4 | 9 | x | 6 | x | 6 | x |
| -200 | 6 | 6 | 8 | 9 | 10 | 6 | 10 | 6 |

**Double Pendulum**

| | Method | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Factor | 1a | 1b | 2a | 2b | 3a | 3b | 4a | 4b |
| 1.1 | 2 | 1 | 2 | 1 | 1 | 1 | 2 | 1 |
| 1.5 | 3 | 2 | 3 | 2 | 3 | 2 | 3 | 2 |
| 2 | 3 | 3 | 4 | 3 | 3 | 3 | 4 | 4 |
| 2.5 | 4 | 6 | 6 | 4 | 4 | 6 | 6 | 4 |
| 3 | 5 | 8 | x | 5 | 3 | 12 | x | 5 |
| 3.5 | 3 | 13 | x | 5 | 3 | 13 | x | 6 |
| 4 | 4 | x | x | x | 4 | x | x | x |

In contrast to the results shown in [Jac01], we are not able to see a *generally* better performance of the methods based on initial value projection for these parameter estimation problems based on mechanical DAE. We recognize a slightly better performance for good initial guesses of the parameters, as we would expect, but some convergence difficulties or larger number of steps, if the initial guesses are far of the solution. Therefore, we do favor one of the two methods.

## 13.3 Complex Example: Hexapod Machine Tool

The preliminary performance tests of the previous sections w.r.t. small models are now approved for more complex model of the hexapod machine tool, which is a model with a high number of closed kinematic loops. We focus on the most promising approaches especially w.r.t. the reduction method. We choose Fixfit as the best option in the serial case with fixed initial values.

We want to study the following options:

- the relaxation strategy
  - standard relaxation

  - non–stiff Baumgarte relaxation
  - limit form of non–stiff Baumgarte relaxation,

- the projection method
  - projection using Euclidean norm
  - mechanically adapted projection
- the reduction method
  - Fixfit
  - reduction method based on invariants (method I)
- the globalization strategy
  - monotonicity test
  - monotonicity test with back projection (BP) step
  - Restrictive Monotonicity Test
  - RMT with back projection
- the consistent optimization variables ($^{*1}$)
- the initial value projection ($^{*2}$)

| No. | Relaxation | Projection | Reduction | Globalization | Misc. |
|---|---|---|---|---|---|
| 1 | standard | – | Fixfit | RMT with BP | |
| 2 | Baumgarte | – | Fixfit | RMT with BP | |
| 3 | limit Baumgarte | Euclidean | Fixfit | RMT with BP | |
| 4 | limit Baumgarte | Mechanic | Fixfit | RMT with BP | |
| 5 | limit Baumgarte | Mechanic | Invariants | RMT with BP | |
| 6 | limit Baumgarte | Mechanic | Fixfit | RMT | |
| 7 | limit Baumgarte | Mechanic | Fixfit | Monotonicity | |
| 8 | limit Baumgarte | Mechanic | Fixfit | Mono. with BP | |
| 9 | limit Baumgarte | Mechanic | Fixfit | RMT with BP | $^{*1}$ |
| 10 | limit Baumgarte | Euclidean | Fixfit | RMT with BP | $^{*1}$ |
| 11 | limit Baumgarte | Mechanic | Fixfit | RMT with BP | $^{*2}$ |
| 12 | limit Baumgarte | Euclidean | Fixfit | RMT with BP | $^{*2}$ |

We use the following set of initial guesses for the parameters:

| | Method Combination Number | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | *4* | *5* | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| +1 | 1 | 1 | 1 | *1* | *1* | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| +20 | 1 | 1 | 2 | *1* | *1* | 2 | 1 | 1 | 1 | 1 | 1 | 1 |
| +50 | 2 | 1 | 3 | *2* | *2* | 2 | 2 | 2 | 1 | 1 | 2 | 2 |
| +100 | x | x | 4 | *2* | *2* | 3 | 2 | 2 | 2 | 2 | 3 | 2 |
| 600 | x | x | x | *4* | *4* | 5 | 4 | 4 | 5 | 5 | x | 4 |
| ±5 | x | x | x | *4* | *4* | 6 | 6 | 6 | 5 | 6 | x | 8 |

where we use abbreviations for the following sets of parameters.

| | $p_0$ | $p_1$ | $p_2$ | $p_3$ | $p_4$ | $p_5$ |
|---|---|---|---|---|---|---|
| Exact | 400 | 500 | 600 | 700 | 800 | 900 |
| +1 | 401 | 501 | 601 | 701 | 801 | 901 |
| +20 | 420 | 520 | 620 | 720 | 820 | 920 |
| +50 | 450 | 550 | 650 | 750 | 850 | 950 |
| +100 | 500 | 600 | 700 | 800 | 900 | 1000 |
| 600 | 600 | 600 | 600 | 600 | 600 | 600 |
| $\pm 5$ | 590 | 595 | 600 | 605 | 610 | 615 |

The results from this more complex test show the following:

- It is best to use the limit form of non–stiff Baumgarte relaxation based on the ideal projection adapted to mechanical systems.
- A comparison of the Fixfit–approach to the corresponding reduced method based on invariants again shows that the methods are comparable providing the same results in this example.
- The question of how to use relaxation is answered in favor to inconsistent variables if the limit form of non–stiff Baumgarte with adapted projection is used.
- A projection of the initial states due to measurement information is discouraged due to worse results for high deviations of the initial parameter values.
- Concerning the globalization strategy, we again suggest to use RMT with back projection.

These results approve the conclusion we did based on the simple examples.

At last we want to present a parameter estimation based on the same model, but with a relative measurement error of $1\%$. The measurement error has zero mean and is normally distributed. We use the combination found to be best in the previous example: the limit form of non–stiff Baumgarte relaxation based on the projection which is adapted to mechanical systems, Fixfit–approach, RMT with back projection step and no initial value projection.

After 10 iterations and using the stated initial guesses, we get the following parameters including standard deviation.

| Parameter | Initial Guess | Exact Value | Estimated Value | |
|---|---|---|---|---|
| $p_0$ | 590 | 400 | 400.6 | $\pm 0.9$ |
| $p_1$ | 595 | 500 | 500.7 | $\pm 0.7$ |
| $p_2$ | 600 | 600 | 600.4 | $\pm 0.5$ |
| $p_3$ | 605 | 700 | 700.4 | $\pm 0.9$ |
| $p_4$ | 610 | 800 | 799.9 | $\pm 0.4$ |
| $p_5$ | 615 | 900 | 900.0 | $\pm 0.7$ |

The result was calculated on a Athlon 1.2GHz with the same software environment and setting of the integration scheme as the previous results. The calculation took 6.3 minutes.

# Chapter 14

# Parameter Estimation for a Biomechanical Model of a Sitting Human

We study a parameter estimation of a human modelled using MBSNAT. We are interested in the influence of a vertical excitation on a human sitting on the seat of a car. Therefore, we focus on a detailed modelling of the spinal column, using five segments which are connected by bushings modelling intervertebral disks. Although we focus on the spinal column, we do not want to model all the vertebras, and we want to keep the other part of the model as simple as possible. In spite of these simplifications, we get a complex three–dimensional model of high dimension. The efficient modelling and simulation of this model is the topic of the first section of this chapter.

As we have to use a heavily idealized model of the biological system, it is difficult to assess the validity of the data. The second section shows a parameter estimation of this dynamic model which estimates the parameters of internal and idealized force–elements. This is done based on measurements of the position of markers stuck to the human – a usual method found in modern laboratories for motion analysis.

## 14.1 Modelling and Simulation of a Human

First it is necessary to describe the idealized physical model. Because of the complexity of this and the difficulty to get good initial guesses for states and parameters describing the kinematic and dynamic properties of the model, we use Calcman3D as described in section 6.1. This tool is able to generate a description based on some simple input data like height, sex and body mass. We study a sitting human with a more detailed segmentation of the spinal column. This leads to the modelling of the following segments as rigid bodies:

The segments of the spine are: head, neck, thorax, lumbus and pelvis. The left

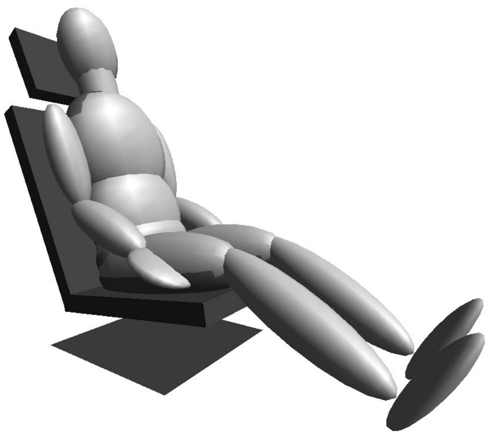


Figure 14.1: Model of a Human

and the right arm are modelled as upper and lower arms together with the hands, and the legs are modelled as thighs, shanks and feet. Furthermore, we introduce wobbling masses to model soft tissue which is connected to the skeletal system. This leads to additional bodies connected to the thorax, lumbus and pelvis as well as both thighs. Moreover, we model the technical part as a huge car, seat–chassis and seatback–chassis which are connected to the corresponding cushion in form of the seat and the seatback, and a neck–protection element. All in all, we get a model using 22 bodies for the biomechanical part and 6 bodies to model the technical part. An OpenGL–visualization of this model can be seen in figure 14.1.

While the main body is only connected by bushings, i.e. a special kind of tensorial forces, the limbs are connected by joints. We use spherical joints for the shoulder, hip and ankle joint, revolute joints for the elbow and knee–joint, and we fix the hand to the lower arm. Additionally, we use revolute joints to connect the seat–chassis, the seatback–chassis and the neck–protection, and translation joints to connect the seat to the seat–chassis, the seatback to the seatback–chassis, and the car to the ground in order to allow a vertical excitation. The seat–chassis is fixed to the car. This definition of the joints leads to a free floating bio–mechanical model with 77 kinematic degrees of freedom and a technical part with five degrees of freedom. We want to stress that the most interesting part – the main body and the wobbling masses – typically consists of bodies that still have all six degrees of freedom. Therefore, a modelling approach based on local coordinate systems as provided by Natural Coordinates is well suited.

To complete the modelling process, we have to add force–elements to this kinematic model. We passively control each revolute or translational joint using the 3D–variant of a spring–damper element. The five segments of the spinal column are connected using bushing elements as described in section 6.2.2 to model the intervertebral disks. Furthermore, we connect the wobbling masses to the corresponding bones by using a tensorial force as described in section 6.2.3. Especially the bushing elements and the latter forces are characterized by high stiffness coefficients in order to provide a certain, small flexibility. This results in a high mechanical stiffness of the differential equation leading to long integration times despite the efficient numerical algorithms and modelling techniques. At last, we connect the biomechanical model to the technical system using point–point Hertz type force–elements as described in section 6.3. As it is possible to gain or loose contact during the simulation, and because the force itself is only defined piecewise constant, we have to provide additional elements which signal events to modify the behavior of the forces accordingly and provide the necessary information for the event–handling algorithms of the integration scheme. Additionally, we have to provide the gravitational force, such that, altogether, we use 43 force–elements.

As a result, the model has 672 dynamic variables or 82 kinematic degrees of freedom. It consists of 42 force–elements, some of which are tensorial forces, some are characterized by high stiffness coefficients, and some are discontinuous.

We first study this model in a simulation context. We start with a configuartion shown in picture (14.1) and zero velocity. We simulate a time–interval of 5.0 seconds, and apply a periodic vertical excitation to the car of 10 cm and with a frequency of 20 Hz. The simulation takes 10 seconds on a machine with Athlon 1.2GHz processor running Windows XP and using Microsoft Visual C++.Net as compiler. We use MBSABM of MBSSIM as an integration scheme to solve the provided differential equation with a relative tolerance of $10^{-6}$ and an absolute tolerance of $10^{-8}$. To get an idea of the movement, a set of snapshots is shown in figure 14.2.

## 14.2 Parameter Estimation of Biomechanical Models

After studying the model in the simulation context, we consider a parameter estimation problem based on this model.

Considering the huge amount of possible parameters, it is obviously not possible to identify all the parameters. Therefore, we focus on 10 internal parameters, which correspond to force constants of heavily idealized elements or which are delicate to measure. These are bushings, forces connecting the wobbling mass to the skeleton, or 3D–forces representing passively acting muscles. A list of these parameters is shown in table 14.2. The initial values of these parameters and the values of the other parameters are provided by Calcman3D and as described in section 6.1. Based on these values, we changed the initial guesses according to table 14.2.

We assume a measurement setting that conforms with a typical situation found in modern laboratories, where several calibrated cameras are used to measure the 3D–position of 10 markers stuck to the test person, and we measure for 0.2 seconds.

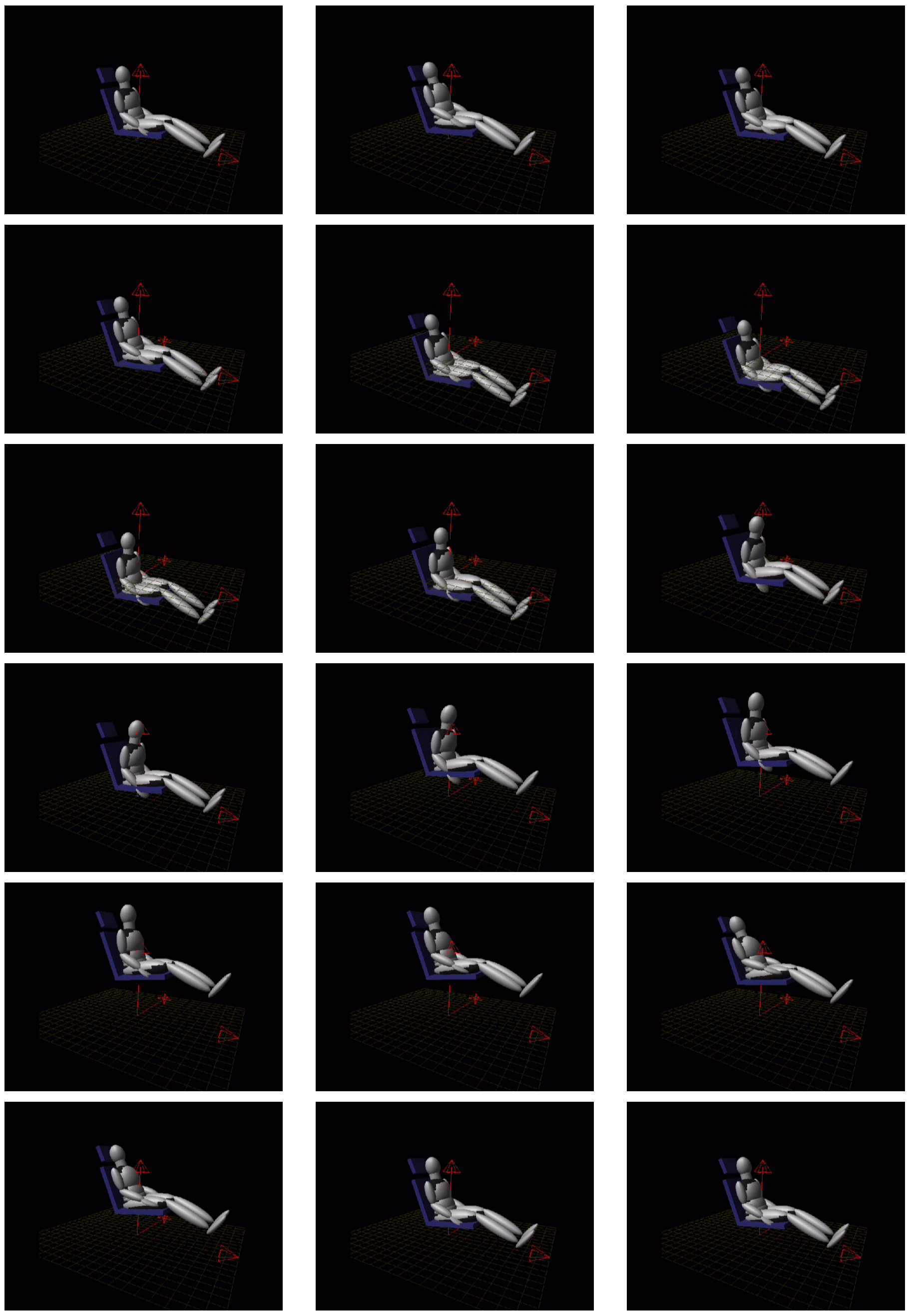


Figure 14.2: Snapshots of a human in a seat with a strong vertical excitation (Please, look at the pictures line for line.)

Table 14.1: List of Markers

| Location | Body | Position | | |
|---|---|---|---|---|
| Breast | Thorax | 0.08 | 0.0 | 0.0 |
| Right Shoulder | Thorax | 0.0 | 0.16 | 0.14 |
| Left Shoulder | Thorax | 0.08 | -0.16 | 0.14 |
| Right Hand | Right Hand | 0.025 | 0.0 | 0.0 |
| Left Hand | Left Hand | 0.025 | 0.0 | 0.0 |
| Right Hip | Pelvis | 0.0 | 0.17 | 0.0 |
| Left Hip | Pelvis | 0.0 | -0.17 | 0.0 |
| Right Thigh | Right Thigh | 0.0 | 0.0 | 0.25 |
| Left Thigh | Left Thigh | 0.0 | 0.0 | 0.25 |
| Forehead | Head | 0.08 | 0.06 | 0.08 |

The cameras typically have a measurement frequency of 120 Hz and the position of the markers are measured with an accuracy of 1 mm. The error is mainly due to the fact that the markers are stuck to the skin, which moves relative to the bones. The position of the markers can be found in table 14.1. We use the z–position of the markers at all measurement time–points.

First studies show that the information provided is not sufficient to accurately estimate the desired parameters. One possibility would be to use a longer time–interval and, therefore, more measurements. We want to study a different approach based on multiple experiments. In this setting we combine the measurements of eight independent experiments with the same set–up to increase the amount of information available.

Using this information, we get a high–dimensional, dynamic parameter estimation problem which consists of 26890 optimization variables, 26880 constraints and 1920 least squares conditions.

To test our approach, we vary the exact parameters as shown in table 14.2, and we apply the parameter estimation algorithm to our generated data. We choose the method best suited for such a problem based on the knowledge of chapter 13. Due to the fact that all initial dynamic states are fixed, we use Fixfit approach. Furthermore, we apply the limit form of non–stiff Baumgarte relaxation using the projection adapted to mechanical systems. We do no initial value projection and apply the Restrictive Monotonicity Test with back projection as a globalization strategy of the Generalized Gauß–Newton method. Furthermore, we use four multiple shooting intervals per experiment.

This problem has been calculated on a computer with a Pentium IV 2.4GHz processor running Windows XP and using Visual C++.Net as compiler. After 7 iterations calculated with a relative integration tolerance of $10^{-8}$ and an absolute tolerance of $10^{-10}$, which takes approximately 4.5 hours, we achieve the result shown in table 14.2.

Table 14.2: Selection of Internal Parameters

| Parameter | | Exact | Initial | Solution |
|---|---|---|---|---|
| Bushing (stiff.) | Pelvis–Lumbus | $0.33$ | $0.67$ | $0.33 \pm 0.03$ |
| | Lumbus–Thorax | $1.0$ | $2.5$ | $1.15 \pm 0.20$ |
| Wobble (tr. stiff.) | Pelvis | $1.0E^7$ | $1.3E^7$ | $1.01E^7 \pm 0.01E^7$ |
| | Lumbus | $1.0E^7$ | $1.3E^7$ | $0.94E^7 \pm 0.07E^7$ |
| | Thorax | $1.0E^7$ | $1.3E^7$ | $1.04E^7 \pm 0.05E^7$ |
| Wobble (tr. damp.) | Lumbus | $100$ | $150$ | $98 \pm 12$ |
| Spring (rot. stiff.) | Ellbow (left) | $50$ | $75$ | $32 \pm 9$ |
| | Ellbow (right) | $50$ | $75$ | $49 \pm 5$ |
| Spring (rot. damp.) | Ellbow (left) | $4.0$ | $6.5$ | $6.6 \pm 1.3$ |
| | Ellbow (right) | $4.0$ | $6.5$ | $4.1 \pm 0.7$ |

The original parameters are reproduced if we take into account the estimated standard deviation of the solution. For additional reading see also [KMB04]

# Chapter 15

# Conclusion and Outlook

## 15.1 Conclusion

The aim of this work was to develop numerical methods to identify the parameters of internal elements of a model of a human, which are difficult to estimate otherwise. The alternative methods that are used up to now, comprise in vivo measurements based on X–ray examinations, invasive surgery and corpse studies. Furthermore, these methods are not able to determine the properties of models that idealize the reality by reducing the number of bodies to a manageable level. In contrast to that, our approach is based on measurement data like the position of markers stuck to the body. These data are much easier to achieve and avoid invasive measurements at the human body.

As we examine the effect of vibrations to the human body, we model the spine in more detail (i.e. we use four segments) as well as the corresponding intervertebral disks and the soft tissue represented by wobbling masses. The resulting rigid body model is of high dimension – 672 dynamic variables and 82 kinematic degrees of freedom – and includes force–elements with high stiffness coefficients. Therefore, it is already challenging in the simulation context.

We present a parameter estimation based on such a complex biomechanical model. Using the methods developed in this thesis, we have been able to identify a selection of ten internal force–constants. To increase the accuracy of the estimated parameters, we have combined the measurement information of eight independent experiments. This results in a very high–dimensional nonlinear parameter estimation problem with 26’890 optimization variables, 26’880 constraints and 1’920 least squares conditions.

The solution of this complex parameter estimation problem has only been possible by providing a tool that is flexible and simple to use, and which provides the solution of the optimization problem in a fast and reliable way. This has been achieved by optimizing each building block – modelling, integration and parameter estimation – in this context independently and – what is even more important – by adapting the components to each other. The most important of these modifications and developments are summarized in the following:

- We provide an efficient modelling of complex high–dimensional 3D rigid body models in Natural Coordinates, which can be evaluated in linear complexity w.r.t. the number of bodies.
- Natural Coordinates in our version lead to simple equations and additional structure which we exploit.
- Using coordinate partitioning developed by us, we are able to easily model arbitrary topologies.
- We provide a fully parameterized model evaluation, which allows for an interactive assembly of a model. This includes the definition of parameters without changing the previously defined model.
- The evaluation of directional derivatives is of linear complexity w.r.t. the number of bodies and the number of directions, whereas the evaluation of one direction is approximately two to three times faster than the evaluation of the differential equation. This derivative information is provided automatically.
- We have developed different reduced methods based on constraints of the optimization problem corresponding to the invariants of the underlying DAE to minimize the number of directional derivatives of the differential equation we have to calculate explicitly.
- We have developed the limit form of non–stiff Baumgarte relaxation in combination with a projection which is adapted to mechanical systems. Numerical results show the high performance of this approach compared to standard methods either using relaxation or working with non–redundant optimization variables.

One the other hand, it is not enough to only develop these numerical methods to treat such a complex task. Of equal importance is an efficient and flexible software that is easy to extend. Therefore, we have designed the object–oriented packages MBSNAT and Parfit++, which implement these numerical algorithms, according to up–to date software engineering approaches:

- three–tier model
- levelization
- interface–oriented programming.

The result is an object–oriented environment, which is able to efficiently treat the complete framework of modelling, simulation and parameter estimation for biomechanical problems.

## 15.2 Outlook

In this thesis, we have demonstrated the solution of a parameter estimation problem based on a large and complex biomechanical model. In order to better use such methods in practice, and in order to improve the possibility of estimating the parameters, we propose several future extensions and developments:

- Due to the high computational effort, we need parallelization in order to better allow for an interactive work, and to study more detailed models. Fortunately, the algorithm allows for the efficient use of a parallel computer for different granularity. It is possible to parallelize the generation of derivatives and the structure originating from multiple experiment problems. Due to the invariant–based reduced methods, it is also possible to parallelize on the level of the multiple shooting discretization.

- Based on the measurement setting shown here, it is not possible to estimate all the parameters (approximately 1000) of the biomechanical model. Therefore, the possibilities of the multiple experiment approach should be used to introduce more and different measurement settings. The amount of additional information provided by an additional measurement setting can be estimated based on sensitivity analysis.

- Optimal experimental design can be applied to automatically choose suitable experiments and optimize the experimental setting of them. With such an optimization it is possible to find e.g. an optimal placement of the markers or an optimal excitation, such that the statistical properties of the solution are improved.

- The biomechanical model shown here is not controlled, i.e. it uses only passive muscles. If one wants to include the important class of experimental settings, which are based on e.g. walking humans, this step is very important. One possibility is to parameterize the controls and estimate them together with the other parameters. This would introduce a huge amount of additional parameters. Another option is to include a model of how the movement is controlled. In this case, a lower number of possible parameters are introduced.

# Appendix A

# Table of Symbols

## Optimization

### General

| | |
|---|---|
| $x$ | all $n$ optimization variables |
| | |
| $r_1$,$J_1$ | $n_1$ least squares conditions (and jacobian) |
| $r_2$,$J_2$ | $n_2$ equality constraints (and jacobian) |
| $r_3$,$J_3$ | $n_3$ inequality constraints (and jacobian) |
| $r_c$,$J_c$ | $n_c$ equality and active inequality constraints (and jacobian) |
| $r$,$J$ | all $n_{nbed}$ conditions (and jacobian) |
| | |
| $x_i$ | optimization variables of ith step |
| $\Delta x_i$ | increment of optimization variables in ith step |
| $\alpha_i$ | damping factor of ith step |
| | |
| $o$ | observation function |
| $\eta$ | measurement data |
| $\sigma$ | standard deviation |

### Dynamic

| | |
|---|---|
| $t$ | time |
| $y$ | dynamic variables |
| $z$ | algebraic variables |
| $q$ | parameter |
| $f$ | right–hand wide of ODE |
| $g$ | constraints |
| $s$ | switching function |

For mechanical symbols and symbols concerning the variational differential equation we refer to the following.

### Multiple Shooting

| | |
|---|---|
| $\tau_i$ | time–points of the multiple shooting discretization |
| $s_i$ | initial values for multiple shooting intervals |
| $h_i$ | matching conditions |
| $D_i$ | jacobian matrix–block |
| $H_i$ | derivative of $h_i(x_i^e, s_i)$ w.r.t. $x_i^e$ |

## Mechanic

### Full Descriptor Form

| | |
|---|---|
| $e$ | variables of external (non–mechanical) dynamic |
| $p$ | position |
| $v$ | velocity |
| $a$ | acceleration |
| $\lambda$ | lagrange multiplier |
| $M$ | mass matrix |
| $G$ | constraint matrix |
| $f_3$ | forces |
| $g_{pos}$ | kinematic or position constraints (invariants) |
| $g_{vel}$ | velocity constraints (invariants) |
| $\gamma$ | acceleration independent part of acceleration constraint |

### Natural Coordinates

| | |
|---|---|
| $o$ | origin of affine coordinate system |
| $e_i$ | axis of affine coordinate system |
| $C_r$ | transformation matrix of natural coordinates |

### Constraint–Types

| | |
|---|---|
| $g^{RC}$,$G^{RC}$ | rigid body constraints and constraint matrix |
| $g^J$,$G^J$ | joint constraints and constraint matrix |

## Optimization of Mechanical Systems

### Variational Differential Equation

| | |
|---|---|
| $W$ | sensitivity of solution w.r.t. initial values |
| $W_p$ | position part of sensitivity matrix |
| $W_v$ | velocity part of sensitivity matrix |
| $W_q$ | parameter part of sensitivity matrix |
| $D$ | sensitivity matrix w.r.t. some direction |
| $d$ | one initial direction |
| $d_p$ | position part of initial direction |
| $d_v$ | velocity part of initial direction |
| $d_q$ | parameter part of initial direction |

### Relaxation

| | |
|---|---|
| $\tilde{g}$ | relaxed constraint (standard relaxation) |
| $\tilde{g}_B$ | relaxed constraint (non–stiff Baumgarte relaxation) |
| $G_v$ | derivative of velocity constraint w.r.t. position variables |
| $\hat{G}$ | derivative of position and velocity constraints w.r.t. states |
| $S$ | basis of the tangent space of the manifold |
| $B$ | basis of the Euklidean space consisting of $S$ and $\hat{G}$ |

# Appendix B

# Basic Interfaces

We show slightly simplified versions of interfaces which were referenced in this work. The interfaces presented here are used to support the most basic communication between the components, which in combination are Parfit++.

**Interface INewton** is used to support the *iterative* solution of a nonlinear problem.

```
class INewton
{
public:
      // initialize object
  virtual void Initialize() =0;
      // sets pointer to solver-component
  virtual void Solver(ISolver* solver) =0;
      // gets pointer to solver-component
  virtual ISolver* Solver() =0;
      // sets pointer to problem-component
  virtual void Problem(IProblem* problem) =0;
      // gets pointer to problem-component
  virtual IProblem* Problem() =0;
      // do optimization
  virtual void Optimize() =0;
      // access number of steps
  virtual int nSteps() const =0;
      // set number of miximal steps
  virtual void MaximumSteps(int max) =0;
      // access stop criteria
  virtual double Accuracy() const =0;
      // set tolerance
  virtual void Accuracy(double tol) =0;
};
```

**Interface ISolver** is used to provide the solution (and decomposition) of a linear system.

```
class ISolver
{
public:
      // initialize object
  virtual void Initialize() =0;
      // sets pointer to right--hand side
  virtual void SetRightHandSide(ISolver_RHS *rhs) =0;
      // decompose linear system
  virtual void Decompose() =0;
      // solve linear system
      // using out also as input, if external_rhs is set
  virtual void Solve(CVector<double> &out,
                     bool external_rhs) =0;
      // forward solution step
  virtual void Forward() =0;
      // backward solution step with same options as Solve
  virtual void Backward(CVector<double> &out,
                        bool external_rhs) =0;
};
```

**Interface ISolver_RHS** is used to supply a linearized system (typically used in combination with components which inherit from ISOLVER).

```
class ISolver_RHS
{
public:
      // initialize object
  virtual void Initialize() =0;
      // access right--hand side vector
  virtual const CVector<double> &RightHandSide() const=0;
      // set right--hand side vector
  virtual void RightHandSide(const CVector<double>&rhs)=0;
      // access jacobian matrix
  virtual const FMatrix<double> &Jacobian() const = 0;
      // sets jacobian matrix
  virtual void Jacobian(const FMatrix<double>& J) =0;
};
```

**Interface IProblem** is the interface to a nonlinear problem. This is a simplified version of the interface.

```
class IProblem
{
public:
      // initialize object
  virtual void Initialize() =0;
      // gets number of states
  virtual int nState() const =0;
      // gets number of equations
  virtual int nEquation() const =0;
      // access state
  virtual const CVector<double> &State() const =0;
      // update state
  virtual void State(const CVector<double>& state) =0;
      // do calculations
      // if flag set, calculate the linearization, too
  virtual void Calculate(bool withderivatives = true) =0;
};
```

**Interface IProblem_Leastsquares** is mixed to the basic interface IPROBLEM such that a problem inheriting both interfaces is able to treat nonlinear constrained least square problems.

```
class IProblem_LeastSquares
{
public:
      // initialize object
  virtual void Initialize() =0;
      // gets number of constraints
  virtual int nConstraints() const =0;
      // gets number of least square conditions
  virtual int nLeastSquares() const =0;
      // gets norm of least square conditions
  virtual double Residuum_Leastsquare() const =0;
      // gets norm of constraints
  virtual double Residuum_Constraint() const =0;
      // gets an pattern-vector concerning the type
      // 0: constraint
      // 1: least square condition
  virtual const CVector<int> &EquationTypePattern() const=0;
};
```

**Interface IIntegrator** is used to access the basic functionality of the integration schemes. This example is drastically modified and should only illustrate the interface.

```
class IIntegrator
{
public:
      // initialize object
  virtual void Initialize() =0;
      // sets pointer to model, i.e. the DE
  virtual IModel* Model() =0;
      // gets model--pointer
  virtual void Model(IModel* model) =0;
      // start integration of model
      // in time-interval [begin, end]
      // with sensitivities depending on withderiv
  virtual void Integrate(double begin,double end,
                       bool withderiv) =0;
};
```

**Interface IModel** is used to access a general model consisting of some state.

```
class IModel
{
public:
      // initialize object
  virtual void Initialize() =0;
      // access number of variables
  virtual int nVariable() const =0;
      // access variables
  virtual const CVector<double>& Variable() const =0;
      // update state
  virtual void UpdateVariable(
                  const CVector<double> &Variables) =0;
};
```

**Interface IModel_ODE** extends the interface IMODEL, if the model is a differential equation, and supplies necessary functionality.

```
class IModel_ODE
{
public:
      // access time
  virtual double Time() const =0;
      // access right-hand side
  virtual const CVector<double>& RightHandSide() const =0;
      // update time
  virtual void UpdateTime(double Time) =0;
      // calculate right-hand side
  virtual void CalculateRightHandSide() =0;
};
```

**Interface IEquation** is used for scalar functions. Several objects implementing a nonlinear problem contain of several such conditions.

```
class IEquation
{
public:
      // initialize object
  virtual void Initialize() =0;
      // access function evaluation
  virtual double Value() const =0;
      // access residuum r of  f(x)-r (=0)
  virtual double Residuum() const =0;
      // set residuum r
  virtual void Residuum(double value) =0;
      // evaluate function
      // if flag is set including the first derivative
  virtual void Calculate(bool withderivatives = false) =0;
};
```

We want to stress that these interfaces are only used to illustrate this writing and the basic concept of the components of Parfit++. Therefore, it is only a small selection of more than 100 interfaces. A thorough and complete list of the interfaces can be found in the INTERFACE–directory of the sources.

# Appendix C

# Implemented Elements

This appendix lists all mechanical elements implemented in MBSNAT together with a very short description. All elements are grouped corresponding to their base class. Therefore, we have constraints, joints that are defined by constraints and forces. Because the base class NBODY treats arbitrary rigid bodies, there are no special elements.

## C.1 Constraints

In this section we use the expression $b(.)$ to describe an affine point fixed to an rigid body respectively the inertial system or the velocity of this point depending on the argument. The argument is either the position–variables $p_i$ or velocity–variables $v_i$ of the first or second body. We use the notation $d(.)$ to define an translational vector fixed to the body. In the following table we give the ID of the constraints used for the XML–Input file together with the definition of the constraints using the notation introduced above.

| ID | description | | |
|---|---|---|---|
| $B0_i$ | $[b_1(p_1) - b_2(p_2)]_i - r$ | $= 0$ | $i \epsilon \{x, y, z\}$ |
| $DDX$ | $d_1(p_1)^T d_2(p_2) - r$ | $= 0$ | |
| $BBDX$ | $[b_1(p_1) - b_2(p_2)]^T d(p_1) - r$ | $= 0$ | |
| $BBX$ | $\frac{1}{2}[\|b_1(p_1) - b_2(p_2)\|^2 - r^2]$ | $= 0$ | |
| $V0$ | $[b_1(v_1) - b_2(v_2)]_i - r$ | $= 0$ | |
| $VVDX$ | $d(p_1)^T [b_1(v_1) - b_2(v_2)] - r$ | $= 0$ | |
| $VVX$ | $\frac{1}{2}[\|b_1(v_1) - b_2(v_2)\|^2 - r^2]$ | $= 0$ | |

## C.2 Joints

The joints are defined by a set of constraints. As the constraints are defined w.r.t. two markers, we give the axes of these markers which define the affine points $b$ and translational vector $d$ used in the definition of the constraints. The first line of constraints restricts translational and the second line rotational relative movement.

| ID | Constraints |
|---|---|
| Spherical | $B0_x$, $B0_y$, $B0_z$ |
| Revolute | $B0_x$, $B0_y$, $B0_z$, $DDX(x_1, z_2)$, $DDX(y_1, z_2)$ |
| Revolute_2 | $B0_x$, $B0_y$, $B0_z$, $DDX(x_2, z_1)$, $DDX(y_2, z_1)$ |
| Universal | $BBDX(p_1, p_2, x_1)$, $BBDX(p_1, p_2, y_1)$ $DDX(x_2, z_1)$, $DDX(y_2, z_1)$ |
| Prismatic | $BBDX(p_1, p_2, x_1)$, $BBDX(p_1, p_2, y_1)$ $DDX(x_2, z_1)$, $DDX(y_2, z_1)$, $DDX(x_1, y_2)$ |
| FixedDistance | $BBX(p_1, p_2)$ |
| Free | |

## C.3 Forces

All forces are specializations of the base class NFORCE. The forces are either external forces which affect all bodies according to their position and velocity, or force–elements which provide an interaction between two or more bodies. The latter are characterized by how they connect the bodies and which force–law they implement.

### External Forces

In this category we only provide the implementation of the gravitation.

| Force | Constants |
|---|---|
| Gravitation | gravitational constant |

### Force–Element

In case of forces interacting between two bodies, there are several possibilities how this interaction is done. We first omit the exact force–law. We get a number of specializations of the base class for force–elements: NFORCEELEMENT which again is a specialization of the base class of all forces NFORCE.

Linear These elements apply a force depending on the difference of two points $b_1(p_1)$ and $b_2(p_2)$ or the difference of velocity at these points $b_1(v_1)$ and $b_2(v_2)$, which are defined w.r.t. different bodies. The resulting force is applied along the axis defined by the difference of these two points.

$$f(\Delta p, \Delta v) = f(b_1(p_1) - b_2(p_2), b_1(v_1) - b_2(v_2))$$

This type of force is implemented in the base class NTRANSLATORIALFORCE.

Z In contrast to linear elements these forces only depend on the z–component w.r.t. the reference frame of the difference (position and velocity level) and only apply the force in this direction.

$$f(\Delta p, \Delta v) = f([b_1(p_1) - b_2(p_2)]^T z, [b_1(v_1) - b_2(v_2)]^T z)$$

The base class is NZFORCE.

Direction Another possibility is to define a force w.r.t. a specific direction $d(p_1)$ , which is formulated w.r.t. one body. This is a generalization of Z–Forces.

$$f(\Delta p, \Delta v) = f([b_1(p_1) - b_2(p_2)]^T d(p_1), [b_1(v_1) - b_2(v_2)]^T d(p_1))$$

The base class is NDIRECTIONFORCE.

Rotational This type of force–elements are sensitiv to a relative rotation around an axis $d(p_1)$.

$$f(\alpha, \dot{\alpha})$$

The corresponding base class is NROTATORIALFORCE.

Spherical These forces which consider all three translational and all three rotational degrees of freedom are discussed in section (6.2.1)

Additionally, there are the following force–laws depending on a translation or rotation on position and velocity level $f(p, v)$.

| Force | Force–Law |
|---|---|
| <Type>_Spring | $c_1(\|\Delta p\| - c_0)$ |
| <Type>_Damper | $c_0\|\Delta v\|$ |
| <Type>_Spring_Damper | $c_1(\|\Delta p\| - c_0) + c_2\|\Delta v\|$ |
| <Type>_Actuator | $c_0$ |
| General_<Type> | (template for general force–law) |

In this notation <Type> has to be set to the corresponding basic structure (like ”Z_Spring”) except for the linear forces where we omitted this information (e.g. ”Spring”).

# Bibliography

[AL94] W. H. Artl and A. C. Lehman. Sensitivity functions of a human head movement model. *Med. Eng. Phys.*, 1994.

[Ali92] T. Alishenas. *Zur numerischen Behandlung, Stabilisierung durch Projektion und Modellierung mechanischer Systeme mit Nebenbedingungen und Invarianten*. PhD thesis, NADA; KTH, S–100 44 Stockholm, Sweden, 1992.

[AP99] U. Ascher and L. Ping. Sequential Regularization Methods for Simulating Mechanical Systems with closed Kinematic Loops. *SIAM J. Scient. Computing*, 1999.

[ATL] ATLAS. *Automatically Tuned Linear Algebra Software*. math-atlas.sourceforge.net.

[Bau99] I. Bauer. *Numerische Verfahren zur Lösung von Anfangswertaufgaben und zur Generierung von ersten und zweiten Ableitungen mit Anwendungen bei Optimierungsaufgaben in Chemie und Verfahrenstechnik*. PhD thesis, University of Heidelberg, 1999. Download at: http://www.ub.uni-heidelberg.de/archiv/1513.

[Bau00] I. Bauer. *Numerische Verfahren zur Lösung von Anfangswertaufgaben und zur Generierung von ersten und zweiten Ableitungen mit Anwendungen bei Optimierungsaufgaben in Chemie und Verfahrenstechnik*. PhD thesis, Universität Heidelberg (IWR), 2000.

[BBS99] I. Bauer, H. G. Bock, and J. P. Schlöder. DAESOL – a BDF-code for the numerical solution of differential algebraic equations. Internal report, IWR, SFB 359, University of Heidelberg, 1999.

[BEJ88] H. G. Bock, E. Eich, and Schlöder J. Numerical solution of constrained least squares boundary value problems in differential–algebraic equations. In K. Strehmel, editor, *Numerical Treatment of Differential Equations*, BG Teubner. Leibzig, 1988.

[BKS00] H. G. Bock, E. Kostina, and J. P. Schlöder. On the role of natural level functions to achieve global convergence for damped Newton methods.

In M. J. D. Powell and S. Scholtes, editors, *Proc 19 IFIP TC 7 Conf. on System Modelling and Optimization*. Kluwer Academic Publishers, 2000.

[Blo99] J. Blommers. *Architecturing Enterprise Solutions with Unix Networking*. Prentice Hall PTR, New Jersey, 1999.

[Boc77] H. G. Bock. Zur numerischen Behandlung zustandsbeschränkter Steuerungsprobleme mit Mehrzielmethode und Homotopieverfahren. *Z. Angew. Math. Mech.*, 57, 1977.

[Boc81] H. G. Bock. Numerical treatment of inverse problems in chemical reaction kinetics. In K. H. Ebert, P. Deuflhard, and W. Jäger, editors, *Modelling of Chemical Reaction Systems*, volume 18 of *Springer Series in Chemical Physics*. Springer, Heidelberg, 1981.

[Boc83] H. G. Bock. Recent advances in parameter identification techniques for ODE. In P. Deuflhard and E. Hairer, editors, *Numerical Treatment of Inverse Problems in Differential and Integral Equations*. Birkhäuser, 1983.

[Boc85] H. G. Bock. *Randwertproblemmethoden zur Parameteridentifizierung in Systemen nichtlinearer Differentialgleichungen*. PhD thesis, Universität Bonn, 1985.

[Boc87] H. G. Bock. *Randwertproblemmethoden zur Parameteridentifizierung in Systemen nichtlinearer Differentialgleichungen*, volume 183 of *Bonner Mathematische Schriften*. University of Bonn, Bonn, 1987.

[Bor01] Borland. *Visibroker for C++ 4.5 (Programmers Guide)*, 2001.

[Box98] D. Box. *COM*. Addison–Wesley, Bonn, 1998.

[Che96] F. L. Chernousko. Construction of a Control when There Are Mixed Constraints. *Journal of Computer and Systems Sciences International*, 1996.

[Dat97] K. Dattatri. *C++ Effective Object–Oriented Software Construction*. Professional Computing. Prentice Hall, Upper Saddle River, 1997.

[Den85] Y. C. Deng. *Human Head–Neck–Upper–Torso Model Response to Dynamic Loading*. PhD thesis, University of California, Berkley, 1985.

[Deu74] P. Deuflhard. A modified newton method for the solution of ill–conditioned systems of nonlinear equations with application to multiple shooting. *Numer. Math.*, 22:289–315, 1974.

[DG87] Y. C. Deng and W. Goldsmith. Response of a human head–neck–upper–torso replica to dynamic loading – II analytical–numerical model. *J. of Biomechanics*, 20, 1987.

[Eic92] E. Eich. *Projizierende Mehrschrittverfahren zur numerischen Lösung von Bewegungsgleichungen technischer Mehrkörpersysteme mit Zwangsbedingungen und Unstetigkeiten*, volume 109 of *Fortschrittberichte VDI* (Reihe 18). VDI–Verlag GmbH, Düsseldorf, 1992.

[Erg86] Normierungsausschuss Ergonomie. *Körpermaße des Menschen.* DIN–Norm 33402, Deutsches Institut für Normierung e. V., Berlin, 1978 / 1984 / 1986.

[Fri00] M. Fritz. Simulating the response of a standing operator to vibration stress by means of a biomechanical model. *Journal of Biomechanics*, 2000.

[GB94] J. Garcìa de Jalòn and E. Bayo. *Kinematic and Dynamic Simulation of Multibody Systems*. Springer, Berlin, 1994.

[GDSR87] K. Gruber, J. Denoth, E. Stuessi, and H. Ruder. The wobbling mass model. *International Series on Biomechanics*, 6, 1987.

[GMW81] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press, 1981.

[Gri98] F. Griffel. *Componentware*. dpunkt.verlag, Heidelberg, 1998.

[Gue97] M. Guenther. *Computersimulationen zur Synthetisierung des muskulär erzeugten menschlichen Gehens unter Verwendung eines biomechanischen Mehrkörpermodells*. PhD thesis, Eberhard-Karls-Universität Tübingen, 1997.

[Hah93] U. Hahn. Entwicklung mehrgliedriger Modelle zur realistischen Simulation dynamischer Prozesse in biologischen Systemen. Master's thesis, Eberhard-Karls-Universität Tübingen, 1993.

[Han76] S. P. Han. Superlinearly convergent variable-metric algorithms for general nonlinear programming problems. *Math. Progr.*, 11:263–282, 1976.

[Hat81] H. Hatze. Myocybernetic control models of skeletal muscle – characteristics and applications. *University of South Africa Press, Pretoria*, 1981.

[Hil38] A. V. Hill. The heat of shortening and the dynamic constants of muscle. In *Proceedings of the Royal Scociety of London B 126*, pages 136–195, 1938.

[Jac01] N. Jacob. Preprocessing-Methoden für Mehrzielverfahren zur Lösung von Parameterschätzproblemen. Master's thesis, Universität Heidelberg, 2001.

[KE85] P. Krämer-Eis. *Ein Mehrzielverfahren zur numerischen Berechnung optimaler Feedback-Steuerungen bei beschränkten nichtlinearen Steuerungsproblemen*, volume 166 of *Bonner Mathematische Schriften*. Universität Bonn, 1985.

[KMB04] C. Kraus, H. Mutschler, and H. G. Bock. Parameter estimation for biomechanical models based on redundand coordinates. *Multibody System Dynamics*, accepted 2004.

[Kra97] C. Kraus. Modellierung und rekursive Algorithmen für Mehrkörpersysteme in Natürlichen Koordinaten. Master's thesis, Universität Heidelberg, 1997.

[KWB00] C. Kraus, M. Winckler, and H. G. Bock. Modeling Mechanical DAE Using Natural Coordinates. *Mathematical and Computer Modelling of Dynamical Systems*, 7,2:145–158, 2000.

[Lak96] J. Lakos. *Large–Scale C++ Software Design*. Addison–Wesley, Reading, 1996.

[LBBS02] D. B. Leineweber, I. Bauer, H. G. Bock, and J. P. Schlöder. An efficient multiple shooting based reduced SQP strategy for large-scale dynamic process optimization. part i: Theoretical aspects. 2002. (submitted).

[Lei95] D. B. Leineweber. Analyse und restrukturierung eines verfahrens zur direkten lösung von optimal-steuerungsproblemen (the theory of muscod in a nutshell). Master's thesis, IWR Heidelberg, 1995.

[Lin98] C. Linnhoff–Popien. *CORBA Kommunikation und Management*. Springer, Berlin, 1998.

[LL75] L. D. Landau and E. M. Lifshitz. *Lehrbuch der theoretischen Physik I*. Akademie Verlag, Berlin, 1975.

[Mom02] K. D. Mombaur. *Stability Optimization of Open–Loop Controlled Walking Robots*. Meß-, Steuerungs- und Regelungstechnik. VDI Verlag, Düsseldorf, 2002.

[MOR95] D. Ma, L. A. Obergefell, and A. L. Rizer. Development of human articulating joint model parameters for crash dynamics simulations. *Stapp Car Crash Conference*, 1995.

[Mut04] H. Mutschler. *Menschmodelle bei niedrigen Beschleunigungen*. PhD thesis, Eberhard-Karls-Universität Tübingen, to be published 2004.

[NAS78] NASA. *Anthropometric source book.* Reference Publication 1024 (I-III), NASA Scientific and Technical Information Office, Springfield, VA, 1978.

[NM65] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[Pow78] M. J. D. Powell. A fast algorithm for nonlinearly constrained optimization calculations. In G. A. Watson, editor, *Numerical Analysis, Dundee 1977*, volume 630 of *Lecture Notes in Mathematics*, Berlin, 1978. Springer.

[SBS98] V. H. Schulz, H. G. Bock, and M. C. Steinbach. Exploiting Invariants in the Numerical Solution of Multipoint Boundary Value Problems for DAEs. *SIAM J. Sci. Comp.*, 19:440–467, 1998.

[Sch86] W. Schiehlen. *Technische Dynamik.* B. G. Teubner, Stuttgart, 1986.

[Sch88] J. P. Schlöder. Numerische Methoden zur Behandlung hochdimensionaler Aufgaben der Parameteridentifizierung. *Bonner Mathematische Schriften*, 187, 1988.

[Sch99] R. v. Schwerin. *MultiBody System SIMulation.* Springer, 1999.

[Sha86] L. F. Shampine. Conservation Laws and the Numerical Solution of ODEs. *Computation and Mathematics with Applications, Part B*, 12, 1986.

[Sim98] J. Simon. Modellierung von Kontaktereignissen bei der Simulation von Laufvorgängen. Master's thesis, Universität Heidelberg, 1998.

[Soe92] van A. J. Soest. *Jumping from structure to control: a simulation study of explosive movements.* PhD thesis, Vrije Universiteit, Amsterdam, 1992.

[Ste98] U. Steinmüller. Konfektioniert: Software–komponenten in java: Beans. *iX*, 2, 1998.

[Sto04] T. Stoßmeister. *Optimales Laufen.* PhD thesis, Universität Heidelberg (IWR), to be published 2004.

[vW97] R. v. Schwerin and M. J. Winckler. A Guide to the Integrator Library MBSSIM. Technical report, Interdisciplinary Center for Scientific Computing (IWR), University of Heidelberg, 1997.

[Wil63] R. B. Wilson. *A simplicial algorithm for concave programming.* PhD thesis, Harvard University, 1963.

[Win01] M. J. Winckler. *Numerische Werkzeuge zur Simulation, Optimierung und Visualisierung unstetiger dynamischer Systeme.* Cuvillier Verlag, Göttingen, 2001.

[WK00] M. J. Winckler and C. Kraus. Simulation of Hexapod Machine Tools by Using Natural Coordinates. In *Proceedings of Year 2000 PKM International Conference*, 2000.

[Yan01] D. Yang. *C++ and Object–Oriented Numeric Computing*. Springer, Berlin, 2001.

[Zaj89] F. E. Zajac. Muscle and tendon: properties, models, scaling, and application to biomechanics and motor control. *CRC Critical Reviews in Biomechanical Engineering*, 17:359–411, 1989.