

Modelica-Based Computational Tools for Sensitivity Analysis via Automatic Differentiation

Von der Fakultät für
Mathematik, Informatik und Naturwissenschaften
der RWTH Aachen University
zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften
genehmigte Dissertation vorgelegt von

Atiyah Mohamed Gamal Elsheikh, M.Sc.
aus Reutlingen

Berichter: Univ.-Prof. Dr.rer.nat. Uwe Naumann
Univ.-Prof. Dr.rer.nat. Wolfgang Wiechert

Tag der mündlichen Prüfung: 31.01.2012

Diese Dissertation ist auf den Internetseiten der
Hochschulbibliothek online verfügbar

Modelica-Based Computational Tools for Sensitivity Analysis via Automatic Differentiation

Atiyah Mohamed Gamal Elsheikh, M.Sc.

September 14, 2011

To my Family

Statement of Authorship

I certify that this thesis is my original work and I properly referenced all sources. Except where reference is made in the text, this document contains no material extracted in whole or in part from a document previously published by me. This thesis was not previously submitted to another academic institution.

Aachen September 14, 2011

Atiyah Mohamed Gamal Elsheikh, M.Sc.

Acknowledgments

First of all, thank God that has given me the power, opportunities and conditions not only to conduct this work, but also to cross this rich journey. This work has been conducted, written and published within four places: Siegen, Jülich, Cairo and Vienna. This has given me the chance to meet and work with many people of different scientific backgrounds and various perspectives. Here I'd like to thank those who have positively influenced the resulting work. In particular,

- Prof. Wolfgang Wiechert for his appreciable contribution, for his special care to the research direction and for supervising me among others the clarification of many advanced topics, written and presentation skills and for many things from which I always benefit
- Prof. Uwe Naumann for discussions, his friendly tips, his contribution to the form of this work and for the influence of his book

Further I'd like to thank:

- Dr. Michael Weitzel for his valuable discussions in advanced programming and algorithms and for his unlimited willing for helping me whenever I managed to crash my Linux
- Dr. Stephan Noack for valuable discussions in Systems Biology and visualization techniques, using first unstable versions of ADModelica and his valuable feedback reports
- Tolga Dalman for his contribution as a turning point for realizing large-scale optimization of Modelica models

More than half of the publications out of this work has been published within my role in the Austrian Institute of Technology. Two colleagues made the combination of the publication-phase of the dissertation and my new role in AIT much easier for me. I'd like to thank

- Dr. Peter Palensky for truthful support and cooperation and for the extraordinary liberal scientific environment
- Dr. Edmund Widl as a truly helpful and cooperative person

I'd like also to thank:

- Dr. Adrian Pop from Linköping university who spent several hours in weekends and Christmas for porting the XMLModelica tool to Linux which was so useful for ADModelica
- Prof. Francesco Casella from Politecnico di Milano, Italy for his long inspiring responds to many questions to ODE and Modelica in OpenModelica forums
- Prof. Peter Verheijen from TU Delft for long enthusiastic discussions in modeling languages and his influential tips concerning logarithmic scaling techniques for parameter estimation
- Prof. Ekatharina Kostina from Marburg university for her discussions in the topics of multiple-shooting algorithms and optimal experimental design
- My colleague Jana Tillack for sharing her models and figures
- Xiao Ke for his useful scripts and the visualization tool conducted within his master thesis
- My colleagues from Siegen Frieder Hadlich, Remo Winz and Dr. Peter Droste

Finally, I'd like to thank my family for their support during my stay in Egypt at 2011 in both tough but interesting times. Their support enabled me to write this thesis and conduct many valuable highlights in this work

Abstract

This work is mainly concerned with sensitivity analysis of DAE-based models described by the modern object-oriented modeling language Modelica. In this context, an automatic differentiation tool named as ADModelica is presented. It fully employs Modelica-based compiler techniques forming a new automatic differentiation approach for non-causal equation-based languages. Already existing open-source compiler tools are utilized for reducing implementation efforts. A generated output model efficiently represents a sensitivity equation system by which parameter sensitivities can be simulated using any existing Modelica simulation environment. The resulting tool has been successfully applied on high-level Modelica models in the field of Systems Biology. In benchmark examples, the performance of the generated models are better than applying common finite difference methods in terms of accuracy and runtime performance. Moreover, the representation of these models permits the exploitation of structural characteristics of sensitivity equation systems for significantly improved runtime performance on supercomputer clusters.

Using ADModelica, several sensitivity analysis application studies of computationally, algorithmically and technically challenging nature have been performed towards the realization of stable efficient parameter estimation process of large and badly-scaled dynamical models. These studies cover among others:

- The examination of several global multistart optimization methods w.r.t. results quality and implementation efforts, in particular the design of new derivative-based hybrid heuristic strategies
- The determination of confidence regions of model parameters via identifiability analysis techniques based on linearized statistics and Monte Carlo bootstrap methods

Within this work further Modelica-based both domain-dependent and domain-independent computational tools have been implemented such as:

- A compact Modelica library for simplified kinetics for modeling complex reaction systems through which model families can be easily specified
- A tool for visualizing scaled parameter sensitivities within a supervised master thesis
- A Modelica-based editor for modeling biochemical reaction networks within a collaborative work with colleges

Finally, this thesis also covers theoretical studies concerning the differential and the structural index of a DAE system and the corresponding sensitivity equation system with an interesting mathematically proven conclusion about their relationship.

Modelica-basierte rechenbetonte Werkzeuge für Sensitivitätsanalyse durch Automatisches Differenzieren

Atiyah Mohamed Gamal Elsheikh, M.Sc.

September 14, 2011

Zusammenfassung

Diese Arbeit befasst sich hauptsächlich mit der Sensitivitätsanalyse DAE-basierter Modelle, die mit der modernen, objektorientierten Modellierungssprache Modelica beschrieben sind. In diesem Zusammenhang wurde ADModelica, ein Werkzeug fürs Automatische Differenzieren von Modelicamodellen, entwickelt. Anderes als gängige Ansätze verwendet dieses Werkzeug Modelica-basierte Compilerverfahren, wodurch ein neuer Ansatz zum Automatischen Differenzieren gleichungsbasierter Sprachen entstanden ist. Um ein derartiges AD-Werkzeug für die umfangreiche Sprache Modelica zu ermöglichen, wurde ein vorhandener quelloffener Compiler verwendet. Ein automatisch generiertes Modell enthält eine effiziente Formulierung der Sensitivitätsgleichung, womit Parametersensitivitäten mithilfe jeder beliebigen Simulationsumgebung für Modelica berechnet werden können. ADModelica wurde auf realistische Modelicamodelle aus dem Bereich der Systembiologie erfolgreich angewendet. Die Benchmarks zeigen, dass die Laufzeiteffizienz der Simulation generierter Modelle und die Genauigkeit der resultierenden Parametersensitivitäten erheblich besser sind als bei den üblichen Methoden der finiten Differenzen. Des Weiteren gestattet die Repräsentation dieser Modelle die Ausnutzung struktureller Eigenschaften der Sensitivitätsgleichung zur verbesserten Simulationslaufzeit auf Hochleistungsrechnern.

Mit ADModelica wurden einige zeitintensive, algorithmisch schwierige und technisch herausfordernde Studien von Anwendungen der Sensitivitätsanalyse durchgeführt. Mit diesen Anwendungen lassen sich Verfahren zur Parameterschätzung schlecht skalierter dynamischer Modelle, eine herausfordernde Problemstellung aus dem Bereich Systembiologie, stabil und effizient realisieren. Diese Studien beinhalten u. a.:

- Untersuchung einiger globaler Optimierungsstrategien im Bezug auf Qualitätsergebnisse und Implementierungsaufwand: In diesem Zusammenhang wurden neue ableitungsbasierte hybride Optimierungsstrategien entworfen.
- Bestimmung der Konfidenzintervalle von Modellparametern durch gängige Techniken aus dem Bereich Identifizierbarkeitsanalyse.

Außerdem wurden sowohl domainabhängige als auch domainunabhängige rechenbetonte Modelica-basierete Softwarewerkzeuge implementiert, z. B.:

- ein Werkzeug zur Visualisierung von skalierten Parametersensitivitäten im Rahmen einer betreuten Masterarbeit,
- eine Modelicabibliothek zur Modellierung biochemischer Netzwerke mit vereinfachten Kinetikgleichungen, durch die Modellfamilien spezifiziert werden können, und
- ein Modelica-basierter Editor zur Modellierung biochemischer Reaktionsnetzwerke im Rahmen einer Zusammenarbeit mit Kollegen.

Des Weiteren enthält diese Dissertation theoretische Studien bezüglich des Differentiations- und Strukturindex von differential-algebraischen Gleichungssystemen mit einem interessanten Beweis zur ihrem Zusammenhang.

List of Symbols and Abbreviations

List of Symbols

Mathematical Symbols and Notation

λ eigenvalue

Algorithmic Complexity / Growth of Functions (Landau Symbols)

$\mathcal{O}(g(n))$ asymptotic upper bound: $\{f(n) \mid \exists c \ 0 \leq f(n) \leq cg(n) \ \forall n \geq n_0\}$

$\Theta(g(n))$ asymptotic tight bound: $\{f(n) \mid \exists c_1, c_2 \ 0 \leq c_1g(n) \leq f(n) \leq c_2g(n) \ \forall n \geq n_0\}$

$\Omega(g(n))$ asymptotic lower bound: $\{f(n) \mid \exists c \ 0 \leq cg(n) \leq f(n) \ \forall n \geq n_0\}$

List of Abbreviations

Mathematics and Computer Science

ABM Adams-Bashforth-Moulton method

AD Automatic Differentiation

AST Abstract Syntax Tree

BDF Backward Difference Formula

BFS Breadth First Search

BLT Block Lower Triangle form

CA Computer Algebra

CD Central Difference

CORBA Common Object Request Broker Architecture

DAE Differential Algebraic Equation

FD Finite Difference

FIM Fisher Information Matrix

FLOPs FLoating-point OPerations

FLOPS FLoating-point OPerations per Second

NLP Non Linear Programming problems

ODE Ordinary Differential Equation

PDE Partial Differential Equation

OMC OpenModelica Compiler

PDAE Partial Differential Algebraic Equation

SCC Strongly Connected Component

List of Symbols and Abbreviations

SD Symbolic Differentiation

Systems Biology

CCCs Concentration Control Coefficients
DNA Deoxyribonucleic Acid
EFM Elementary Flux Mode
FCCs Flux Control Coefficients
MCA Metabolic Control Analysis
MFA Metabolic Flux Analysis
MS Mass Spectrometry
mRNA messenger Ribonucleic Acid
NMR Nuclear Magnetic Resonance
SMFA Stoichiometric Metabolic Flux Analysis
SRE Stimulus Response Experiment
TCA Tri-Carboxylic Acid

Software and languages

ACSL Advanced Simulation Continuous Language
API Application Programming Interface
Dymola Dynamic Modeling Language
GUI Graphical User Interface
MSL Modelica Standard Library
SBML Systems Biology Markup Language
UML Unified Modeling Language
XML EXtended Markup Language

Contents

1. Introduction	1
1.1. Motivation	1
1.2. Main topics of this work	2
1.3. Structure of the thesis	3
I. Fundamental Concepts of Modelica	5
2. Introduction to Modelica	7
2.1. Background and motivation to Modelica	7
2.2. Universal physical concepts behind Modelica	9
2.3. Equation-based object oriented modeling with Modelica	11
3. Compiler Methods for Modelica	13
3.1. Code flattening and optimization	13
3.2. Graph theory tools for DAE systems	14
3.3. System decomposition in BLT-Format	16
3.4. Tearing algorithms	17
4. Mathematical Aspects behind Modelica	19
4.1. A mathematical representation of Modelica models	19
4.2. Handling general DAE systems	20
4.3. Index reduction using the structural index	21
4.4. Numerical methods for ODE/DAE systems	23
II. Fundamentals in Systems Biology	25
5. Basic Concepts of Systems Biology	27
5.1. Basic Concepts	27
5.1.1. Systems Biology	27
5.1.2. Metabolic Engineering	28
5.2. Enzyme kinetics	29
5.2.1. Mechanistic kinetics	30
5.2.2. Kinetic formats	32

Contents

6. Modeling Examples	35
6.1. Modeling biochemical reaction networks	35
6.2. Modeling labeled biochemical reaction networks	37
6.3. Vertical modeling	39
6.4. Summary of models	40
 III. Modeling Applications in Systems Biology with Modelica	 41
7. Modelica as a Descriptive Language for Systems Biology	43
7.1. SBML	44
7.2. Modeling biochemical networks with the Biochem library	45
7.2.1. What is the Biochem library?	45
7.2.2. Overview of the Biochem library	46
7.3. Advantages of Modelica versus SBML	47
 8. The ADGenKinetics library	 49
8.1. Overview	49
8.2. The structure of ADGenKinetics	50
8.3. Implementation of reactions	51
8.4. Examples	52
 9. The Omix-Modelica plugin	 55
9.1. Main difficulties of converting an Omix network to Modelica	55
9.2. Design of the Omix-Modelica plugin	56
9.3. The problem of enormous numbers of kinetics	57
9.4. Enriching the Biochem library with annotations	58
9.5. The contribution made to the Omix-Modelica plugin through this work	59
 10. Implementation of Model Families in Modelica	 61
10.1. The concept of model family	61
10.2. Specification of model families in Modelica	63
10.2.1. Kinetic law variants	64
10.2.2. Enzymatic variants	65
10.3. The advantages of the Modelica approach	65
10.3.1. Common issues with the Omix-Modelica plugin	65
10.3.2. Model family: Modelica versus SBML	66
 IV. Sensitivity Analysis of Modelica Models	 67
11. Parameter Sensitivities of DAE Systems	69
11.1. Common methods	69
11.2. Introduction to AD	70

11.3. Algorithmic aspects of AD	73
12. Overview of ADModelica	75
12.1. Why AD for Modelica?	75
12.2. ADModelica: An AD tool for Modelica models	76
12.3. Description of the generated code	77
13. Algorithmic Concepts behind ADModelica	79
13.1. AD of equations	79
13.1.1. Equations vs. assignments	79
13.1.2. Non-causal equation-based AD	79
13.1.3. Eliminating derivative objects of intermediate variables	81
13.1.4. Efficiency	82
13.2. Modelica-based compiler optimization	82
13.2.1. Removal of trivial equations	83
13.2.2. Code optimization with the Modelica compiler	83
13.2.3. High-level Modelica-based compiler techniques	83
14. The Index of Sensitivity Equation Systems	85
14.1. Motivation	85
14.2. DAE systems of differential index one	86
14.3. DAE systems of arbitrary differential index	87
14.3.1. Common structural characteristics	87
14.3.2. The structural index of sensitivity equation systems	89
15. Accuracy of Finite Difference Methods	91
15.1. Finite difference methods	91
15.2. Accuracy benchmark	92
16. Efficient Simulation of Sensitivity Equation Systems	95
16.1. Runtime benchmark	95
16.2. Runtime and space complexity	97
16.3. Runtime performance improvement	99
V. Applications of Sensitivity Analysis	101
17. Large-scale Parameter Estimation	103
17.1. Common problems behind large-scale parameter estimation	103
17.2. Identification of optimal start values	105
17.3. Hybrid heuristics	106
17.3.1. Relay mode: Scatter search	107
17.3.2. Co-evolutionary hybridization	108
17.3.3. Benchmark	108

Contents

18. Statistical Analysis of Parameter Estimation	111
19. Visualization of Parameter Sensitivities	117
19.1. Motivation	117
19.2. Control coefficients	118
19.3. Visualization of control coefficients	120
20. Summary, Outlook and Future Perspectives	123
20.1. Summary	123
20.2. Outlook	125
 VI. Technical Notes	 129
A. Semantical Advantages of Modelica against SBML	131
B. Specialized ODE/DAE Solvers for Parameter Sensitivities	135
C. Possible Approaches for Differentiating Modelica Models	139
C.1. On which level should AD be applied?	139
C.2. Automatic differentiation of Modelica libraries	141
D. A Survey of Optimization Algorithms	143
D.1. Technical difficulties of large-scale parameter estimation	143
D.2. Metaheuristics	144
D.3. Gradient-based optimization algorithms	145
D.4. Some global optimization algorithms	147
E. Implemented Software	151
E.1. C++ software for communicating with Dymola	151
E.2. Interactive dynamic menus	152
 Bibliography	 155

Chapter 1.

Introduction

1.1. Motivation

Model-based scientific research aiming at gaining valuable insights into complex systems usually requires an in-depth knowledge in a wide spectrum of scientific majors. From one side, this enforces multidisciplinary activities to take place in a collaborative manner. From the other side, experiences from highly-specialized areas become a requirement for a successful continuous progress. These aspects are much obvious in the fields of Systems Biology and Metabolic Engineering (cf. section 5.1). Knowledge in these domains is usually gained from data resulting from various experiment types (Noack 2009). Due to the diversity and the richness of these data sources, comprehensive information would not be directly deduced. In this context, mathematical modeling becomes very vital for providing a quantitative description of the underlying systems. The resulting models are the basis of mathematical tools for dealing with sophisticated questions. These questions are answered by solving mathematical problems that may correspond to an elementary simulation of an ODE system or an optimization problem but also to a computationally challenging inverse problem.

The ability of constructing models in a flexible and intuitive way forms a strong basis for supporting collaborative research. While specialists can directly focus on the physical construction of highly-complex models without paying attention at technical and pure mathematical details, computational scientists can focus on solving complex mathematical problems without getting deeply involved in the scientific background of the underlying models. These desired features are one of the essential advantages of modern modeling and simulation languages. Such languages based on standardized design concepts have been continuously developed along decades ago, one of the most modern of which is Modelica. Modelica is a universal equation-based modeling language supporting object-oriented design and code-reuse from a continuously growing large set of libraries. It imposes a lot of attractive features with which physical modeling becomes a straightforward task. Construction of hierarchically organized complex models becomes the task of dragging, dropping, browsing and connecting icons together.

This work follows a newly established trend aiming at building a bridge between the Systems Biology community and the physical modeling community through the language

Modelica. The attractive features of Modelica make us believe that this trend is always going to get better evolved and more practiced in the near future (Wiechert et al. 2010, Elsheikh 2012). The main contribution of this work is mainly concerned with sensitivity analysis of Modelica-based models in the field of Systems Biology. The provided tools both at modeling and simulation levels efficiently contribute to the solution of computationally-expensive tasks related to hard ill-posed inverse problems. Meanwhile, being a universal modeling language, some of the implemented tools and concepts can be generally applied to all types of Modelica models as the title of this thesis directly suggests. A hidden aspect of the title is that the resulting works are influenced by many concepts on which Modelica is based as it will be illustrated in many contexts.

This work has been partially performed within the Evonik industries and BMBF co-funded project SysMAP¹. This project taken by several academical working groups has the ambitious goal of studying, analyzing and understanding various cellular activities within the organism *Corynebacterium Glutamicum* seeking an integrative understanding of its metabolic regulation for improving the fermentation process (cf. chapter 5). This thesis summarizes some of the activities taken within the modeling part of the SysMAP project.

1.2. Main topics of this work

Modeling and simulation of physical systems is, in general, a complex iterative process. Asserted models are necessarily based on simplifications and in many cases are subject to improvement and optimization. In this context, a wide range of applications of sensitivity analysis can assist the modeling process, from parameter fitting and optimization through model validation to statistical analysis and experimental design. For DAE-based models, computation of parameter sensitivities represents the potentially most expensive part in such tasks. These common methods, among others, drew increasing attention to a research area of scientific computing, i.e. Automatic Differentiation or Algorithmic Differentiation (AD) of program code. The main task within this work is aiming at providing efficient solutions for sensitivity analysis of DAE-based Modelica models in terms of performance and accuracy using AD techniques. The following key aspects need to be realized:

1. Implementation of a reliable AD tool for performing sensitivity analysis of large-scale Modelica models with little efforts by utilizing Modelica concepts, available open-source compilers and common advanced DAE solvers
2. Providing a reliable framework for performing parameter estimation of highly non-linear large-scale Modelica models using advanced global optimization algorithms and statistical analysis techniques for examining the quality of the fits and the identifiability of model parameters

¹ Systems Biology on microbial Amino acid producers (project no. 0313704)

The fulfillment of these tasks are largely influenced by:

1. The underlying domain of models (i.e. Systems Biology)
2. The nature of the universal modeling language Modelica

Models from the field of Systems Biology, describing complex cellular processes, are necessarily based on idealized assumptions that could be far away from reality. Many tiny details get neglected to avoid overparameterization of models. Ultimately, it is not known how far these assumptions influence the quality of the descriptive models in terms of correctness, validity and predictive power. In comparison with some other physical domains where the underlying "human-made" technical systems are fully understood up to their tiniest parts, model assumptions could be experimentally examined and justified in better ways. Moreover, constructed models are based on measurements expressing cellular processes at various levels in the cell (so called Omics-data). From one side, these data measurements are erroneous. From the other side, the underlying inverse problem for model identification is so sensitive to measurements (i.e. ill-posed).

The employment of Modelica as a young domain-independent language had serious consequences on the underlying tasks needed for achieving the desired goals. Modelica, comprehensively presented in part one, still suffer from the lack of developer-oriented supporting tools in some serious aspects concerning standardized low-level communication with Modelica-based simulators. This is rather a general problem to the whole community to which standardized solutions are being suggested and continuously developed (Blochwitz et al. 2011). At the modeling level, though there are appreciable efforts for providing first guidelines for implementing libraries supporting modeling applications in the field of Systems Biology (Nilsson and Fritzson 2005), there are still no standardized published libraries in this field. Many desired features required for a practical realization of vital modeling applications are not supported.

Under the light of the previous discussion, further goals need to be achieved. At the modeling level, the absence of domain-dependent tools with which models can be easily constructed is a further motivation for implementing and enhancing existing Modelica libraries. With these libraries, applications requiring automatic model generation with highly specialized editors can be supported. At the technical level, Modelica-based infra structure software for supporting low-level communication via standard programming languages needs to be implemented. Based on these tools, implementing global optimization algorithms, supporting parameter estimation as an incremental process and utilizing supercomputers for achieving high performance are realized.

1.3. Structure of the thesis

This thesis consists of six parts. In part I, a comprehensive introduction to the language Modelica is given. In chapter 2, an overview of the basic features of Modelica is presented.

Chapter 1. Introduction

Chapter 3 summarizes the basic compiler steps performed to transform a high-level Modelica model into executable C code. Finally, chapter 4 introduces basic mathematical aspects concerning the simulation of DAE-based Modelica models. Part II is concerned with the fundamentals of the field of Systems Biology, the main applications domain of this work. Basic concepts of this field is introduced in chapter 5. Chapter 6 summarizes common modeling examples related to biochemical reaction networks.

Part III is concerned with Modelica-based tools and applications, performed within this work, in the field of Systems Biology. Chapter 7 discusses the advantages of Modelica in the field of Systems Biology. In this context, Modelica is compared with SBML, the standard specification language used by the Systems Biology community. Further comparative aspects are presented in the following chapters. Chapter 8 demonstrates the open-source free Modelica library for modeling biochemical reaction networks using generalized kinetic formats adequate for applications demanding automatic model generations. Chapter 9 presents a highly-specialized Modelica-based editor for biochemical networks. Finally, chapter 10 describes a reliable implementation of model families in Modelica, an important concept in the model validation process.

Part IV is concerned with algorithmic differentiation of Modelica models. Chapter 11 presents common ways for computing parameter sensitivities of DAE systems. In this context, basic notions of AD are introduced. Chapter 12 introduces the tool ADModelica, its design and an overview of generated code. In chapter 13, algorithmic concepts behind ADModelica adopting a new approach relevant for equation-based languages are discussed. Additionally, the main Modelica compiler optimization steps performed on ADModelica generated code are clarified. Chapter 14 presents a theoretical result of this work concerning the differential index of ADModelica code (i.e. the differentiated DAE systems). In chapter 15, the accuracy of finite difference methods applied to DAE systems is compared with analytical methods. Chapter 16 presents a runtime benchmark of ADModelica generated code.

Part V is concerned with computationally-expensive applications of sensitivity analysis in the field of Systems Biology. Chapter 17 discusses large-scale parameter estimation using new derivative-based hybrid heuristics designed within this work. Chapter 18 presents statical methods for examining the quality of parameter estimation results and determining the identifiability of model parameters. In chapter 19, a tool for visualizing parameter sensitivities is presented. Finally, chapter 20 summarizes the achieved results within this work and presents an outlook to further possible extensions to this work.

Finally part VI is concerned with technical notes related to all previous parts. The covered topics include among others: implementation details of some self-implemented Modelica libraries emphasizing highly-specialized aspects as well as algorithmic details behind some advanced DAE solvers, statistical methods and optimization algorithms.

Part I.

Fundamental Concepts of Modelica

Chapter 2.

Introduction to Modelica

This section gives a brief introduction to basic concepts necessary and sufficient to understand the problems demonstrated in this thesis. This includes some background to Modelica which gives a quick overview to the circumstances under which Modelica was invented and hence illustrates the basic motivation and goals behind this language. Additionally, clarifying the main modeling concepts adopted is necessary for appreciating the way resulting models are handled and solved. Finally, quick and basic elementary language constructs are illustrated through standard examples. For more details, comprehensive introduction can be found in (Fritzson 2003, Tiller 2001).

2.1. Background and motivation to Modelica

Since the invention of computers, modeling and simulation has been rapidly evolving both from methodological and technical perspectives. Early programmers were able to use ready "off-the-shelf" integrators and concentrate on the formulation of ODEs instead of explicitly coding an integrator themselves. The trend of simplifying the modeling and simulation process and making the user concentrate less on solution methods has continued ever since (Astroem et al. 1998). Initially, specialized modeling tools restricted to special domains enhanced by elegant GUI and numerical solvers optimal for particular models were implemented (eg. Spice for electrical circuits (Nagel and Pederson 1973)). Meanwhile, general-purpose tools have also emerged adopting the so-called block-diagram approach where a block of unknown quantities is computed by known quantities represented by another block as in Simulink. From one side, the user of such tools is not restricted to a specific domain, however he may require more time for engineering the formulation of his specific problem to more general concepts.

In 1978 Hilding Elmqvist pioneered the so-called non-causal modeling approach adopted by the Dymola modeling language implemented within his PhD thesis (Elmqvist 1978). This approach reduces the gap between specialized and general-purpose domain tools in a couple of aspects. An important aspect is that it relies on universal physical concepts (cf. section 2.2) with which many applications from different domains can be easily modeled and even integrated for multidisciplinary applications. The modeler is able to focus on modeling from a physical perspective by building up complex systems using elementary library components and connections similar to the real conceptual topology of the mod-

Chapter 2. Introduction to Modelica

eled system as any domain-specific tools. Additionally, graph and symbolic algorithms were introduced to transform the resulting model into a solvable form for appropriate ODE solvers. An important milestone in the development of this approach came in 1988 with the development of the Pantelides algorithm for DAE index reduction (Pantelides 1988). This allowed a large set of DAE-based models to be considered without bothering the end-user with the task of transforming his DAE model into solvable ODEs (cf. section 4.2).

Afterwards, many simulation tools from industry and academic sides relying on the non-causal modeling approach have been developed. While each of these languages may exhibit its own important aspects and distinguished design features, it was not possible to exchange models among these different groups. Additionally, a lot of excessive individual and conventional efforts have been done to implement these tools. Consequently, in 1996 again he was Hilding Elmqvist who initiated an effort to unify the splintered landscape of existing modeling languages. Many meetings by a wide spectrum of involved participants have been organized to discuss and design a modeling language that:

1. remains domain neutral as much as possible.
2. adopts many other modeling paradigms like the block-diagram approach for continuous-time based system, finite state automata and petri nets for discrete systems.
3. adopts relevant features provided by so far existing modeling tools.
4. provides a model-exchange specification language that can describe the small pieces of complex systems and their interrelationship among each others.
5. makes equations (and not assignments) be the base elements for describing the behavior of system components while also supporting the usual classical constructs.

As an ultimate result of all these efforts, the open-source non-proprietary specification of Modelica was developed and is continuously subject to further constructive discussions on an annual basis (Elmqvist and Mattsson 1997). For these reasons, Modelica can be considered as the convergence of many elegant ideas, well-established concepts and methodologies and as the successor of many successful modeling tools. Additionally, a free Modelica Standard Library (MSL) has been developed so that users of Modelica would not have to create their own basic models for many common modeling domains. Many compilers and simulation environments both commercial (eg. Dymola Elmqvist (1993), MathModelica Fritzson et al. (2002b)) and academic (OpenModelica Compiler (OMC) Fritzson et al. (2002a) and Mosilab Nytsch-Geusen and et al. (2005)) have been implemented for the task of performing equations generation, symbolic transformation and simulation. Even Matlab and Maple started to support Modelica specification using their powerful symbolic engine capabilities. OMC has been extensively exploited for assisting the implementation of many tools throughout this work, whereas Dymola is used for performing the simulations.

2.2. Universal physical concepts behind Modelica

Figure 2.1 shows two Modelica models from two different domains given in graphical notation. Regardless of the syntax (section 2.3) and semantics (chapter 7) of these two models, this section is mainly concerned with the basic modeling principles behind Modelica that enable descriptive modeling of physical systems from different domains. The resulting models are usually analogous to the topology of the underlying systems in reality. In

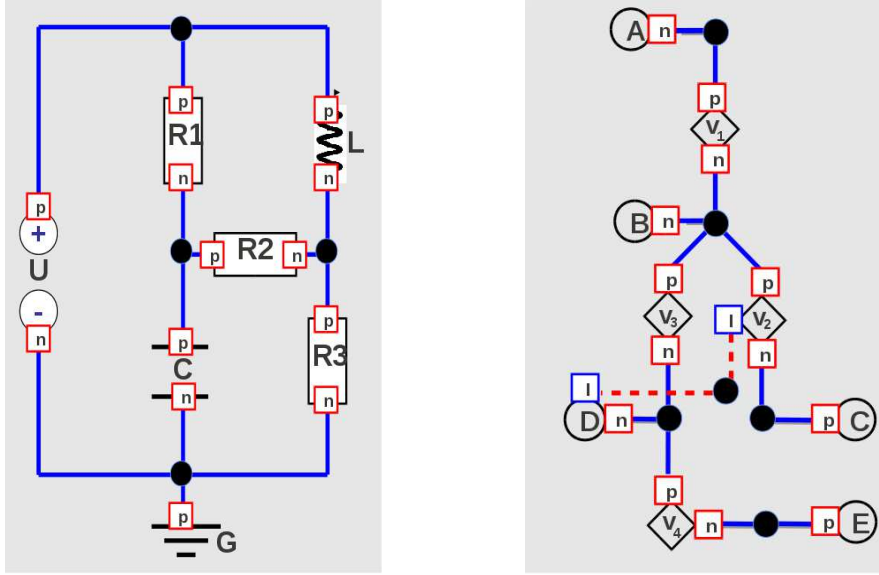


Figure 2.1.: Graphical Modelica Models composed of hierarchically organized components linked by using connectors through connections

comparison with the earlier block diagram approach, it is much easier to design and manipulate such presented models based on non-causal modeling (Mattsson and Elmqvist 1997). For example, while a slight modification to a complex system based on the early block-diagram approach (eg. insertion of an additional resistor) can be difficult to carry out, significant changes in a model following the non-causal design of a complex system is very straight forward.

The non-causal approach attempts to view a physical system, no matter how complex it is, as a finite set of independent components (eg. resistors, capacitors, etc.). Each component is considered as an independent particle that influences and gets influenced by the external world through universal conservation laws found in physical domains. Figure 2.2 illustrates how conservation laws are modeled. Each component (eg. resistor or a substance) is equipped with a port (or ports) with which communication to the external world is done by plugging it into other identical ports of other components.

Chapter 2. Introduction to Modelica

These ports are called connectors in the Modelica language. Within these connectors two kinds of quantities are characterized:

1. *flow variables*: carriers for transporting energy, charges, material etc.
2. *potential variables*: as carriers for representing the potential (i.e. level of energy) of a system

By plugging identical ports into each other, the interactions among independent components become defined by connection equations which assemble the underlying conservation laws of the modeled system. For example, the principle of conservation of energy states

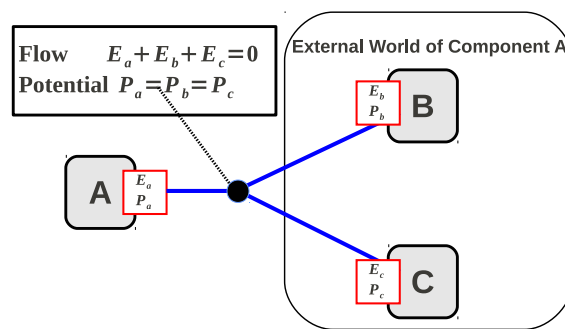


Figure 2.2.: Modeling flow of energy from external world to a component and vice versa

that the amount of energy in a closed system can neither be created nor destroyed. Additionally, the potential of two connected points must be equal. Therefore, plugged ports describes two kind of equations:

1. sum-to-zero equation for flow variables
2. equality equations for potential variables

As a consequence, the sum of all energy flows into and out of a component must be equal to zero. In (Fritzson 2003), it is shown how to choose the energy carrier and energy

Table 2.1.: The choice of energy carriers and energy level variables for non-causal modeling of electrical circuits and biochemical networks

Criteria - Domain	Electrical Engineering	Biochemical Engineering
Potential variables	Voltage	Substance concentration
Flow variables	Electrical charges	Reaction Rate
Conservation Law	Kirchhoff's law	Material flow balance

level variables for several physical domains. Table 2.1 demonstrates this for the running examples. Important consequences of non-causal modeling are that

- the notion of causality among variables (i.e. input and output variables) is completely absent. This clarifies why the topology of a physical system is preserved
- multidisciplinary modeling (Elsheikh et al. 2012) is feasible by simply identifying the energy carrier and energy level variables that assemble the flow of energy into and from a subsystem

2.3. Equation-based object oriented modeling with Modelica

This section gives a quick overview of basic Modelica constructs with which it is quite possible to build up reasonable physical systems. The presented simplified code is extracted from the Modelica Standard Library (MSL) for building electrical circuits. Such abstract examples are optimal for:

1. introducing Modelica constructs which are self-explainable and straightforward to extend to other domains
2. demonstrating Modelica compiler techniques with which the resulting equation systems are generated (cf. chapter 3).

In following chapters, no repetition of Modelica constructs are done unless new ideas or concepts require further new constructs. Chapter 7 is devoted for illustrating the advantages of Modelica in Systems Biology where additional constructs are presented.

Figure 2.3 shows the implementation of basic components with which electrical circuits is built. The most elementary unit is the connector class for representing the communication ports. Connectors identify two types of variables, potential variables (the voltage) and flow variables (the current). The abstract class *ElectricalPin* declares two connectors: *p* as a positive pole and *n* as a negative pole. The positive and negative notions represent dummy standards for directions (upper and left poles are positive, lower and right poles are negative) but they don't influence the mathematical representation in any aspect. By extending the abstract class, the components resistor, capacitor, inductor, etc. are implemented using equation-based syntax. Note that these equations don't specify any input-output relation, in contrast to assignments in classical procedural languages. These equations can be formulated in implicit form because Modelica specification does not enforce the modeler to write equations in explicit format. Once the implementation of all components are available, modeling becomes the matter of dragging and dropping icons, and connecting ports to each other using common graphical editors. The corresponding intuitive syntax of the graphical representation of the models in figure 2.1 is presented in figure 2.4. Following a similar approach, the biochemical network can be also implemented (Wiechert et al. 2010, Elsheikh 2012).

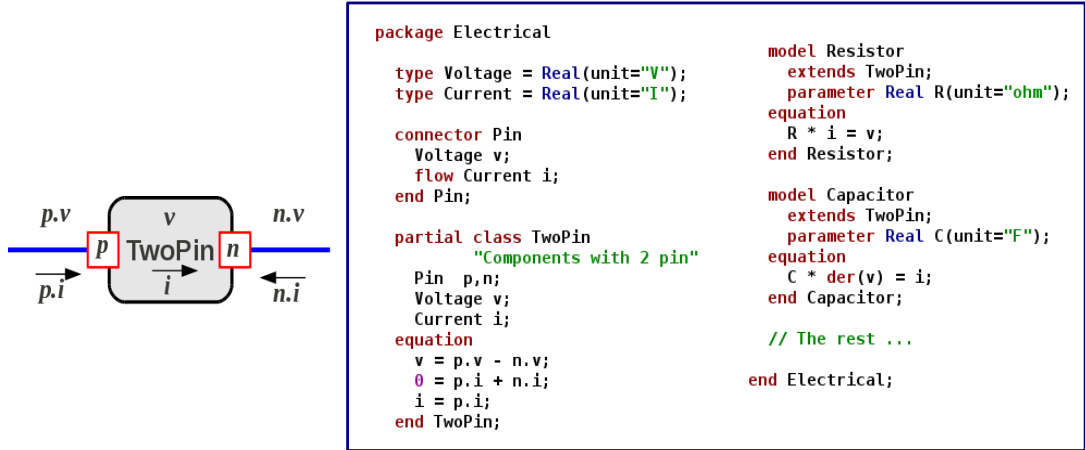


Figure 2.3.: Implementation of basic components of electrical circuits

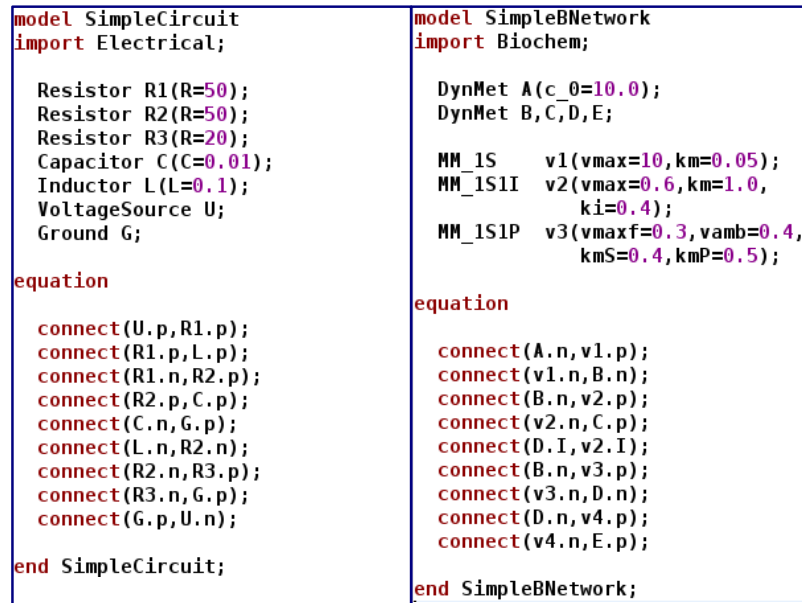


Figure 2.4.: Implementation of an electrical circuit and a biochemical network

Chapter 3.

Compiler Methods for Modelica

As already illustrated in the last chapter, the object-oriented approach of Modelica as well as the physical concepts behind it makes it quite easy for a modeler to build up large models consisting of thousands of equations. Consequently such large models demand special techniques both at compilation and simulation levels in order to perform efficient real-time simulations. This chapter demonstrates basic compiler steps taken to transform a Modelica model probably given in graphical format to simulation code through running examples, cf. figure 3.1. Only techniques needed throughout this work are emphasized. More details can be found in comprehensive literature like (Cellier 1991). The problem of index reduction is introduced in section 4.2, and the symbolic algorithm for solving this problem is presented in the context of theoretical results presented in chapter 14. Numerical methods for solving the resulting equation systems are presented in chapter 4.

3.1. Code flattening and optimization

Given a Modelica model with high-level constructs in graphical notations as in figure 2.1, the whole configuration is assembled into pure mathematical representation by the following rules:

1. For each component instance, one copy of all equations is generated with distinguished identifiers for local and port variables.
2. For each connection between two or more instances, potential variables are set to be equal and flow variables are summed to zero.

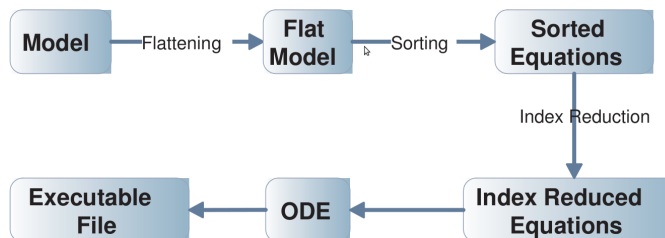


Figure 3.1.: Modelica Compilation Process

For example figure 3.2 shows a part of the assembled equations system describing the current and voltage at each point for the electrical circuit example. Equations system for the biochemical network example can be also generated in a similar manner, see (Wiechert et al. 2010).

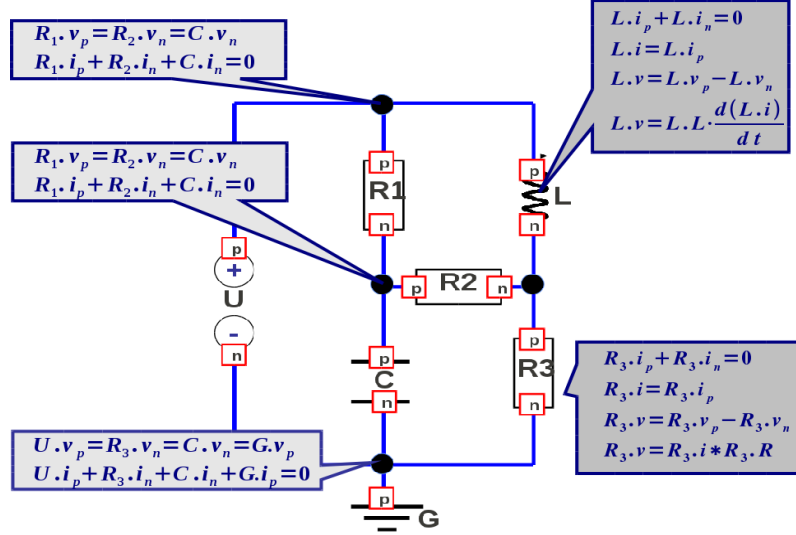


Figure 3.2.: The mathematical representation of connection points. Together with the equation systems of all components, the resulting equation system describes the current and voltage at each point within electrical circuit

The dimension of such automatically generated DAE systems is usually large due to the presence of `connect` statements corresponding to many equations of the form $u = v$ and $u + v = 0$. Such equations need not to be passed to the solver and they can get eliminated from the flattened DAE system by keeping one instance variable for each group of alias variables (Maffezzoni et al. 1996). A Modelica compiler employs simple computer algebra methods for drastically reducing such equation systems into minimal set of equations as shown in figure 3.3.

3.2. Graph theory tools for DAE systems

In order to symbolically handle and simulate automatically generated DAE systems, intermediate representations in form of directed bipartite graphs are used.

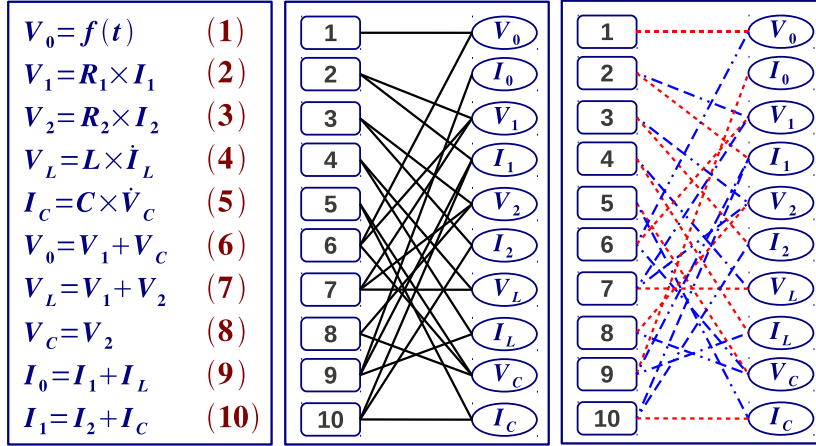


Figure 3.3.: Left: The simplified equation system of the electrical circuit. Dot notations are simplified with simpler identifiers
 Middle: The bipartite graph representation of the equation system
 Right: The structure digraph: A red edge means equation solves variable, blue edge means variable is used for solving the equation

Definition 3.1 (Bipartite graph representation of a DAE system). *A Bipartite Graph $G = (V, E)$ representation of a DAE system of the form ((4.2)) consists of a set of vertices:*

$$V = V_e \cup V_x =$$

$$\{e_i : \text{equation number } i\} \cup \{x_i : x_i \text{ is a variable}\} \quad i = 1, 2, \dots, N$$

and a set of edges:

$$E = \{(e_i, x_j) : x_j \text{ arises in equation } e_i\} \quad i, j \in \{1, 2, \dots, N\}$$

Remark 3.2. 1. $|V_e| = |V_x| = n$, otherwise the equation system is not consistent

2. G is clearly a bipartite Graph, i.e. \exists no edge (u, v) s.t. $\{u, v\} \subset V_x$ or $\{u, v\} \subset V_e$

The bipartite graph can be transformed by recursive applications of some simple rules into a directed graph called *Structure Digraph* for deducing the causality among variables and equations (i.e. which variable or group of variables is solved by which equation or group of equations). These rules are stated as follows:

1. equations with one variable would certainly solve this variable
2. any variable arising only once, is solved by the equation at which it arises

By applying these rules recursively, there are three possible causality relations for each edge $(e, x) \in E$,

1. e explicitly solves x .
2. x is used for solving e , i.e. x is used to solve another variable in e .
3. no explicit causality relation. In this case e becomes bidirectional.

Figure 3.3 shows the bipartite graph and the structure digraph representations of the electrical circuit from figure 2.1. In ideal cases, each equation explicitly solves a certain variable. In practice, groups of variables and equations usually need to be fulfilled concurrently.

3.3. System decomposition in BLT-Format

The resulting intermediate format from the last subsection inherits causality information that enable further improvement in simulation run-time. Instead of solving these equations as one single block, it is possible to decompose such equation system into smaller blocks of equations, which can be sequentially solved in a faster way due to the reduction of the overall complexity. This is done by applying Tarjan's algorithm (Tarjan 1972) to the structure digraph to obtain a set of Strongly Connected Components (SCCs). Figure 3.4 shows the resulting SCCs by applying Tarjan's algorithm on the structure digraph in figure 3.3. These SCCs establish a topological sorting of the equations from the SCCs, by which the equation system is transformed into the so-called BLT format¹. Having transformed the equation system into blocks of equations, several methods could be used for solving the resulting systems depending upon the resulting subsystems. In lucky cases analytical methods might be applied for solving simple equations (eg. explicit equations, linear ODE systems or analytically solvable ODE systems). In practice numerical methods for solving nonlinear-system of equations or DAE systems are applied.

In the electrical circuit example, the equations are sequentially solved according to the established topological sorting shown in figure 3.4. In the biochemical network example, BLT form shows a cascaded systems of DAE. Standard Modelica simulation environments would not attempt to exploit such format for fast sequential solving procedure using specialized solvers. In (Nöh and Wiechert 2004) an attempt for solving cascaded systems for special type of models is done. In general exploiting BLT format is common for algebraic equations but not yet exploited at DAE level. However, it is shown in section 13.2 that the underlying BLT format of DAE systems explains the behavior of the corresponding parameter sensitivities (Elsheikh and Wiechert 2008).

¹because the adjacency matrix of the bipartite graph is in block triangular form

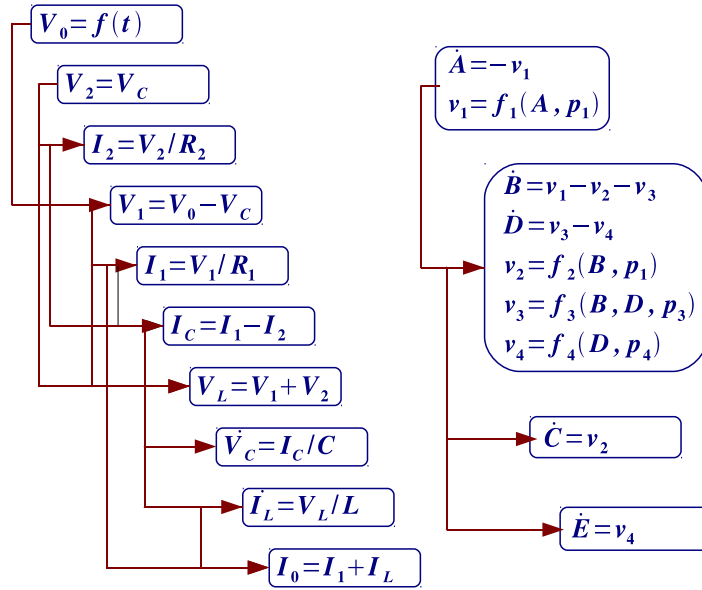


Figure 3.4.: The sorted equation according to the topological sorting out established from of the SCCs of structure digraph of the electrical circuit and biochemical network models

3.4. Tearing algorithms

In practice, it is not often the case that a system can be decomposed as in figure 3.4. By exchanging the capacitor with a resistor in the example of figure 2.1, the decomposition of the resulting equation system looks as shown in figure 3.5 (Cellier 1991). The large equation subsystem represented by the large SCC is referred to as *algebraic loop*. Variables in such algebraic loops don't exhibit any causality relationship and they need to be solved concurrently as a single block of equations. Such linear blocks of equations in figure 3.5 can be solved using Gaussian elimination. In large physical systems where large sets of equation systems result, efficiency is further improved by considering a sophisticated technique called tearing method (Kron 1963). Tearing algorithms are used for breaking algebraic loops into smaller equation systems. This is done by heuristically assigning an initial solution guess for a variable (*tear variable*).

In the running example from figure 3.5, assigning an initial guess for I_3 makes the entire set of equations directly solvable. After solving for other variables, a better guess is achieved for I_3 and the iteration can be done again and again until I_3 remains constant. This iteration can be even reformulated as an optimization problem that can be efficiently solved using Newton-like method. A significant issue for the success of such method is the choice of the tearing variables. Usually domain-specific knowledge is used for

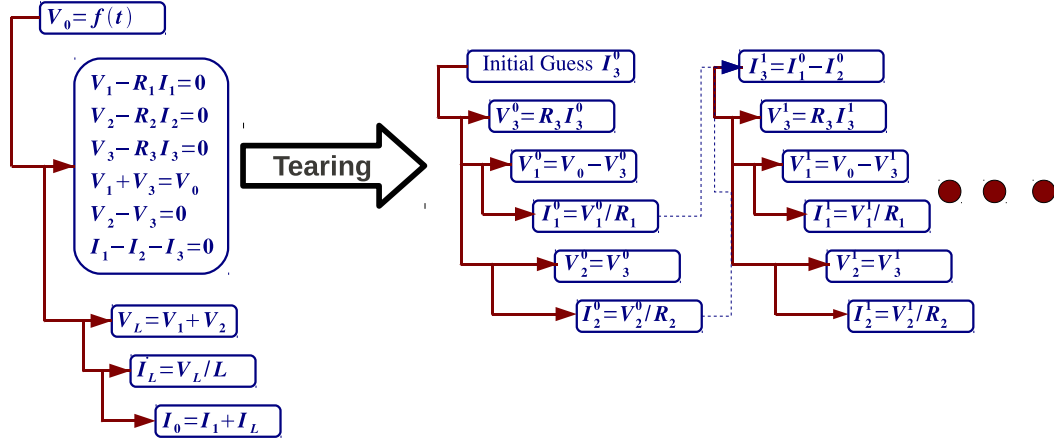


Figure 3.5.: Left: The equation system consists of a large algebraic loop
Right: Breaking the algebraic loop by the tearing algorithm

choosing successful candidates. (Elmqvist and Otter 1994) presents a general strategy for picking tear variables. However, in general there is no guarantee that such algorithm would always work. In appendix D a global optimization multistart strategy is presented. This strategy utilizes the idea of tearing algorithms for breaking parameter estimation of large DAE systems with cyclic structure into smaller parameter estimation subproblems (Elsheikh et al. 2009).

Chapter 4.

Mathematical Aspects behind Modelica

This chapter introduces basic mathematical terminologies for numerical simulation of Modelica models. These Modelica models may follow a variety of modeling paradigms as mentioned (cf. section 2.1). Section 4.1 fixes the mathematical representation of the set of Modelica models on which conclusions made in this work are valid. Section 4.2 discusses some solvability issues of DAE systems in general and emphasizes the basic differences with ODE systems. Section 4.3 introduces the algorithmic techniques applied by standard Modelica compilers for transforming a general DAE system into a numerically solvable DAE system. Finally, numerical aspects behind common ODE/DAE solvers utilized by standard Modelica simulation environments are discussed in section 4.4.

4.1. A mathematical representation of Modelica models

In chapter 3 it is shown how a Modelica model, given in a graphical specification, is transformed into pure mathematical formulation. In general, for Modelica models based on continuous-time scale, the resulting DAE system may have the form:

$$\begin{aligned} f(\dot{y}(t), y(t), z(t), p, t) &= 0 & , & \quad y(t_0) = y_0(p) \\ g(y(t), z(t), p, t) &= 0 & , & \quad z(t_0) = z_0(p) \end{aligned} \tag{4.1}$$

where $y \in R^{n_1}$ stands for state variables, $z \in R^{n_2}$ for the algebraic variables, $p \in R^m$ for parameters and t for time. Notice that

$$f(\dot{y}_0(p), y_0(p), z_0(p), p, t_0) = g(y_0(p), z_0(p), p, t_0) = 0$$

is only fulfilled if consistent initial conditions are present. Such initial conditions could be specified by the modeler which either need to be consistent or they can be made consistent by the Modelica compiler. Alternatively, the DAE system (4.1) can be rewritten as:

$$F(\dot{x}, x, p, t) = 0 \quad , \quad x(t_0) = x_0(p) \tag{4.2}$$

where $x = \begin{pmatrix} y \\ z \end{pmatrix} \in R^n$, $n = n_1 + n_2$ and $F : R^{n+m+1} \rightarrow R^n$. Equations (4.1) and (4.2) do not correspond to a generalized form of hybrid DAEs where events are present. Although some of the presented models in this thesis can be categorized under hybrid systems due

to the presence of discrete events in few ignorable limited cases, the conclusions made in this thesis are basically drawn on models based on continuous-time scale. Nevertheless, conclusions regarding generalized hybrid systems are avoided.

4.2. Handling general DAE systems

Given an explicit ODE system

$$\dot{v} = h(v, p, t) \quad , \quad v(t_0) = v_0 \quad (4.3)$$

it can be solved by *integrating* the continuous-time state variables from their derivatives, expressed by the ODE right-hand side in equation (4.3) (eg. explicit Euler formula). For a DAE system like (4.1), the function g expresses algebraic constraints among state variables since $\partial g / \partial \dot{y} = 0$ or equivalently $|\partial F / \partial \dot{x}| = 0$ in equation (4.2). Applying integration alone leads in general to a structurally singular problem except for special cases. In order to solve a DAE system, not only integration but also differentiation should be included, in order to reformulate the system into a solvable ODE system. In this case, a selected subset of algebraic equations gets differentiated in order to extend the constraints among state variables and their derivatives and hence to enable integration.

Example 4.4.

$$\begin{array}{ccc} y_1 = t^2 & \xrightarrow{\text{diff. 1st eq.}} & \dot{y}_1 = 2t \text{ , } y_1(0) = 0 \\ \dot{y}_1 = y_2 & & \xrightarrow{\text{diff. 2nd eq.}} \dot{y}_1 = 2t \text{ , } y_1(0) = 0 \\ & & y_2 = 2t \quad \dot{y}_2 = 2 \text{ , } y_2(0) = 0 \end{array}$$

The DAE system is transformed to an ODE system with consistent initial conditions.

The solution process within the last example aims actually at the reduction of the so-called *Differential Index* defined as follows (Brenan et al. 1989):

Definition 4.5 (The Differential Index of a DAE System). *For a general DAE system (4.2) the differential index along a solution $x(t)$ is the minimum number of differentiations of the system which would be required together with the help of algebraic substitutions to derive \dot{x} uniquely as a continuous function in terms of x, p and t . Thus, the index is defined in terms of the overdetermined system*

$$\begin{aligned} F(\dot{x}, x, p, t) &= 0 \\ \frac{dF}{dt}(\dot{x}, x, p, t) &= 0 \\ &\vdots \\ \frac{d^q F}{dt^q}(\dot{x}, x, p, t) &= 0 \end{aligned} \quad (4.6)$$

to be the smallest q so that \dot{x} can be solved in terms of x, p and t .

Informally, the differential index of a DAE system is the minimum number of times that subsets of equations need to be differentiated to get transformed to an ODE system. Using the above definition, the DAE system (4.2) could be transformed into an ODE system by differentiating all equations w.r.t. time to obtain:

$$F_{\ddot{x}} \cdot \ddot{x} + F_{\dot{x}} \cdot \dot{x} + \dot{F} = 0 \quad (4.7)$$

\dot{x} can be expressed as a function in x, p and t only if F_x is nonsingular, otherwise the differential index is more than one. In this case some of the equations in (4.7) are used for reducing the index by substituting them in equation (4.2). This process is done iteratively until a solvable ODE system is obtained. Computationally, it is difficult to determine the differential index in this manner.

4.3. Index reduction using the structural index

Typical Modelica models are represented by finite number of components with many connections. These connections represent algebraic constraints among state variables as illustrated in chapter 2. Therefore, Modelica models can be of higher index. Nevertheless, different representation of the same model can generate DAE systems with lower index. In general, the index of a DAE system is not a property of the modeled system but a property of the model representation and therefore a function of the modeling methodology (Fritzson 2003). Domain specific tools can utilize domain specific rules for applying index reduction. However, these domain specific rules don't fit into the Modelica concept and violate the generalized methodology adopted by Modelica. From one side, it is then superior to have such higher index representation from modeling perspective. From the other side, reliable general algorithms for index reduction need to be followed.

Computationally, it is not practical to reduce a DAE system into an ODE system using the standard definition 4.5 of the differential index. An efficient alternative is to reduce the so-called *structural index* used to approximate the differential index defined through Pantelides algorithm (Pantelides 1988). This algorithm is used for selecting the equations subject to differentiation and algebraic substitution targeted at reducing the structural index with the help of graph matching algorithms. In this way, DAE systems with high index can be mechanically transformed into solvable ODE systems. Nevertheless, while the differential index is of a numerical nature (see definition 4.5), the structural index relies purely on the topology of the bipartite graph representation of the underlying DAE-model. Consequently, the structural index may not reflect the true differential index corresponding to hidden numerical singularities of the Jacobian that cannot be captured by the topology alone. The relationship of structural index

Chapter 4. Mathematical Aspects behind Modelica

and differential index was a source of confusion in literature (Reissig et al. 1999), but in general the structural index is not necessarily equal to the differential index, though both are equal in many practical cases. The following definitions are needed for defining the structural index (Leitold and Hangos 2001):

Definition 4.8 (Matching). *Given a graph $G = (V, E, w)$, where V and E are sets of vertices and edges respectively. $w : E \rightarrow N$ is a pre-given weight function. A Matching $M^s(G)$ is a group of s edges:*

$$M^s(G) = \{e_i \in E, |M^s(G)| = s\},$$

*s.t. no two edges in $M^s(G)$ are incident in a vertex. A **perfect Matching** is a matching where all vertices are covered. The **Weight of a Matching** $M^s(G)$ is given by:*

$$w(M^s(G)) = \sum_{i=1}^s w(e_i), \quad e_i \in M^s(G)$$

*A **Maximal Matching of size s** $M_{max}^s(G)$ is a matching where $w(M_{max}^s(G)) \geq w(M_k^s(G)) \forall$ possible matchings $M_k^s(G)$ of size s .*

Definition 4.9 (Structural Index of a DAE-System). *Given the DAE system (4.2) and let $G = (V, E, w)$ be the bipartite graph representation of the DAE System (definition 3.1) with a weight function $w : E \rightarrow N$ defined as follows:*

$$w((e_i, x_j)) = \text{The highest time derivative order of variable } x_j \in V_x \text{ in equation } e_i \in V_e$$

The structural index is given by:

$$I_{str}(G) = w(M_{max}^{n-1}(G)) - w(M_{max}^n(G)) + 1$$

Remark 4.10. *The number of variables in equation (4.2) is equal to n , number of equations is also n and hence a perfect matching has to be of size n .*

By identifying maximal matchings within bipartite graphs (Uno 2001, Frenkel et al. 2012) Index reduction is performed using Pantelides algorithm combined with the method of dummy derivatives (Mattsson and Söderlind 1993) roughly performed as follows:

4.4. Numerical methods for ODE/DAE systems

1. Identify $M_{max}^n(G) = \max_i M_i^n(G)$ and similarly $M_{max}^{n-1}(G) = \max_j M_j^{n-1}(G)$.
2. Compute $I_{str}(G) = w(M_{max}^{n-1}(G) - w(M_{max}^n(G)) + 1$
3. $e' = \frac{d}{dt}e$, for the unique $e \in M_{max}^n(G)$ and $e \notin M_{max}^{n-1}(G)$
4. augment the DAE system with e' . However, the DAE system becomes overdetermined since the number of equations is more than the number of variables.
5. replace one of the differentiated algebraic variable with new dummy algebraic variable. The DAE system becomes consistent

The structural index of the new system is decremented and the process is recursively repeated until a solvable system is achieved. Polynomial runtime graph matching algorithms are applicable on bipartite graph representation.

4.4. Numerical methods for ODE/DAE systems

In this section, numerical techniques used by common Modelica simulation environment for solving resulting ODE/DAE systems are discussed. Some numerical techniques for solving ODE systems can be extended to DAE systems of index one by viewing a DAE system like (4.1) as a regularized ODE system:

$$\begin{aligned} f(\dot{y}(t), y(t), z(t), p, t) &= 0 & , & \quad y(t_0) = y_0(p) \\ g(y(t), z(t), p, t) &= \epsilon \dot{z}(t) & , & \quad z(t_0) = z_0(p) \end{aligned} \quad (4.11)$$

and assume that $0 < \epsilon \ll 1$ and $|g_y| \neq 0$. This is a very stiff ODE system and hence ODE stiff methods can be applied to (4.11) with $\epsilon \rightarrow 0$ for obtaining a feasible solution (Gear, C. W. 1971). On the other side, general higher index DAE systems cannot be considered in this way as there are no DAE solvers for DAE systems with arbitrary differential indices. In this case, the differential index should be reduced to one or zero possibly with the help of AD techniques (Campbell 1994).

ODE/DAE systems generated from typical Modelica models are usually sparse, can be stiff and may require event handling. Consequently, common Modelica simulation environments tend to employ generalized implicit ODE/DAE solvers which are generally applicable to all possibilities. These generalized solvers employ, among others, implicit multistep methods like the ABM for non-stiff systems or BDF with wider stability region for stiff systems (Cellier 1991). In order to illustrate the underlying numerical integration process, for the sake of simplicity the numerical solution of the explicit ODE system (4.3) is demonstrated. Nevertheless numerical solvers for implicit ODEs or DAEs index one systems can be also used. The numerical integration process begins with an iteration of the form (Cohen and Hindmarsh 1996):

$$\sum_{i=0}^{k_1} \left[\alpha_{n,i} v_{n-i} + \delta_n \sum_{i=0}^{k_2} \beta_{n,i} \dot{v}_{n-i} \right] = 0 \quad , \quad \alpha_{n,0} = -1 \quad (4.12)$$

for solving for v_n at time step t_n with step size δ_n . $\alpha_{n,i}$ and $\beta_{n,i}$ are variable coefficients depending on the variable order q_n at time t_n to meet accuracy conditions in an efficient manner. For ABM, $k_1 = 1$ and $k_2 = q_n - 1$ while for BDF, $k_1 = q_n$ and $k_2 = 0$. At each time step t_n , a large nonlinear equation system of the form

$$\begin{aligned} G(v_n) &= v_n - \delta_n \beta_{n,0} h(v_n, p, t_n) - a_n = 0 \\ a_n &= \sum_{i>0} [\alpha_{n,i} v_{n-i} + \delta_n \beta_{n,i} \dot{v}_{n-i}] \end{aligned} \quad (4.13)$$

need to be solved. This is done by solving a Gauss-Newton scheme of the form:

$$M [v_n^{m+1} - v_n^m] = -G(v_n^m) \quad (4.14)$$

with an initial guess v_n^0 estimated from available history data and:

$$M \approx \frac{\partial G}{\partial v} = I - \delta_n \beta_{n,0} \frac{\partial h}{\partial v} \quad (4.15)$$

The resulting system can be sparse and ill-conditioned. Thus, an efficient Modelica simulation environment would attempt to use an efficient sparse iterative solver employing preconditioning techniques (Saad 2003). In this context, an ideal preconditioning matrix is chosen to be a fairly good approximation of the Jacobian so that factorization becomes cheap and avoidable for many following time steps (Brown and Hindmarsh 1986). Furthermore, $\frac{\partial h}{\partial v}$ should be analytically computed which is generally recommended both for performance and accuracy (Brenan et al. 1989). In this context, AD techniques can be employed for computing the Jacobian as done with Dymola (Olsson et al. 2005) and OpenModelica (Braun and Bachmann 2011). Missing any of the mentioned aspects may lead to significant reduction in the quality of the solution. This may also explain the wide gap in performance among various Modelica simulation environments. For instance, the Dymola simulation environment employs among others the following solvers:

- *LSODAR* (Hindmarsh 1983): uses ABM for non-stiff systems. It switches automatically to BDF if any stiffness is detected. The solver can also handle events with the help of root finding algorithms.
- *DASSL*: (Brenan et al. 1989): the default solver for many simulation environments. It is specially designed for solving DAE index one systems. It implements BDF method and provides the opportunity to supply a routine for analytical Jacobian. Many variations and descendant of this solvers exist for handling more special cases.

Using such generalized solvers has two perspectives. From one side, many physical models can be then reliably simulated. From the other side specialized features of the system may not be detected for improved efficient solution. Nevertheless, the gap between run-time performance of using specialized solvers for special kind of problems and that of generalized solvers can be reduced in many cases by employing symbolic and graph algorithms (cf. chapter 3). Some Modelica simulation environments provide solver prototyping capability to allow end-user employing his own solver for specialized problems (Claeys et al. 2006).

Part II.

Fundamentals in Systems Biology

Chapter 5.

Basic Concepts of Systems Biology

This chapter introduces some fundamentals of Systems Biology, the main domain of the modeling applications handled in this work. Section 5.1 introduces some basic concepts in the fields Systems Biology and Metabolic Engineering. Section 5.2 presents a basic overview of enzyme kinetics as an important topic necessary for understanding all later chapters within this part. Such enzyme kinetics are used for describing enzymatic reactions, the most elementary components on which the models handled in this work rely. Additionally, to gain some insights into the implemented advanced tools, a basic understanding of the phenomenological meaning behind enzyme kinetics is needed. In particular, the variety of enzyme kinetics and the underlying mechanisms behind them are emphasized.

5.1. Basic Concepts

5.1.1. Systems Biology

The main building block of all living organisms is the cell. Whether a cell belongs to a multicellular or unicellular organism, all cells are structurally similar and they perform similar functions for accomplishing vital life activities (Alberts et al. 2002). The field of Systems Biology aims at understanding complex cellular processes by studying the underlying biological components and their mutual interactions. These interactions give rise to the activities, the functions and the behavior of the cell (Kitano 2002). One of the stated goals of this field is the discovery of hidden information through a cyclic procedure composed of the following steps (Klipp et al. 2005):

- Proposition of testable hypotheses based on theoretical and gained knowledge
- Validation by experiments
- Quantitative description of data through mathematical models

These steps cover many highly-interrelated levels of cellular processes:

1. Genome level: proteins among other organism characteristics are coded by the DNA
2. Transcriptome level: DNA sequences are copied by an intermediate carrier, the mRNA to get translated to amino acids and proteins (enzymes)

3. Proteome level: the present set of proteins and their amounts determine which cellular activities take place and how they get performed
4. Metabolome level: all cellular biochemical enzymatic reactions that take place for substrate uptake (e.g. Glucose), energy building (*Catabolism*), performing vital processes and constructing cell structures (*Anabolism*)
5. Fluxome level: the reaction paths by which a substrate is converted into a product

The regulation among these levels is very complex. From one side, the construction plans and the amount of enzymes in the cell are coded by the DNA. From the other side, the metabolism influences the cell functions at genome level resulting in feed-back loops. Moreover, another aspect that contributes to the complexity is that the cellular processes are operating on completely different time scales. While an enzymatic reaction can be recorded in milliseconds scaler, the influence of a genetic modification may take hours. Consequently, the complexity of such *regulatory networks* makes the aims of Systems Biology very challenging since raw data corresponding to all mentioned levels (so called omics-data) need to be understood in an integrative manner. Modeling is considered as a vital assisting tool for resolving such challenges. From one side, further physical meaning and in-depth knowledge can be extracted from the available pure data. Additionally, with the help of validated models, further hidden information can be gained from the underlying system under study.

5.1.2. Metabolic Engineering

Microorganisms like bacteria, fungi, algae and protozoans are tiny organisms of significant importance for biotechnological applications. They have been utilized since thousands of years in producing vinegar, bread, alcoholic beverages, milk products and others. With the raise of DNA technologies, the scope of biotechnological applications has been tremendously enlarged through the field of *Metabolic Engineering*. In (Stephanopoulos et al. 1998), this field is defined as *the directed improvement of production formation or cellular properties through the modification of specific biochemical reaction(s) or the introduction of new one(s) with the use of recombinant DNA technology*. That is, the considered organisms are viewed as elementary chemical factories by engineering the metabolism through genetic modifications to achieve desired goals like:

- reduction of cellular energy use or waste production
- control enzyme activity or substrate selectivity

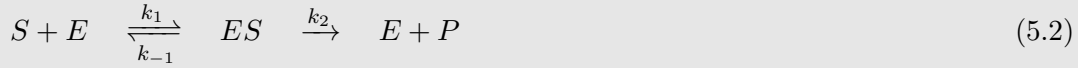
Many biotechnological products vital for agricultural and environmental technologies, food-, pharmacy- and medicine industries involve Metabolic Engineering methods. For industry-relevant microorganisms cultured in a large-scale level, applications of metabolic engineering attempt to employ genetically modified strains of the selected microorganism for improved productivity in terms of costs, quality and quantity.

In order to achieve successful genetic modifications, it should be known how such modifications would influence the targeted regulatory networks. The complexity of cellular processes and the interaction of their regulation processes limit the impact of genetic modifications if done in an ad-hoc fashion. Consequently, a model-based predictive metabolic engineering benefits from the field of Systems Biology. By applying systematic concepts and tools from the field of Systems Biology, directed genetic modifications can be examined and predicted. Additionally, validated models can assist the identification of targets for Metabolic Engineering based on quantitative information rather than performing genetic modifications based on intuitive decisions.

5.2. Enzyme kinetics

Vital cellular processes are performed according to the present set of enzymatic reactions at the metabolism level. The main elements of such networks (cf. section 6.1) are the involved enzymatic reactions. An *enzymatic reaction* is catalyzed by a specific enzyme (say E). The molecules of such enzymes can bind only to a specific set of substrates molecules (say S). The substrate molecules within the substrate-enzyme complex ES are vastly transformed to the product molecules (say P). Such reactions can be mathematically modeled by decomposing this process into more elementary steps as in the following simple example.

Example 5.1. *The schema of the simple irreversible uniuni reaction with one substrate S and one product P catalyzed by an enzyme E looks as follows:*



k_{-1} is a rate constant describing the decomposition rate of the complex ES into E and S . k_1 and k_2 are analogous rate constants. The concentration changes of the reactants can be described by the following ODEs:

$$\begin{aligned} \frac{d[S]}{dt} &= -k_1[S][E] \\ \frac{d[E]}{dt} &= -k_1[S][E] + (k_{-1} + k_2)[ES] \\ \frac{d[ES]}{dt} &= k_1[S][E] - (k_{-1} + k_2)[ES] \\ \frac{d[P]}{dt} &= k_2[ES] = v \end{aligned} \quad (5.3)$$

The reaction rate v is equal to the rate of product formation.

The previous ODE system is nonlinear and is not analytically solvable in this form. In the following subsections, some analytically derived nonlinear algebraic functions, the so called *enzyme kinetics* for describing enzymatic reactions v are presented.

5.2.1. Mechanistic kinetics

Mechanistic kinetics aim at describing enzymatic mechanisms behind enzymatic reactions. These kinetics are analytically derived from rate equations as in (5.3) by taking additional assumptions regarding to the underlying enzymatic mechanism (or equivalently the rate constants) (Bisswanger 2002).

Michaelis-Menten kinetics

For the simple reaction (5.2), Michaelis and Menten observed that the conversion of E and S to ES and vice versa is much faster than the decomposition of ES into E and P (i.e. $k_1, k_{-1} \gg k_2$). Equivalently, under the assumptions of quasi-equilibrium conditions $d[E]/dt = d[ES]/dt = 0$ and that the total amount of enzyme $[E]_0 = [E] + [ES]$ is constant, the ODE system (5.3) becomes analytically solvable and results in the so-called Michaelis-Menten kinetic:

$$v = \frac{k_2[E]_0[S]}{\frac{k_{-1}+k_2}{k_1} + [S]} = \frac{V_{max}[S]}{K_m + [S]} \quad (5.4)$$

From the previous equation, the following statements can be made:

1. for $[S] \ll K_m$: v increases linearly w.r.t. $[S]$ implying that substrate molecules are more likely to bind with many existing free enzyme molecules
2. for $[S] \gg K_m$: v approaches its asymptotically upper bound V_{max} as more enzyme molecules are bound with existing substrate molecules

The parameter K_m is equal to the substrate concentration that yields the half-maximal reaction rate $V_{max}/2$. These two parameters represent important enzymatic characteristics demonstrating how quickly the enzyme becomes saturated and what its maximum activity is. For reversible uni-uni reactions, the following kinetic can be analytically derived in a similar manner

$$v = \frac{V_{max}^{fwd}[S]}{K_{mS} + [S]} - \frac{V_{max}^{bwd}[P]}{K_{mP} + [P]} \quad (5.5)$$

Similarly, V_{max}^{fwd} and V_{max}^{bwd} denote the maximal possible reaction activity in forward and backward directions, respectively. The parameters K_{mS} and K_{mP} denote the substrate and product concentration causing half maximal forward and backward reaction rates, respectively.

Reactions with Effectors

Enzymatic activities can be influenced by *effectors* by which the formation rate of a reaction's product is changed. A blocking influence is referred to as *inhibition* while an

accelerating influence is called *activation*. For an irreversible reaction inhibited by I , the derivation of a mechanistic kinetic may lead to:

$$v = \frac{V_{max}[S]}{K_m(1 + [I]/K_I) + [S]} \quad (5.6)$$

Where K_I is a parameter that expresses the ratio of EI formation to EI decomposition. Therefore, v decreases with larger K_I . This kinetic states that v decreases with an increasing amount of I 's molecules. This equation corresponds to the so-called *competitive inhibition* where the molecules of an inhibitor I compete with the molecules of a substrate S for binding with the molecules of the enzyme E . In this case, the release of P is blocked by I . Further types of inhibition mechanisms are distinguished by mechanistic kinetics according to whether

- the inhibitor binds to the complex ES
- the inhibitor binds to S , see figure 5.1 (Tillack 2008)
- the reversibility of the inhibition

The resulting kinetics are distinguished by the formulation and kinetic parameters.

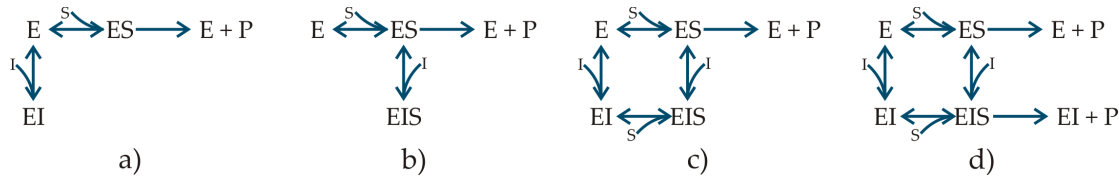


Figure 5.1.: Various enzyme inhibition mechanisms

Multi-substrate reactions

Most cellular reactions involve more than one substrate and one product. Mechanistic kinetics become more sophisticated. Their analytical derivation additionally considers the sequence in which substrates bind and products are released. For example, within a bi-bi reaction, (two substrates S_1, S_2 and two products P_1, P_2) the underlying enzymatic mechanisms some of which shown in figure 5.2 are distinguished according to whether binding to enzyme is done

- in random order, (i.e. E binds with both of S_1 and S_2 in any order)
- in a sequential order, (i.e. S_2 binds only with the complex ES_1)
- alternate binding of substrates and release of products (ping-pong mechanisms)

- which intermediate complexes are formed (only ES_1 , ES_2 or also ES_1S_2)
- Interactions among reactants (e.g. inhibition through product formation)

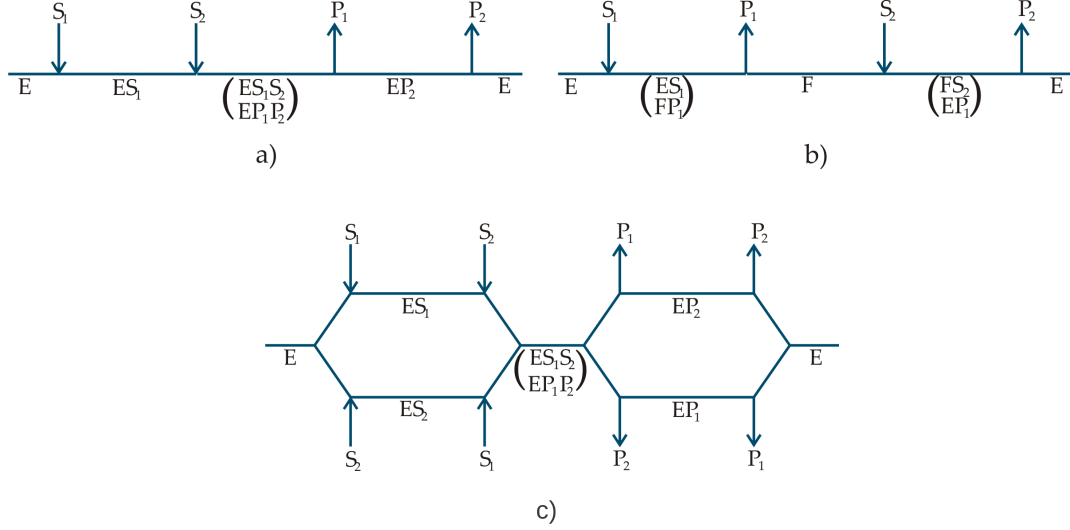
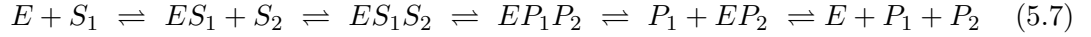


Figure 5.2.: Multi-substrate enzyme mechanisms: a) Sequential b) Ping-Pong c) Random

For example, the kinetic of an ordered bi-bi reaction (i.e. binding in a specific order)



is described with the equation:

$$v = \frac{V_{max}[S_1][S_2]}{K_{iS_1}K_{mS_2} + K_{mS_2}[S_1] + [S_1][S_2]} \quad (5.8)$$

Interestingly, the previous kinetic format is a special case of the general kinetic formula of random bi-bi mechanism revealing that ordered mechanisms are special cases of the general random mechanisms. In summary, an enormous number of equation patterns corresponding to combinatorially large number of enzymatic mechanisms exist. As it is shown later, this causes challenging difficulties in the modeling process since hundreds of components need to be separately implemented for expressing different enzyme binding mechanisms.

5.2.2. Kinetic formats

As already shown, mechanistic kinetics characterize detailed description of the underlying enzymatic mechanisms. These kinetics pose however some problems when used for

describing enzymatic reactions within cellular environment. Under crowded conditions within a cellular environments, a lot of effectors may influence the enzyme activity. When considering all these effectors and other typical interactions, the corresponding kinetic becomes very complex and over-parametrized. Parameter dependencies are enhanced when estimating the parameters with experimentally generated data. This argument motivates the employment of the s.c. generalized kinetics which rely on more simplified assumptions two of which are introduced. The first type is the so-called *convenience kinetics* which assumes a reversible rapid equilibrium with random binding mechanism (Liebermeister and Klipp 2006). In this way, the corresponding mechanistic kinetic of any reaction with arbitrary number of substrates S_i , products P_j , inhibitors I_b and activators A_a becomes:

$$v = K_E \cdot \prod_a \frac{K_{A_a} + [A_a]}{K_{A_a}} \cdot \prod_b \frac{K_{I_b}}{K_{I_b} + [I_b]} \cdot \frac{V_{max}^{fwd} \prod_i \frac{[S_i]}{K_{mS_i}} - V_{max}^{bwd} \prod_j \frac{[P_j]}{K_{mP_j}}}{\prod_i \left(1 + \frac{[S_i]}{K_{mS_i}}\right) + \prod_j \left(1 + \frac{[P_j]}{K_{mP_j}}\right) - 1} \quad (5.9)$$

Another kinetic format is the *linlog* kinetic formulated as:

$$v = v^0 + \sum_i \alpha_i \cdot \ln\left(\frac{[S_i]}{S_i^0}\right) + \sum_j \beta_j \cdot \ln\left(\frac{[P_j]}{P_j^0}\right) + \sum_a \gamma_a \cdot \ln\left(\frac{[A_a]}{A_a^0}\right) + \sum_b \delta_b \cdot \ln\left(\frac{[I_b]}{I_b^0}\right) \quad (5.10)$$

In contrary to usual mechanistic parameters, which provide descriptive physical insights into enzymatic mechanisms, linlog parameters are based on scaled sensitivities describing the influence of characteristic changes of enzymes on a referenced reaction rate at a reference steady-state v^0 (Heijnen 2005).

One of the main advantages of the presented kinetics in the context of this work is that they are expressed in terms of generalized structured formulas very adequate for compact implementation (Elsheikh 2012) (cf. chapter 8) and automatic generation of highly complex models (cf. chapters 9 and 10). However, one of the limitation of such kinetics is that they may not describe the enzymatic behavior accurately in some boundary cases. A comparative study has been presented in (Hadlich et al. 2009).

Chapter 6.

Modeling Examples

This chapter demonstrates several modeling examples from the domain of Systems Biology. Section 6.1 introduces the standard biochemical network modeling approach for describing the dynamics of the cellular metabolism. These models are based on enzyme kinetics presented in the previous chapter. In section 6.2, the introduced models are extended for describing labeled biochemical networks as an important tool for flux quantification. Section 6.3 presents the vertical modeling approach for linking the metabolism together with the genome level. Finally, all models handled in this work are briefly summarized.

6.1. Modeling biochemical reaction networks

Biochemical reaction networks are used for describing cellular processes of the metabolism. Figure 6.1 demonstrates a typical biochemical network of enzymatic reactions termed as "Spirallus" which represents an abstraction of the Tri-Carboxylic Acid (TCA) cycle (Wahl 2007). The set of freely distributed metabolites A, B, C, D, E and F are viewed as pools, while the reactions are viewed as intermediate edges among the metabolites. With the presence of substrates being taken up through the initial reaction v_{upt} , intermediate reactions becomes active and the two products E_{ex}, F_{ex} get produced as long as enough substrate molecules are taken up (or absorbed). The reactions v_{upt}, v_3 and v_4 are inhibited by the molecules of the metabolites A, D and C , respectively.

Mathematical structure

Biochemical network modeling is used to describe the dynamics of molecular species and their interactions within cellular environment. The models are usually based on the continuum¹ and homogeneity assumptions². The law of mass conservation³ is used for describing the rate of change in the mass of intermediate metabolites in a biochemical network (Wiechert et al. 2010). These models have more or less the following structure:

$$\dot{c} = N \cdot v(c, \alpha), \quad c(0) = c_0 \tag{6.1}$$

¹All chemical species involved have such a high copy number to be described by a continuous concentration value

²Diffusion processes are so fast that concentrations can be considered to be spatially homogeneous

³the mass within a closed system remains constant over time

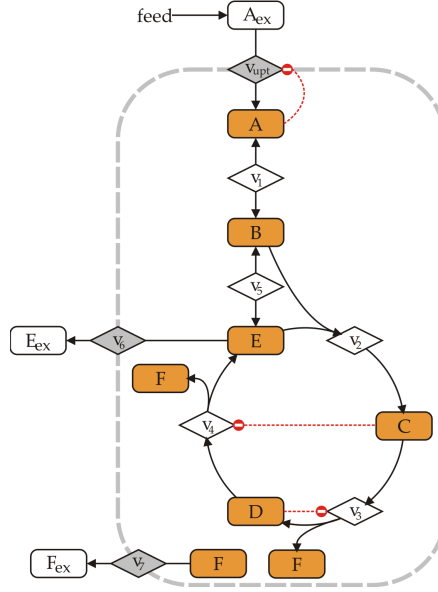


Figure 6.1.: An abstraction of TCA cycle.

where $c \in R^m$ stands for vectors of the metabolite concentrations, $v = v(c, p) \in R^n$ is a vector of reaction rates described by enzyme kinetics, α is kinetic parameters vector describing enzyme characteristics and $N \in R^{m \times n}$ is the reactions stoichiometry describing the number of molecules in any single reaction.

Example 6.2. *The dynamics of intracellular metabolites within the simple network of figure 6.1 are described with the DAE system:*

$$\begin{aligned}
 [\dot{A}] &= v_{upt} - v_1 & [\dot{B}] &= v_1 - v_2 - v_5 \\
 [\dot{C}] &= v_2 - v_3 & [\dot{D}] &= v_3 - v_4 \\
 [\dot{E}] &= v_4 + v_5 - v_2 - v_6 & [\dot{F}] &= v_3 + v_4 - v_7 \\
 [\dot{E}]_{ex} &= v_6 & [\dot{F}]_{ex} &= v_7
 \end{aligned}$$

Reaction rates v_i are expressed using convenience kinetics. For example v_1 becomes:

$$v_1 = \frac{V_{max,1}^{fwd} \cdot [A]/K_{mA,1} - V_{max,1}^{bwd} \cdot [B]/K_{mB,1}}{(1 + [A]/K_{mA,1}) + (1 + [B]/K_{mB,1}) - 1}$$

The above system can be rewritten in matrix form as in equation (6.1) with:

$$N = \begin{array}{c} A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} v_f \quad v_u \quad v_1 \quad v_2 \quad v_3 \quad v_4 \quad v_5 \quad v_6 \quad v_7 \\ \begin{pmatrix} 0 & +1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & +1 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & +1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & +1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & +1 & +1 & -1 & 0 \\ 0 & 0 & 0 & 0 & +1 & +1 & 0 & 0 & -1 \end{pmatrix} \end{array}$$

In real-life applications, intracellular metabolites concentration data obtained by *stimulus response experiments*⁴ (SRE) form the basis for constructing and validating biochemical network models (Oldiges and Takors 2005). These models quantify the dynamics of the considered reaction networks and give more insights into the underlying cellular processes.

6.2. Modeling labeled biochemical reaction networks

While methodologies for obtaining dynamic intracellular metabolites concentration data are available, there is no direct way for measuring reaction rates. Instead, *metabolic flux analysis* (MFA) techniques are used to quantify the fluxes of a biochemical network under quasi-steady state assumption (i.e. determining the reaction rates v under the assumption that $dc/dt = 0$) experimentally achieved by a *chemostat* process. In such a process, a strain of microorganisms in a biochemical reactor is feed by a fixed amount of substrate in a way that the substrate concentration is assumed to be constant. Mathematically, for certain subsets of biochemical reaction networks of linear non-cyclic structure, the set of all admissible fluxes known as the *elementary flux modes* (EFM) are represented within the null space of the stoichiometry matrix N , (i.e. the solution of $N \cdot v = 0$, see equation (6.1)). This technique is so limited in the sense that the number of reactions is usually more than the number of metabolites, i.e. $m > n$ and hence there are infinitely many solutions. In the Spirallus network of figure 6.1, by measuring the extracellular fluxes i.e. the substrate influx v_{feed} and the product outfluxes v_6 and v_7 , m becomes equal to n and only one exact solution for fluxes exist.

¹³C MFA

¹³C MFA is used to determine the unknown fluxes by employing ¹²C and ¹³C *isotopomers*, used as labeling tracers in biochemical networks (Wiechert 2001). *Isotopes* are different types of atoms of the same chemical element. *Isomers* are compounds that have identical molecular formula but have different atoms arrangements. Isotopomers are isotopic

⁴Disturb the metabolism of a cultivated strain of microorganisms in a biochemical reactor by a sudden pulse of substrate mostly Glucose

isomers. The isotope ^{12}C is considered to be unlabeled (0) while ^{13}C is considered to be labeled (1). For isotopomers of n isotopes, there are 2^n different possible arrangements (i.e. labels). Within a chemostat process with labeled substrates, such labels in a biochemical network can be traced for determining the fluxes.

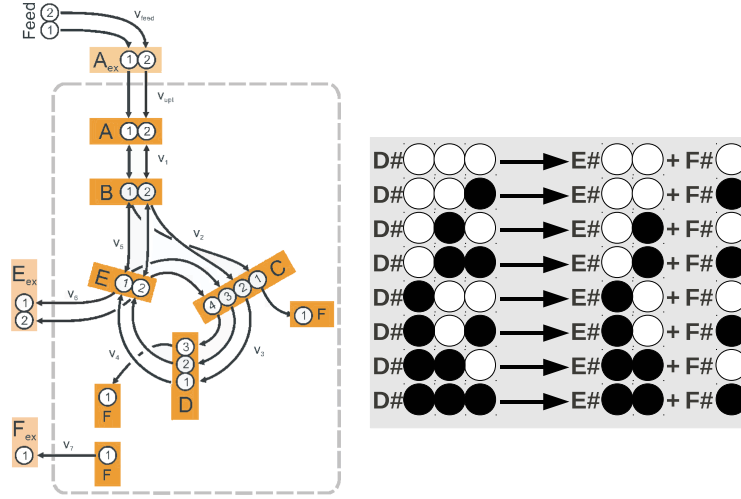


Figure 6.2.: TCA-Cycle model with labeled isotopomers

Figure 6.2 shows the path of isotopomers within the Spirallus network. A metabolite molecule D marked by an isotopomer of the arrangement ^{13}C - ^{12}C - ^{13}C is termed as $D\#101$. It can be noticed that a labeled metabolite molecule $E\#01$ is formed only through $D\#01x$, $x \in \{0, 1\}$. The mathematical representation of labeled networks models are extended from equation (6.1) by considering each distinguishable labeled metabolite to be a stand-alone substance that can be measured independently (Wiechert and de Graaf 1997). In this case, for each labeled metabolite, a corresponding rate equation is present and hence the size of the DAE system becomes so large.

Example 6.3. *There are 42 rate equations needed for representing the concentration changes of labeled metabolites from the Spirallus network of figure 6.2. For example, the concentration changes of labeled metabolites D are represented by 8 rate equations:*

$$\frac{dD\#ijk}{dt} = c_{0ijk} \cdot v_3 + c_{1ijk} \cdot v_3 - f_i \cdot v_4 - e_{jk} \cdot v_4 \quad , \quad D\#ijk(0) = D_0\#ijk$$

with

$$c_{xijk} = \frac{C\#xijk}{C} \quad , \quad f_i = \frac{F\#i}{F} \quad , \quad e_{jk} = \frac{E\#jk}{E} \quad \forall x, i, j, k \in \{0, 1\}$$

The reaction rates v_i remain exactly as in example 6.2. The whole DAE system is of size 102 in comparison with only 16 equations for the simple Spirallus network.

Models of labeled biochemical networks can correspond to two types of experiments:

1. *stationary* ^{13}C MFA, the underlying networks are stationary both at metabolic and isotopic levels (i.e. $dc/dt = dc\#i/dt = 0 \forall$ possible i). In this case, the corresponding model does not include any differential equation.
2. *instationary* ^{13}C MFA where the underlying networks are stationary at metabolic level but instationary at isotopic level (i.e. $dc/dt = 0$, $dc\#i/dt \neq 0$)

With the availability of extracellular fluxes and labeled pool measurements, fluxes can be quantified using the resulting equations.

6.3. Vertical modeling

The approaches demonstrated so far correspond to a horizontal modeling approach, in which presented models cover only the metabolism level in the cell. Interpretation of single-level data without considering the network context (i.e. the regulation among different levels) may lead to wrong conclusions. These shortcomings are overcome by the vertical modeling approach in which various regulation levels are linked with each other. In (Noack 2009), models describing different regulation levels of the cell, in particular genome and metabolome focusing on the central metabolism of the citrate cycle were presented, cf. figure 6.3. The corresponding mathematical model includes the standard model representation of biochemical reaction networks as follows:

$$\dot{c} = N \cdot v(c, E, \alpha) \quad (6.4)$$

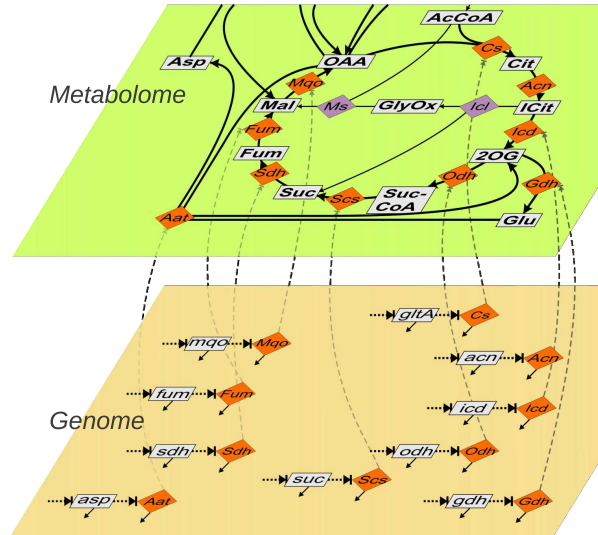


Figure 6.3.: Vertical model (Noack 2009)

Additionally, enzyme kinetic v is dependent on the amount of enzymes E controlled by the transcription process. The amount of enzymes is modeled by rate equations for describing the concentration change of each enzyme E_k :

$$\begin{aligned}\frac{d[E_k]}{dt} &= k_{tl}(\mu) \cdot mRNA_k - (k_{dE} + \mu) \cdot [E_k] \\ \frac{dRNA_k}{dt} &= k_{tc}(\mu) \cdot GP_k - (k_{d_{mRNA}} + \mu) \cdot mRNA_k\end{aligned}$$

Where k_* are relevant rate constants for describing the transcription process.

6.4. Summary of models

The models handled in this work are concerned with biochemical networks of simple *prokaryotes*⁵ like *Escherichia coli* and *Corynebacterium glutamicum*. In contrary to *eukaryotes*⁶, they have simpler structure due to the lack of nucleus and other membrane-bound organs. The genetic sequences of both organisms as well as their central metabolism are fully known and identified. Table 6.4 summarizes some of the models, their sizes and their various applications context used within this work. "PE" and "SA" stand for parameter estimation and sensitivity analysis, respectively. Only non-trivial equations are counted.

Table 6.1.: Summary of models

Model	Type	Dim	Par	Context
Spirallus	reaction network	25	45	Benchmark for SA
Coryne	reaction network	173	268	Benchmark for SA
EMP	instationary ¹³ C MFA	1382	335	Parallel SA
Spirallus	instationary ¹³ C MFA	95	54	Monte Carlo simulation of PE
Coryne	instationary ¹³ C MFA	2201	457	Monte Carlo simulation of PE
Visser	reaction network	19	19	kinetic variants
Haunschild	reaction network	16	16-20	enzymatic variants
TCA	reaction network	182	124	Control coefficients
C-Glutamicum	vertical model	518	234	Control coefficients

⁵comes from the Greek (pro, "before") + (karyon, "nut" or "kernel")

⁶comes from the Greek (eu, "good") and (karyon)

Part III.

Modeling Applications in Systems Biology with Modelica

Chapter 7.

Modelica as a Descriptive Language for Systems Biology

Biochemical network modeling is usually done using the *Systems Biology Markup Language* (SBML), the standard language established and maintained by the Systems Biology community. On the other hand, employing Modelica as a modern language for biochemical network modeling applications is still a new trend that is practiced from a small community. In (Wiechert et al. 2010), Modelica was compared with SBML in many aspects for showing the advantages and disadvantages of the underlying approaches of both sides. It was concluded that Modelica is capable of supporting large set of standard SBML-based applications. These conclusions are emphasized with further applications presented in this and next chapters. It is shown that efficient employment of powerful Modelica language constructs and existing developer-oriented modeling tools for model parsing and automatic model generation significantly simplifies the implementation of highly specialized tools for Systems Biology. Additionally, the expressiveness of Modelica constructs can reduce the gap of using universal modeling languages for specialized complex tasks using descriptive specifications with easy-to-derive semantics relevant for automatic model generation.

This chapter is structured as follows: Section 7.1 gives an overview of SBML. Section 7.2 introduces the Biochem library, the current non-standard¹ Modelica attempt for modeling biochemical reaction networks. Finally, section 7.3 lists the main conclusions made in (Wiechert et al. 2010) regarding the comparison between Modelica and SBML. The next chapters provide further comparative aspects under the light of further three contributions of this work:

1. An algorithmically differentiated Modelica library for generalized kinetics adequate for applications of automatic model generation
2. A highly-specialized Modelica-based editor for biochemical reaction networks
3. A Modelica-based approach for specifying model families, an important methodology for model validation

¹it is not a part of the standard Modelica library

7.1. SBML

SBML was established as a standardized model exchange language for biochemical network models. It is based on the extended markup language (XML) format and thus supplies a hierarchical structure for the exchange of biochemical network data (Hucka and et al 2003). A specific strength of SBML is that there is a numerous number of freely-available tools based on this language. In particular, tools for formulating, parsing, manipulating, analyzing and translating models and generating code for simulation are available. In order to give a quick overview about the structure of an SBML document, the simple example from figure 7.1 is presented. The different sections of a hierarchically structured SBML description are given as follows :

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <sbml xmlns="http://www.sbml.org/sbml/level2/version3" level="2"
  metaid="metaid_01" version="3">
- <model id="model_01" metaid="metaid_02" name="SimpleNetwork">
+ <annotation>
+ <listOfCompartments>
- <listOfSpecies>
  <species compartment="compartment_01" id="species_01"
    initialAmount="10.0" metaid="metaid_02" name="A" />
  <species compartment="compartment_01" id="species_02"
    metaid="metaid_03" name="B" />
</listOfSpecies>
- <listOfReactions>
  - <reaction id="reaction_01" metaid="metaid_04" name="v1" reversible="false">
    - <listOfReactants>
      <speciesReference name="A" species="species_01" />
    </listOfReactants>
    - <listOfProducts>
      <speciesReference name="B" species="species_02" />
    </listOfProducts>
    - <kineticLaw>
      + <math xmlns="http://www.w3.org/1998/Math/MathML">
      - <listOfParameters>
        <parameter id="parameter_01" name="vmax" value="10.0" />
        <parameter id="parameter_02" name="km" value="0.05" />
      </listOfParameters>
      </math>
    </kineticLaw>
    </reaction>
  </listOfReactions>
</model>
```

Figure 7.1.: Segment of an SBML file represent a part of the Spirallus network

1. The whole system might be spatially structured into different disjoined homogeneous compartments. A list of these compartments is given in the compartment section
2. The species section gives a list of all chemical substances involved in the reaction network (here: A, B, etc.). For each specie, it must be specified to which spatial compartment it belongs, its biochemical name and its initial concentration
3. All network reactions are arranged in the reaction section (here: v_1, v_2 , etc.). Each reaction definition includes a list of all reactants and products as well as a mathemat-

ical term for a kinetic model describing the biochemical conversion. Corresponding kinetic parameters are specified in the parameter list (here: $v_{max,1}$ etc.)

Given this information, the complete network in figure 6.1 can be assembled. Using an SBML based biochemical network modeling tool, the information can be supplied in a user friendly form from which the SBML document is automatically generated.

7.2. Modeling biochemical networks with the Biochem library

7.2.1. What is the Biochem library?

For modeling biochemical networks with Modelica, considerable efforts have been done by designing the Biochem library (Nilsson and Fritzson 2005). The Biochem library is an abstract general-purpose library for modeling biochemical network models. It provides only guidelines and design principles, (e.g. basic implementable interfaces and basic types) rather than a real implementation. According to the available publication, the Biochem library provides about 99 abstract reaction types under the restriction that a reaction can get connected to at most 3 substrates, 3 products and 1 effector. Out of these abstract types, many reaction kinetics can be derived. Within the library *Metabolic*, a published implementation of *Biochem*, at least 180 kinetics are implemented and classified according to the number of substrates and products within many sub-packages. If more than one effector is required, which is very realistic in real-life applications, the number of components corresponding to various kinetics would be so high. Aiming at advanced implementation of the Biochem library adequate for real-life network models, further concepts have been extended² some of which are:

- Cofactors³
- Reactions with more than one effector
- Further components for vertical modeling

Within this work, further extensions have been taken regarding

- *Mechanistic kinetics*: emphasizing implicit hierarchies for hierarchical modeling
- *Generalized kinetics*: exploiting generalized formulas for template modeling

This was particularly useful for supporting advanced applications based on automatic model generation presented in chapter 9 and 10. In particular, generalized implementation of linlog kinetics and convenience kinetics for anonymous number of reactants with the help of advanced constructs in Modelica have been carried out throughout this work (Elsheikh 2012), cf. chapter 8. The resulting library, ADGenKinetics (algorithmically differentiated Modelica library for generalized kinetics), is available online at the standard Modelica website⁴.

²by Stephan Noack, Institute of Bio- and Geosciences, research centre Jülich

³Reactants which participate in many reactions

⁴www.modelica.org/libraries/adgenkinetics.html

7.2.2. Overview of the Biochem library

Figure 7.2 presents an overview of a subset of a simplified implementation of the Biochem library. More implementation details are comprehensively given in (Nilsson and Fritzson 2005). With the provided types, an implementation of a reaction kinetic usually extends

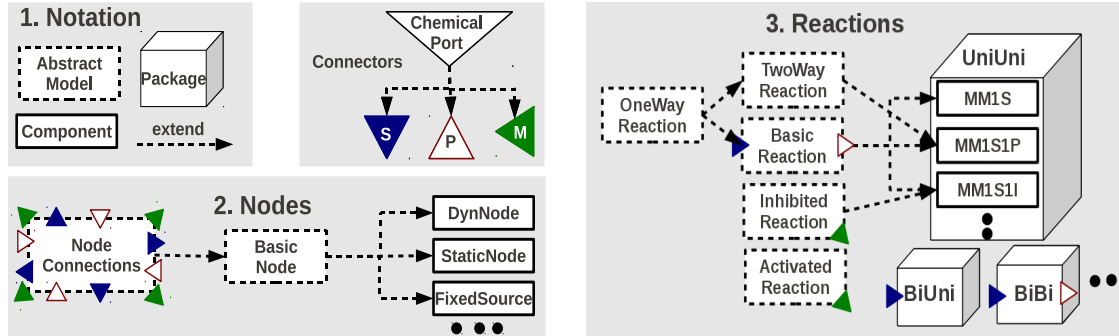


Figure 7.2.: General overview of the Biochem library

the base type of *BasicReaction* and further interfaces: *OneWayReaction* or *TwoWayReaction* for irreversible, reversible reactions respectively. Reactions affected by one inhibitor or one activator extend the interface *InhibitedReaction*, *ActivatedReaction* respectively. The implementation of a kinetic of an inhibited uni-uni reaction looks as follows:

Listing 7.1: a Uni-Uni inhibited irreversible kinetics

```

model MMS1I "Michaelis Menten irreversible and 1 inhibitor"
  extends BasicReaction    "UniUni";
  extends OneWayReaction  "Irreversible";
  extends InhibitedReaction "1 Inhibitor";
  parameter Units.Affinity km = 1.0;
equation
  v = kf*S1.c/(km_S1 + S1.c) * k_I1/(I1.c + k_I1);
  S1.v = -v;
  P1.v = v;
end MM1SI;
  
```

An implementation of the Spirallus network may look as follows:

Listing 7.2: Implementation of network of fig.

```

model Spirallus
  import Biochem.*;

  // Metabolites
  NodeElements.FixedSource Aex(c_0=10.0);
  NodeElements.DynNode B,C,D,E,F;
  NodeElements.ProductNode Eex,Fex;
  NodeElements.CofactorNode F;
  
```

```
// Reactions
Reactions.UniUni.MMS1I vupt(kf=10.0,km_S1=0.05,k_I1=.4);
Reactions.UniUni.MMSP v1(kf=1.5,kb=1.2,km_P1=.3),
    v5(kf=0.6,kb=0.4,km_S1=0.2);
Reactions.BiUni.MMS v2(kf=0.6,km_S1=1.0);
Reactions.UniBi.MMS1I v3(kf=0.3,kb=0.4,km_S1=0.4),
    v4(kf=0.3,kb=0.4,km_S1=0.4,k_I1=0.2);
Reaction.UniUni.DegS v6(kf=0.4),v7(kf=0.3);

equation

connect(Aex.S1,vupt.S1); connect(vupt.P1,A.P1); connect(A.M1,vupt.M1);
connect(A.S1,v1.S1); connect(v1.P1,B.P1);
connect(B.S1,v2.S1); connect(E.S1,v2.S2); connect(v2.P1,C.P1);
connect(C.S1,v3.S1); connect(v3.P1,D.P1); connect(v3.P2,F.P1); connect(D.M1,v3.M1);
connect(D.S1,v4.S1); connect(v4.P1,F.P2); connect(v4.P2,E.P1); connect(C.M1,v4.M1);
connect(B.S2,v5.S1); connect(v5.P1,E.P2);
connect(E.S2,v6.S1); connect(v6.P1,Eex.P1);
connect(F.S1,v7.S1); connect(v7.P1,Fex.P1);

end Spirallus;
```

7.3. Advantages of Modelica versus SBML

In (Wiechert et al. 2010) Modelica was compared with SBML for Systems Biology applications from different contexts. SBML from the one side, being the standard language for biochemical network modeling, is maintained and supported by the Systems Biology community, is distinguished by a large set of tools for highly specialized tasks. Additionally, a lot of network models are already stored in the SBML format in public databases. From the other side, it was shown that Modelica is superior when hierarchical modeling, multidisciplinary modeling, object-oriented reusable components and support of different modeling flavors are desired. Appendix A demonstrates more examples, where Modelica as a rich specification language shows further advantages from expressibility and mathematical perspectives. In particular the following aspects are addressed:

- The impact of declaration on the mathematical structure
- Whether a correct syntax corresponds to well-defined mathematical formulation
- The completeness of equations
- The ability of describing topological relations among compartments
- Freedom of equations formulation
- Shortcomings in some SBML constructs

It is shown that while a syntactically valid SBML document can be semantically or mathematically a wrong model, Modelica as a real programming language inherits natural mechanisms with which some semantical errors are forbidden at the early phase of modeling. Moreover and according to the followed design approach of a Modelica library, the common structural information within kinetic formulas can be utilized for implementing extendible interfaces (cf. section 7.2) or templates for generalized kinetics (Elsheikh et al. 2012) (cf. chapter 8). These templates can be then specialized according to the number of reactants, products, specific effectors, reversibility etc. In contrary, the declaration of reaction types within a SBML document has no impact on the mathematical formulation of the corresponding kinetics.

Chapter 8.

The ADGenKinetics library

This section demonstrates the *ADGenKinetics* library which implementation is motivated by the shortage of relying on mechanistic kinetics for applications of automatic model generations demonstrated in the following chapters. The library is one of the very few published libraries in the domain of Systems Biology. It is open-source, available in the standard Modelica website¹ and it is provided under the Modelica license version 2². The power of Modelica constructs is utilized for providing a compact implementation of simplified kinetic formats with generalized structured formulas. This gives the opportunity of realizing biochemical reaction networks using few number of reaction components, in contrast to libraries based on classical mechanistic kinetics requiring hundreds of reaction components.

8.1. Overview

ADGenKinetics (Elsheikh 2012) provides a compact implementation of simplified kinetics formats (cf. subsection 5.2.2). The employment of generalized kinetics has two advantages from two perspectives:

1. From the modeling perspective: Utilization of generalized kinetics formulas provides the opportunity of implementing a compact library with so few numbers of components that the user neither needs to choose an enzyme kinetic component from a long list of components nor he needs to self implement newer enzyme kinetics for newer cases of non considered reactions
2. From the implementation perspective: By efficient employment of powerful Modelica language constructs, the implementation of highly specialized practical library for modeling biochemical network applications gets simplified

ADGenKinetics follows in many aspects the main guidelines provided by the *Biochem* library (cf. section 7.2). The key aspects distinguishing *ADGenKinetics* appear whenever the mathematical structures of simplified kinetics are utilized for implementing interfaces for the underlying generalized formulas. These interfaces are specialized according to the

¹<https://www.modelica.org/libraries>

²<https://www.modelica.org/licenses/ModelicaLicense2>

number of reactants, products, specific effectors as well as reactions (ir)reversibility. Additionally, it is worth to mention that *ADGenKinetics* is the first algorithmically differentiated Modelica library. Additional differentiated components are used for the computation of parameter sensitivities.

8.2. The structure of ADGenKinetics

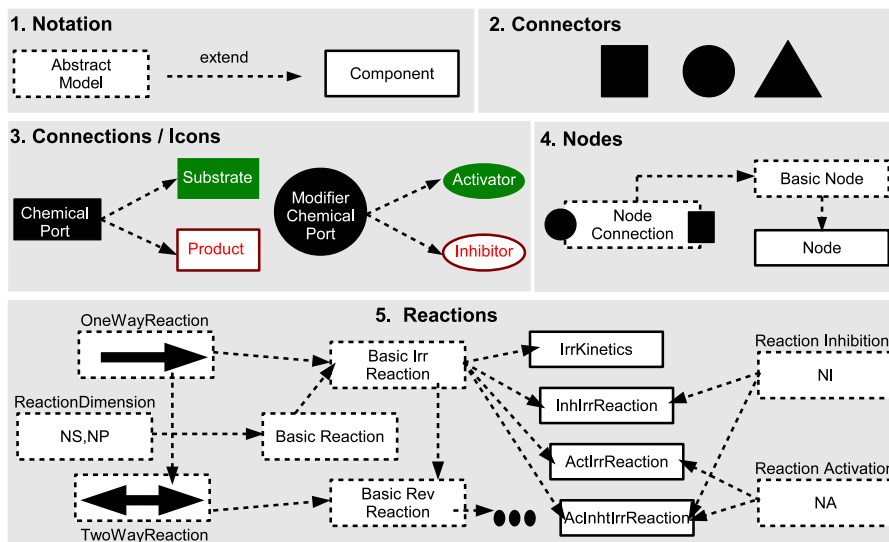


Figure 8.1.: General overview of the *ADGenKinetics* library

Figure 8.1 summarizes the presented library. The following packages are available:

1. *Interfaces*: connectors and classification interfaces independent from specific kinetic formats
2. *NodeElements*: components for nodes
3. *Reactions*: generalized non-specific components for reactions. Specific implementations are located in specialized subpackages like *convenience*, *linlog*, etc.
4. *Derivatives*: extended components for computing parameter sensitivities
5. *Examples*: biochemical network models as usage examples

Many classes within the *Biochem* library have been taken for the implementation of generalized kinetics, for instance the library structure, physical units, naming conventions and others. However, further components of *ADGenKinetics* are specially designed for providing compact implementation of generalized kinetic formats. Components for nodes

are declared with only one connector *ChemicalPort* to arbitrary number of reactions and one connector to arbitrary number of effectors (Wiechert et al. 2010). A node acts as a reactant, i.e. substrate or product, to the connected reactions via *ChemicalPort*, while it acts as effector to the connected reactions via *ModifierChemicalPort*. This is a bit more flexible than *Biochem* which implements a node with a static fixed number of connections to reactions. The next section is devoted for the illustrating the implementation of *Reactions*, the significant part of *ADGenKinetics*.

8.3. Implementation of reactions

Typical kinetics are implemented by extending several generalized abstract classes which specifies a reaction according to:

1. its dimension: how many substrates and products are involved as well as the stoichiometry of the reactants
2. its reversibility
3. whether the reaction is effected by other modifiers, how many and their types

The implementation of some of these basic classes are shown as follows:

Listing 8.1: The dimension of a reaction

```
class ReactionDimension "Dimension and structure of a reaction"
  parameter Integer NS = 1 "Number of substrates";
  parameter Units.StoichiometricCoef n_S[NS]=ones(NS) "Stoichiometry of all substrates";
  parameter Integer NP = 1 "Number of products";
  parameter Units.StoichiometricCoef n_P[NP]=ones(NP) "Stoichiometry of all products";
end ReactionDimension;
```

Using the previous class, an abstract type for reactions slightly modified version from the one provided in *Biochem* is given as follows:

Listing 8.2: The dimension of a reaction

```
partial model BasicReaction "basic declaration of a reaction"
  extends Interfaces.dynamic.Dimension.ReactionDimension;
  Units.VolumetricReactionRate v "reaction rate";
  Interfaces.ChemicalPort_S rc_S[NS] "connection to substrates";
  Interfaces.ChemicalPort_P rc_P[NP] "connection to product";
equation
  rc_S[:, r] = n_S[:, r] * v;
  rc_P[:, r] = -n_P[:, r] * v;
end BasicReaction;
```

Specification of the reaction reversibility is done via the related classes *OneWayReaction* and *TwoWayReaction*. These classes provide the basic declaration of related kinetic parameters and they are directly taken from *Biochem*. Moreover, two additional abstract classes *BasicIrrReaction* and *BasicRevReaction* are introduced in the proposed library for emphasizing type abstractions among implemented kinetics, for instance:

Listing 8.3: Basic reversible reaction

```

partial model BasicRevReaction "basic implementation of a reversible reaction"
  extends Reactions.convenience.dynamic.BasicIrrReaction;
  extends Interfaces.Reversible.TwoWay;
  Real P1 "Product terms nominator";
  Real P2 "Product terms denominator";
  parameter Units.AffinityConst KmP[NS] = ones(NS) "Affinity constants";
equation
  P1 = Vbwdmax * product({rc_P[i].c/KmP[i] for i in 1:NP});
  P2 = product({rc_P[i].c/KmP[i] + 1 for i in 1:NP});
end BasicRevReaction;

```

The corresponding classes for specifying the effectors are given by the classes *ReactionInhibition* and *ReactionActivation*, for instance:

Listing 8.4: The inhibitors of a reaction

```

partial model ReactionInhibition "Inhibition influencing a reaction"
  parameter Integer NI = 1 "# Metabolites inhibiting the reaction";
  Interfaces.ModifierChemicalPort_I mc_I[NI];
  parameter Units.AffinityConst KI[NI] = ones(NI) "Inhibition parameters";
  Real I "Inhibition term";
equation
  I = product({KI[i] / (KI[i] + mc_I[i].c) for i in 1:NI});
end ReactionInhibition;

```

Using these classes, all reaction types of convenience kinetics are realized only with eight classes. For instance, the implementation of convenience kinetics for reversible inhibited reactions with arbitrary numbers of reactant substrates, products and inhibitors is given as follows:

Listing 8.5: Kinetic for reversible inhibited reaction

```

class InhRevKinetic "S1+S2+... <==I1,I2,..==> P1,P2,..."
  extends Reactions.convenience.dynamic.BasicRevReaction;
  extends Reactions.convenience.dynamic.ReactionInhibition;
equation
  v = I * (S1 - P1) / (S2 + P2 - 1);
end InhRevKinetic;

```

8.4. Examples

The implementation of the Spirallus network (cf. section 6.1) is assembled as follows:

Listing 8.6: Implementation of the Spirallus network with ADGenKinetics

```

model Spirallusdyn "An abstraction of the TCA cycle"
  import ADGenKinetics.NodeElements.dynamic.*;
  import ADGenKinetics.Reactions.convenience.dynamic.*;
  Node Aex(c_0=1);
  InhIrrKinetic vupt(NS=1,NP=1,NI=1,Vfwdmax=1.0,KmS={0.1},KI={3.0});

```

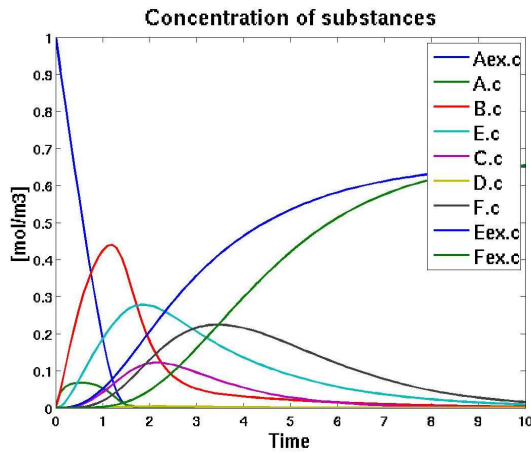



Figure 8.2.: Concentration of substances

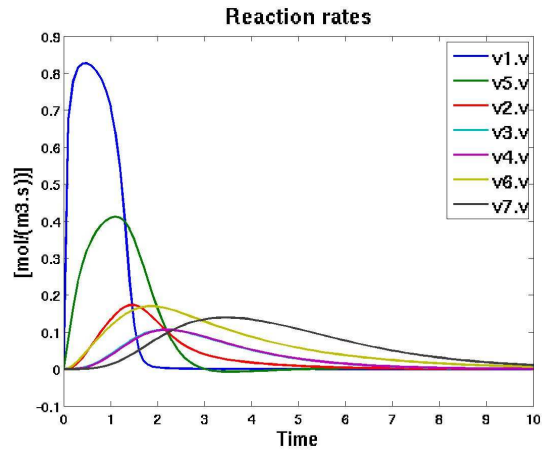


Figure 8.3.: Reaction rates

```

ModifierNode A;
RevKinetic v1(NS=1,NP=1,Vfwdmax=3.0,Vbwdmax=1.0,KmS={0.1},KmP={3.0});
Node B;
...
equation
// vupt
connect(Aex.rc,vupt.rc_S[1]);
connect(vupt.rc_P[1],A.rc);
connect(vupt.mc_I[1],A.mc);
// v1
connect(A.rc,v1.rc_S[1]);
connect(v1.rc_P[1],B.rc);
...
end Spirallusdyn;

```

Figures 8.2 and 8.3 demonstrate the simulation results of the concentration of chemical substances and the reaction rates of the Spirallus network, respectively.

Simulating parameter sensitivities

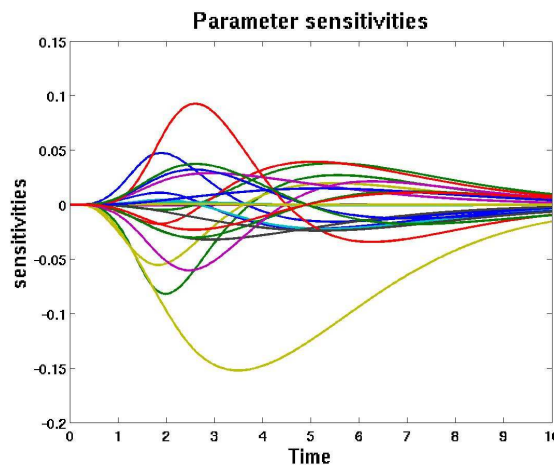
Using the subpackage *Derivatives*, parameter sensitivities can be computed in a straight forward way. For the Spirallus example, this can be done by slightly modifying the declaration part of the code from listing 8.6 to the following:

Listing 8.7: Implementation of the dynamics of the Spirallus network together with the parameter sensitivities

```

model ad_Spirallusdyn
import ADGenKinetics.Derivatives.NodeElements.dynamic.*;
import ADGenKinetics.Derivatives.Reactions.convenience.dynamic.*;

```

Figure 8.4.: Parameter sensitivities of v_7

```

import ADGenKinetics.Derivatives.Functions.*;
inner parameter Integer NG = 24 "Number of gradients";
Node Aex(c_0=1);
InhRevKinetic vupt(NS=1,NP=1,
    Vfwdmax=1.0,g_Vfwdmax=unitVector(1,NG),
    KmS={0.1},g_KmS={unitVector(2,NG)},
    KI={3.0},g_KI={unitVector(3,NG)});
...
IrrKinetic v7(NS=1,NP=1,
    Vfwdmax=2.0,g_Vfwdmax=unitVector(23,NG),
    KmS={3.0},g_KmS={unitVector(24,NG)});
Node Fex;

equation
    // equations remain as before
    ...
end ad_Spirallusdyn;

```

In the last model, the standard types for nodes and reactions are replaced by the extended types within the subpackage *Derivatives*. An additional unique parameter NG is declared, specifying the number of active parameters w.r.t. which derivatives are sought. Finally, the input gradient of any parameter p is initialized with the help of the function $unitVector(i,NG)$ which returns a unit vector of length NG with the i th component equal to one. In this way, for any variable v , $g_v[i]$ corresponds to $\partial v / \partial p$. For parameters with non-initialized gradients, they simply become inactive. Figure 8.4 shows the parameter sensitivities of the reaction v_7 w.r.t. all kinetic parameters. More comprehensive details behind the employed methodologies are given in (Elsheikh 201xb) and appendix C.2.

Chapter 9.

The Omix-Modelica plugin

This chapter presents a Modelica-based simulator tool for Omix, a highly-specialized biochemical network modeling editor. Omix (Droste et al. 2009) is a general-purpose editor for constructing, editing and visualizing biochemical networks in a semi-automatic manner (i.e. with the help of manual positioning from the user). All figures of biochemical network models in this work are drawn by Omix. Additionally, Omix is a developer-oriented tool as it provides guidelines for systematic implementation of additional sophisticated features through standardized plugins. An example is shown in figure 9.1 presenting an Omix snap-shot of a dynamic simulation of the labeled *Spirallus* network (Tillack et al. 2009). Aiming at the implementation of a Modelica-based simulator, this section describes the basic steps done from Modelica perspectives for overcoming serious difficulties and providing a simplified implementation. The Omix-Modelica plugin is a result of a cooperative work¹, aiming at converting manually edited biochemical networks by the Omix tool to valid Modelica models based on the Biochem library and its derivations. The exact contribution of this work is summarized at the end of this section.

9.1. Main difficulties of converting an Omix network to Modelica

Omix is able to visualize biochemical networks based on SBML format. This is a rather straightforward task, since the required information is present in the SBML model, like the present metabolites, the participant reactions, which metabolites are involved in which reactions, present effectors and many others. Consequently, with the help of manual positioning of metabolites and reactions, SBML network models can be drawn with Omix. In contrary, transforming an Omix network model to a Biochem-based model is much harder. The reason is that there is no one-to-one direct association of the given Omix network to the components of the Biochem library and vice versa. For example, given the implementation in listing 7.1 implementing the kinetics of an inhibited uni-uni reaction, required information cannot be directly deduced at syntax level such as:

- the number of substrates, products and effectors of a reaction
- the type of a metabolite (e.g. whether it is a cofactor)
- the presence and the type of the involved effectors

¹together with Peter Droste, research centre Jülich, Germany

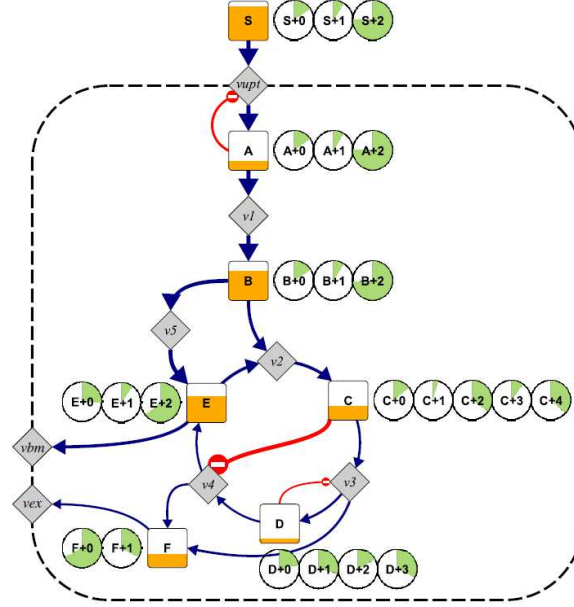


Figure 9.1.: Animation of a dynamic simulation of the labeled Spirallus network. Animation is done by scaled simulation data of intermediate concentration and reaction rates by filled boxes and varying arrow widths, respectively

In another word, direct automatic generation of Modelica models is not possible and hence assigning kinetics to reactions should be done manually. In this case, for each reaction, a corresponding enzyme kinetic should be manually assigned from hundreds of available kinetics, most of which are structurally not the right one (e.g. different number of substrates or effectors, etc.)

9.2. Design of the Omix-Modelica plugin

The basic steps performed by the Omix-Modelica plugin as illustrated in figure 9.3 are summarized as follows:

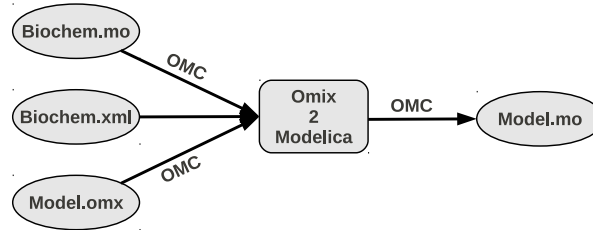


Figure 9.2.: General design of Omix2Modelica plugin

9.3. The problem of enormous numbers of kinetics

1. OMC is used to parse the basic components within the Biochem library
2. The parsed components are classified into types (metabolites and kinetics) with the help of an XML specification file
3. The user assigns the types of all metabolites and the values of kinetic parameters of an Omix network from a dialog menu. Only the types specified within the XML-file appears in the dialog menu
4. OMC is used to automatically generate the Modelica model (i.e. declaration of components and connection equations are automatically inserted by OMC rather than following an ad-hoc fashion)

The specification file explicitly declares which subset of types needs to appear at all in the dialog menu when the user assigns the types of metabolites and enzyme kinetics. In this way, the implementation is not restricted to a fixed library design. Moreover, different subsets of kinetics can be maintained instead of presenting all possible available kinetics in dialog menus.

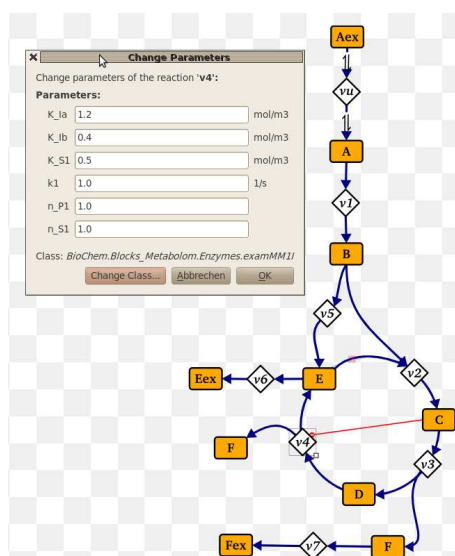


Figure 9.3.: Assigning parameter values to reaction v_4 of the Spirallus network

9.3. The problem of enormous numbers of kinetics

A main difficulty of the current implementation is that an enormous number of kinetics appearing in the dialog menu can be assigned to a reaction. Whenever, a large-scale network is modeled, further kinetics, which can be not yet available, need to be additionally implemented. This has many consequences:

1. The set of available reaction kinetics candidates get enlarged
2. The XML specification file needs to be extended by the user
3. Due to the potentially large number of types for enzyme kinetics, the user needs to identify the right kinetic from hundred of candidates
4. The user may assign a structurally wrong kinetic assignment to a reaction of different dimension

This is an obstacle for a smooth flexible modeling where several kinetics pro each reaction need to be examined, simulated and validated.

9.4. Enriching the Biochem library with annotations

These problems can be solved by enriching the Biochem library components with additional annotations. The Modelica annotations are kind of structural tools-oriented documentation used by many standard software and Modelica-oriented development editors. Such annotations can explicitly declare the missing information to announce the Omix-Modelica plugin which class of reactions it can get assigned to. Annotation elements can be also used for

- differentiating between metabolites and cofactors which require special handling
- declaring the number of inhibitors and activators

An example of enhanced annotations is given in the model *BasicReaction* from listing 8.2 as follows:

Listing 9.1: Annotation for number of reactants

```
partial model BasicReaction
  annotation(ReactionKinetic(nsubstrates=1,nproducts=1));
  ...
end BasicReaction;
```

announcing that any model extending this abstract type has one substrate and one product. Nevertheless for reaction kinetics declaring further substrates or products, the annotation should then be overwritten:

Listing 9.2: Overwriting Annotation

```
partial model BiUniReaction
  extends BasicReaction
  annotation(ReactionKinetic(nsubstrates=2));
  Interfaces .ChemicalPort_S S2;
  ...
end BasicReaction;
```

9.5. The contribution made to the Omix-Modelica plugin through this work

If the number of substrates or products are parameterized, i.e. specified by a parameter declaring array variable of non fixed size, the same annotation should be used with that parameter.

Whenever these interfaces are extended by specialized kinetics, these annotations are also inherited within these kinetics which explicitly declare the dimension of the underlying reaction. By a proper implementation extension of Omix-Modelica plugin, it can be guaranteed that the user cannot assign a structurally wrong kinetic for a corresponding reaction. In this case, fewer relevant kinetics would appear in a dialog menu instead of hundreds of kinetics. Moreover, whenever extra kinetics are implemented by extending annotated interfaces, no or little reconfiguration procedures need to be excessively repeated. In particular, there is no need to extend the XML specification file. Another way is to overcome the enormous number of present kinetics is to use approximative kinetics instead if relevant cf. chapter 8.

In summary, exploiting Modelica constructs makes the transformation of an Omix network model to Modelica model possible in the same way an SBML document can be transformed to an Omix network.

9.5. The contribution made to the Omix-Modelica plugin through this work

The activities performed within this work have been substantially contributed to the design and the implementation of the Omix-Modelica plugin in the following aspects:

1. The employment of OMC for model parsing and automatic model generation has tremendously simplified the implementation of the Omix-Modelica plugin. This has contributed also to the modularity of the back-end library, on which generated models are based. Without OMC, the resulting plugin would rather rely on an adhoc static implementation possibly fixed towards only one library
2. The first versions of this product has been also tested and experimented for providing further feedback and improvement potentials
3. Further recommendations and suggestions for expected future problems regarding enormous numbers of reaction kinetics are given in this thesis
4. The first Modelica library aiming at providing compact implementation of simplified kinetic formats adequate for applications of automatic model generation is implemented, cf. chapter 8. This library does not only serve as a candidate for specific enzyme kinetics but it is also suitable as a base library for modeling arbitrary large-scale networks with very few number of reaction components

Chapter 10.

Implementation of Model Families in Modelica

This chapter presents a novel way for implementing model families in Modelica, an important concept for model validation. Section 10.1 introduces the concept of model family in the context of biochemical network modeling. Section 10.2 gives a quick overview of specification of model family in Modelica. Finally, the common concepts between model family and the Omix-Modelica plugin are emphasized. Additionally, the Modelica approach is compared with an SBML-based approach for model family specification.

10.1. The concept of model family

Generally a mathematical model could be considered to be valid if it is able to

1. describe the system it is modeling
2. predict the real system behavior w.r.t. different inputs

For network models corresponding to cellular processes, it is quite impossible to assert that any model satisfies the above definition (Wiechert and Takors 2004). In particular, many enzyme-related information like amount, interactions with other metabolites and influence of cellular environmental conditions on enzyme binding mechanisms are partially or totally unknown within the specific system under study. In order to resolve this situation, a lot of simplification assumptions need to be considered by model construction. Namely, for each reaction there could be some assumptions regarding:

- the set of influentially significant effectors
- the specific enzyme binding mechanisms

that need to be considered by the model. Each combination of assumptions results in a distinguished model w.r.t. the resulting kinetic terms. The set of all assumption combinations compose a space of models referred to as *model family* (Haunschild 2006). Figure 10.1 presents two abstract network models that have been used in literatures for demonstrating applications of model families. The network model in the left of the figure has been used to examine the accurateness of the linlog kinetics against mechanistic Michaelis-Menten

kinetics (Visser and Heijnen 2003). This is done by generating in-silico simulation data out of a reference model represented by mechanistic kinetics and incrementally replacing mechanistic kinetics by linlog kinetics one reaction after another. After each replacement parameter estimation is done to reproduce the original data.

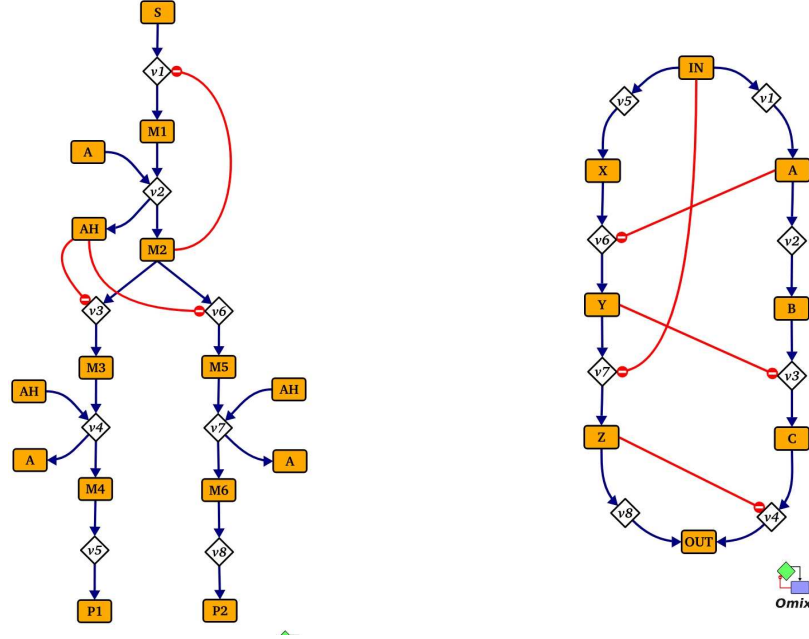


Figure 10.1.: Network models for examining kinetic variants

The other network model has been used to examine how far simulations of different networks with different enzyme inhibition mechanisms can reproduce each other (Haunschild et al. 2006). Initially, one of the models has been considered as a reference model for generating in silico simulation data. The model space consisted of 2880 models, only some of which were explored and tuned with parameter estimation due to the computational complexity behind them. Interestingly, couple of models reproduced similar behavior. This shows that reproduction of data does not imply the correctness of model. Nevertheless, qualified models that have been subject to intensive validation through generation of experiments can be assumed to inherit part of the truth in a way, that model-based decisions can be collectively made in an integrative manner. The following is a formal definition of model family developed within this work:

Definition 10.1 (Model variants of a metabolic network M). Suppose that M is a metabolic network with n reactions r_1, r_2, \dots, r_n . Assume that

$$V_i = \{v_i^1(\alpha_i^1), v_i^2(\alpha_i^2), \dots, v_i^{n_i}(\alpha_i^{n_i})\}$$

is the set of all admissible kinetic variants for a reaction r_i according to some given assumptions. Assume also that M_{i_1, i_2, \dots, i_n} represents the mathematical description of M using the kinetics $v_1^{i_1}(\alpha_1^{i_1})$, $v_2^{i_2}(\alpha_2^{i_2})$ and $v_n^{i_n}(\alpha_n^{i_n})$ with $1 \leq i_j \leq n_j \forall j \in \{1, 2, \dots, n\}$. Then the model space S is defined as:

$$S = \left\{ M_{i_1, i_2, \dots, i_n} : v_j^{i_j}(\alpha_j^{i_j}) \in V_j, \forall j \in \{1, 2, \dots, n\} \right\}$$

with $|S| = \prod_{i=1}^n n_i$. Two models $M_{r_1, r_2, \dots, r_n}, M_{s_1, s_2, \dots, s_n} \in S$ are adjacent if $r_i = s_i \forall i \in \{1, 2, \dots, n\} / \{j\}$ except exactly one $i \neq j$. In this case, the first model can get transformed to the second model by exchanging the kinetic $v_j^{r_j}$ with $v_j^{s_j}$.

As it will be shown in the next section, the hidden mapping behind the definition of adjacent models is the base of representing model families in Modelica.

Example 10.2. Assuming that there are about 3 kinetics law variants for each reaction in the first network model. Given that v_i^1, v_i^2 and v_i^3 represent the linlog, convenience and one variant of mechanistic kinetics respectively. Then, the model $M_{12111111}$ stands for a model described by the linlog kinetics except the second reaction with the convenience kinetics. Two models are adjacent in the model space if they differ only in one digit.

In order to choose valid model candidates describing a system, they must be validated with experimental data. In practice, the space of models is rather finite but it is composed of combinatorially high number of models not all of which can be validated due to the huge computational complexity. Model selection strategies attempt to rather explore only a small subspace of models for identifying a subset of models satisfying:

$$\min_i \left\{ Crit \left[\min_{\alpha_j} \left[P \left(M^*, M^i(\alpha_j^i) \right) \right] \right] \right\} \quad (10.3)$$

where P corresponds to parameter estimation function using a reference model M^* with the help of some selection criteria like the Akaike criteria (Akaike 1980).

10.2. Specification of model families in Modelica

The basic idea behind the presented approach is to provide a compact representation of a model family through one single model by making use of template programming. In Modelica this is realized by making use of:

- replaceable components for declaring unfixed reaction mechanisms and

- annotations for specifying the candidate choices

The resulting model specification declares a well-defined model space. As it will be shown, element models within this space can be expressed in terms of each others in the sense of definition 10.1. In order to simplify the clarification of this concept, approximative kinetics are used instead of mechanistic ones. Nevertheless, the same approach is also applicable for mechanistic kinetics. The implementation relies on additional sub-packages for the Biochem library, providing more components with which hierarchies of descriptive elements are present at declaration level. In the followings, the basic idea is presented.

10.2.1. Kinetic law variants

The realization of kinetic law variants with the left network of figure 10.1 is done as follows:

Listing 10.1: Replaceable Kinetics

```
model KineticVariants
  import ModBioChem.*;
  replaceable class RKinetics = Reactions.LinLogReaction
    extends Interfaces.Kinetics.KineticLaw
    annotation(choices(choice=Reactions.LinLogReaction,
                     choice=Reactions.ConvenienceReaction));
  // Metabolite Declaration
  NodeElements.Node AH(c_0=1.0,N=3);
  NodeElements.Node M1(c_0=3.0,N=2);
  ...
  // Reaction Declaration
  replaceable RKinetics R2(NS=2,NP=2,kf=1.0,E_S={1.0,1.0},E_P={1.0,1.0});
  ...
equation
  ...
end KineticVariants;
```

This model describes the dynamics of the chosen metabolic network using the linlog kinetics. In this model kinetic laws are declared as replaceable types and annotations are used for specifying other alternative kinetics. In this way, the same network modeled by convenience kinetics can be described as follows:

Listing 10.2: Convenience kinetics variant

```
model ConvenienceVariant = KineticVariants(
  redeclare replaceable class RKinetics = Reactions.ConvenienceReaction
  extends Interfaces.Kinetics.KineticsLaw);
```

The parameters (e.g. specialized kinetic parameters) which are not explicitly present at the declaration take the default values. The optimal parameter values need to be determined by parameter estimation. For large models, it makes sense to perform the replacement of kinetics in an incremental manner, i.e. replace a reaction one after another with incremental parameter estimation. This is done as follows:

10.3. The advantages of the Modelica approach

Listing 10.3: Incremental replacement

```
model M_12111111 =  
  KineticVariants(redeclare replaceable  
    Reactions.ConvenienceKinetics.Types.BiBi R2(km_1=0.5,km_2=0.5));
```

The class *BiBi* is a static type definition from the *ConvenienceKinetics* class with fixed number of substrate and products. Note that in that specific class parameters (e.g. k_{m1}) can now be present at declaration level with suggested start values for the optimization algorithm. In the same way, further models can be generated from $M_{12111111}$.

10.2.2. Enzymatic variants

For representing enzymatic variants, almost the same approach is adopted. The only issue which is different is that the number of effectors influencing a certain reaction may change from a model to another within a model family. For this reason, the implementation requires reaction effectors to be present at declaration level. The declaration of the reaction v_4 in the right network model of figure 10.1 looks as follows:

Listing 10.4: Inhibited reaction declaration

```
// Reaction Declaration  
replaceable RKinetics R4(redeclare class RModifier =  
  Interfaces.Effectors.InhibitedReaction(NI=1), NS=1,NP=1,inhibitors = {Z.mc},  
  E_I={1.0},kf=1.0,E_S={1.0},E_P={1.0});
```

Given a model with the reaction v_4 inhibited by node Z , another model with an additional inhibition node Y can be simply described as follows:

Listing 10.5: Enzymatic variants

```
model EnzymaticVariants_00121010 =  
  EnzymaticVariants(replaceable redeclare RKinetics  
    R4(NI=2,inhibitors = {Z.mc,Y.mc},E_I={1.0,-0.2} );
```

In a similar manner, any model in the underlying model space can be explicitly generated in terms of a reference model.

10.3. The advantages of the Modelica approach

10.3.1. Common issues with the Omix-Modelica plugin

Given a Modelica model in which a model family is internally specified, an automatic model generator would be needed to realize model selection strategies. An automatic model generator needs to perform the following steps:

1. parse the model specification
2. parse the underlying library

3. generate the required model

These steps are partially performed with the Omix-Modelica plugin. It is not a surprise to say that the implementation approach of this tool was actually the same approach planned for realizing model families in Modelica. Both concepts benefit from:

- generalized kinetics with which automatic model generation is easily done
- OMC with which model parsing and generation is easily done
- annotation elements with which missing information is declared

Once the implementation of the Omix-Modelica plugin achieves a mature state, it is recommended to extend it as an editor for model family specification in which

- the user can specify kinetic candidates for each reaction and let
- the specification of model family be automatically generated

10.3.2. Model family: Modelica versus SBML

In (Haunschild 2006), SBML was extended with additional elements in order to be able to specify model families. Consequently, additional efforts have been taken for implementing additional parser for the extended SBML documents. Out of such specification, it was possible to generate equation-based models subject to simulation and parameter estimation for the validation process according to a model selection strategy. From the other side, it is shown that a compact specification of model families can be completely specified using only one single Modelica model without the need of additional language elements. The specification has the advantage that

1. the resulting model spaces can be easily explored in the sense of definition 10.1
2. it could be parsed by existing tools, OMC and the Omix-Modelica plugin

The powerful expressibility of Modelica constructs makes it possible to generate descriptive models in terms of each other. This is favorable for incremental model selection strategies (i.e. models expressed in terms of one additional modification of another model). Furthermore, the tools used within the Omix-Modelica plugin can be also employed for parsing model family specification and systematic model generation.

Thus, this shows that full utilization of Modelica constructs makes Modelica models enough descriptive as SBML for supporting Omix models. Additionally, the implementation of model families is totally realizable in Modelica without additional language elements. Thus, existing model parsing and model generation tools can fully support such sophisticated tasks keeping individual efforts at an acceptable minimal level.

Part IV.

Sensitivity Analysis of Modelica Models

Chapter 11.

Parameter Sensitivities of DAE Systems

The last part demonstrates some modeling applications in Systems Biology based on DAE systems. The discussion of the last chapter together with the applications shown in part V show that it is not only model simulations but also the whole package of sensitivity analysis, model validation and discrimination, optimization and others which are most useful to gain knowledge. A significant computational part within all these applications is based on the computationally expensive task of sensitivity analysis of DAE systems for computing parameter sensitivities. This part introduces new tools and equation-based concepts developed throughout this work aiming at efficient automatic sensitivity analysis of Modelica models. In this chapter, analytical methods for computing parameter sensitivities of DAE systems through direct integration of sensitivity equation systems is introduced in section 11.1. Section 11.2 presents a simplified overview of classical *automatic differentiation* (AD) techniques with which sensitivity equations of explicit DAE systems can be efficiently derived as shown in section 11.3. The presented overview of classical AD techniques is demonstrated in a way that serves as a base for the new invented equation-based AD techniques, especially developed for DAE-based models.

11.1. Common methods

Generally, sensitivity analysis aims at analyzing the behavior of the corresponding model outputs w.r.t. model inputs. The validity of the resulting analysis is either local around the chosen input values but can have also global scope via global sensitivity analysis methods (Saltelli et al. 2004). The scope of applications in this work requires only local sensitivity analysis of DAE systems around input parameter values. Formally, consider the DAE system:

$$F(\dot{x}, x, p, t) = 0 \quad , \quad x(t_0) = x_0(p) \quad (11.1)$$

where $x(t) \in R^n$ and $p \in R^m$ represent state variables and model parameters, respectively. Required are the derivatives dx/dp (or a subset of them) which quantify, according to their mathematical definition, the impact of parameters p on variables x . One way for numerical computation of parameter sensitivity is to employ *finite difference* (FD) methods. From one side, these methods are straightforward to implement. From the other side, they cause a lot of numerical problems so that the computation of accurate results could be

not possible, as shown in chapter 15. A better alternative is to compute derivatives analytically. Namely, for DAE systems of the form (11.1) with differential index less than or equal to one, corresponding parameter sensitivities can be computed by directly integrating the *sensitivity equation system* consisting of the original DAE system (11.1) and the *sensitivity equation subsystems* obtained by differentiating all equations w.r.t. desired parameters:

$$F_{\dot{x}}\dot{s}_i + F_x s_i + F_{p_i} = 0 \quad , \quad s(t_0) = \frac{\partial x_0(p)}{\partial p_i} \quad (11.2)$$

$$\text{where} \quad s_i = \frac{\partial x}{\partial p_i} \quad \text{for} \quad i = 1, 2, \dots, m$$

The dimension of the whole system is equal to $n + mn$. All sensitivity equation subsystems are independent from each other. However the terms $F_{\dot{x}}$, F_x and F_p depend on x solved by the original DAE system (11.1). Hence, the sensitivity equation subsystems (11.2) cannot be decoupled from the original DAE system (11.1). Integration of the sensitivity equation systems could be inefficient and much slower than simple versions of FD methods. The reason is that the number of iterations required by the DAE solver for maintaining error tolerance may become excessively enlarged. Moreover, high-dimensional Jacobians need to be factorized for the solution process of the underlying Newton-like scheme (4.14). Alternatively, further advanced methods exist by which some of the above mentioned drawbacks are partially overcome. These methods, summarized in appendix B, provide more efficient implementation by exploiting the structure of sensitivity equation systems for problem decomposition, cheap factorization and parallelization.

11.2. Introduction to AD

In this section, a technique for evaluating analytical derivatives, termed in literatures as *automatic differentiation* or *algorithmic differentiation* (Griewank and Walther 2008, Naumann 2012), is introduced. AD is a methodology that refers to algorithmic techniques for semantic augmentation of numerical programs with additional code for computing derivatives. AD employs common compiler techniques to compute an efficient representation of analytical derivatives without the drawbacks present in symbolic differentiation (shown in the following) and FD methods (cf. chapter 15). AD is fundamentally different that common symbolic differentiation methods available from common *Computer Algebra* (CA) packages as illustrated throughout the following example.

Symbolic differentiation

Sensitivity equation subsystems can be explicitly computed by symbolic differentiation techniques. Nevertheless, within explicitly differentiated formulas, many common subexpressions are excessively evaluated as in the following example.

Example 11.3 (Evaluating sensitivity equations by symbolic differentiation). *A typical equation of an enzyme kinetic formula within a large biochemical network model would look similar to:*

$$\dot{S} = -v \quad , \quad v = V \frac{S}{S + K_m} \frac{K_I}{I + K_I} \quad (11.4)$$

The corresponding sensitivity subsystem w.r.t. any parameter p is:

$$\begin{aligned} \dot{S}_p &= -v_p \\ v_p &= V_p \frac{S}{S + K_m} \frac{K_I}{I + K_I} + V \frac{S_p(S + K_m) - S(S_p + (K_m)_p)}{(S + K_m)^2} \frac{K_I}{I + K_I} + \\ &\quad V \frac{S}{S + K_m} \frac{(K_I)_p(I + K_I) - K_I(I_p + (K_I)_p)}{(I + K_I)^2} \end{aligned} \quad (11.5)$$

The terms $S + K_m$, $1/(S + K_m)$ and $S/(S + K_m)$ are evaluated 6, 5 and 4 times at each single iteration within the DAE-solver. Additionally, the same terms appear in all sensitivity equations obtained by differentiating the kinetic equation (11.4) w.r.t. all parameters. Overall, for typical kinetic equations summarized in section 5.2, there is a large excessive number of common expressions that need to be evaluated when integrating symbolically differentiated equations for computing parameter sensitivities.

Common Concepts in AD

In this subsection, some basic terminologies of AD, used throughout this part are introduced. Formally, given a numerical program P that computes a function:

$$f : x \in R^m \rightarrow y \in R^n$$

with m independent variables as inputs and n dependent variables as outputs, AD computes a new code P' additionally evaluating partial derivatives. P' is either explicitly generated or it is the same copy of P in which standard mathematical operations are semantically overloaded with derivative rules. The former is referred to as *semantical source-to-source transformation* while the later approach is known as *operator overloading*. A summary of common software is given in (Naumann et al. 2004). AD comes in two flavours, the forward mode evaluating the *tangent linear model* and the backward mode evaluating the *adjoint model*. The latter approach is out of the scope of this work.

The tangent linear model evaluates a function of the form $f' = (\partial y / \partial x) \cdot x^{(1)}$ together with f , where:

Chapter 11. Parameter Sensitivities of DAE Systems

- $(\partial y / \partial x) \in R^{n \times m}$ corresponds to the Jacobian
- $x^{(1)} = (\partial x / \partial s) \in R^m$ is a seed vector specifying the desired directional derivatives
- s is an auxiliary variable chosen in a way making f' equal to the desired directional derivatives

The Jacobian $(\partial y / \partial x)$ can be accumulated by executing P' m times, each time with s initialized to a different unit vector $e_i \in R^n$ for $i = 1, 2, \dots, n$. The cost of executing the generated derivatives is in $O(m \cdot f(x))$ which is identical to the cost of FD methods. If the Jacobian is sparse, the sparsity pattern can be exploited for efficient computations by initializing $x^{(1)}$ in a way that different partial derivatives are computed simultaneously.

The variables y are usually not expressed as an explicit function of x , but in terms of *intermediate* or *temporary variables* u_* as follows:

$$y = f(x) = u_k(u_{k-1}(\dots(u_2(u_1(x)))\dots))$$

These temporary variables u_* represent intermediate computations corresponding to branches and loops. A *derivative object* represents derivatives information, such as a vector of partial derivatives $(\partial u / \partial v_1, \dots, \partial u / \partial v_n)^T$ of a variable u w.r.t. a vector $v = (v_1, v_2, \dots, v_n)^T$. Any program variable with which a derivative object is associated is called an *active variable*, otherwise it is termed as an *inactive variable*. AD exploits the chain rule for generating derivatives in the form of:

$$\frac{\partial y}{\partial x} \cdot x^{(1)} = \frac{\partial y}{\partial s} = \frac{\partial y}{\partial u_k} \frac{\partial u_k}{\partial u_{k-1}} \dots \frac{\partial u_2}{\partial u_1} \frac{\partial u_1}{\partial x} \frac{\partial x}{\partial s}$$

If the program P expresses a functional unit evaluating a DAE system (11.1) and parameter sensitivities are sought through integrating the corresponding sensitivity equation system (11.1) and (11.2), then the parameters p are the independent variables and x are the dependent variables. This should not be confusing since parameters can be also thought as variables whose values are constant within a single simulation but different from a simulation to another. As a conservative strategy, all dependent variables $x_i, i = 1, \dots, n$ are considered to be active, i.e. a corresponding derivative object dx_i/dp is always associated. If only a subset of parameters p are desired, only the corresponding subset of the sensitivity equations need to be considered by rewriting equation (11.2) to:

$$[F_x \dot{s} + F_x s + F_p] J_{in} = 0 \quad , \quad s(t_0) = \frac{dx_0(p)}{dp} \quad (11.6)$$

Where $J_{in} \in R^{m \times m}$ is the *input Jacobian* specifying the independent parameters w.r.t. which derivatives are computed. If all parameters p are independent and all parameter sensitivities are desired then $J_{in} = I_m$.

11.3. Algorithmic aspects of AD

The key concept behind AD is that any mathematical function expressed as a program, no matter how complex it is, can be decomposed into a limited set of elementary equations composed of basic operations (e.g. $+$, $-$, ..etc.) and intrinsic functions (e.g. \sin and \cos). The derivative of each of these elementary equations can be computed by applying the chain rule to combine their local partial derivatives. Consequently, the resulting common subexpressions are executed only once each iteration.

Example 11.7 (Deriving sensitivity equations with AD). *Given equation (11.4) and using the Abstract Syntax Tree (AST) representation shown in figure 11.1, the kinetic equation and the corresponding sensitivity equation w.r.t. a set of parameters p_1, p_2, \dots, p_m can be represented with only two temporary variables as:*

$$\begin{array}{llll}
 T_1 & := & S + K_m & \implies & T'_1 & := & S' + K'_m \\
 T_1 & := & 1/T_1 & \implies & T'_1 & := & -T'_1 T_1 T_1 \\
 T_1 & := & S T_1 & \implies & T'_1 & := & S' T_1 + S T'_1 \\
 T_2 & := & I + K_I & \implies & T'_2 & := & I' + K'_I \\
 T_2 & := & 1/T_2 & \implies & T'_2 & := & -T'_2 T_2 T_2 \\
 T_2 & := & K_I T_2 & \implies & T'_2 & := & K'_I T_2 + K_I T'_2 \\
 T_1 & := & T_1 T_2 & \implies & T'_2 & := & T'_1 T_2 + T_1 T'_2 \\
 v & := & V T_2 & \implies & v' & := & V' T_2 + V T'_2
 \end{array}$$

where $X' = (dX/dp_1, dX/dp_2, \dots, dX/dp_m)^T \quad \forall \text{ variables } X$

and $:=$ corresponds to an assignment relationship between an output and inputs.

The following observations concerning the previous example are emphasized:

1. The assignment $T'_1 := -T'_1 T_1 T_1$ is equivalent to the expression:

$$T'_1 = \frac{S' + K'}{(S + K)^2}$$

2. The expression $T_1 T_1$ can be stored in a temporary variable and need not to be performed m times
3. Further arithmetic operations can be eliminated by exploiting the sparsity of the input Jacobian J_{in} (e.g. K' and V' have maximally 1 nonzero entry)

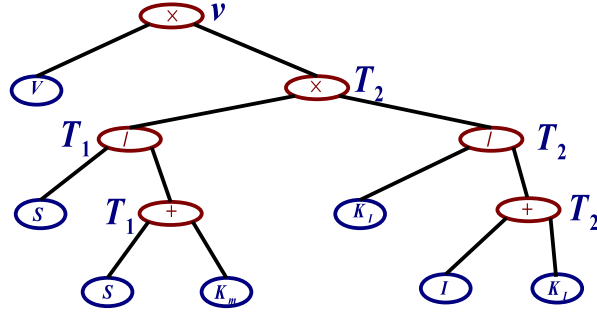


Figure 11.1.: AST of the right hand side of the kinetic equation from (11.4). Each non-leave node is represented by a reusable temporary variable.

The following table summarizes the number of *FLoating-point OPerations* (FLOPs) for one evaluation of equation (11.4) and its parameter sensitivities using all presented approaches for sensitivity analysis.

# Ops	+	−	×	/
Eq. (11.4)	2	1	2	2
CD equation (15.3)	$8m$	4	$8m$	$9m$
SD equation (11.5)	$14m$	$3m$	$10m$	$8m$
AD example (11.7)	$4+2m$	$1m$	$4 + 8m$	0

SD corresponds to symbolic differentiation. CD corresponds to a CD formula of order 4 illustrated in section 15.1. The table shows that a successful application of AD gives the most promising approach taking into account that division is computationally the most expensive arithmetic operation. This AD approach, though presented in a simplified way, is rather adequate for explicit DAE systems implemented in an assignment format within a procedural language like C/Fortran. This way is not applicable for equation-based languages like Modelica. In chapter 13, an alternative modified equation-based approach is presented.

Chapter 12.

Overview of ADModelica

This chapter introduces ADModelica an AD tool supporting Modelica models. The next chapter discusses the algorithmic aspects behind the developed equation-based approach. In section 12.1, the employment of AD techniques for Modelica models is motivated. The adopted approach is realized through the ADModelica tool presented in section 12.2. Finally, section 12.3 gives an example of an ADModelica generated model.

12.1. Why AD for Modelica?

Standard Modelica compilers already employ AD techniques for evaluating analytical Jacobians for the task of index reduction and numerical integration (Olsson et al. 2005, Braun and Bachmann 2011), cf. chapter 4. Similarly, in order to compute parameter sensitivities of a given DAE-based Modelica model, the discussion of the previous chapter reveals that the best approach is to integrate the corresponding sensitivity equation systems computed by AD techniques for the following reasons. Firstly, providing derivatives within a Modelica model makes the derived model independent from common Modelica compilers and accessible for various simulation environments. Secondly, the derivatives produced by AD techniques don't contain computationally expensive common sub-expressions. This is significant for the present model applications, cf. equation 5.9. Additionally, a Modelica model is usually given in a high-level abstraction rather than a pure mathematical formulation using library components and connections. These components could be implemented using loops, branches and other language constructs. Therefore, it makes sense to adopt Modelica high-level compiler techniques by AD for:

1. accessing the underlying DAE system of the given model
2. reducing the size and complexity of sensitivity equations, rather than blind differentiation of all equations as equation (11.2) suggests (cf. section 13.2)

Luckily, the implementation of a Modelica-based AD tool for Modelica models can be assisted using OMC in a similar way to the Omix-Modelica plugin (cf. section 9.2). With OMC (cf. section 2.1), it is possible to parse a high-level model and to access its underlying solvable optimized mathematical formulation. Consequently, there are different abstraction levels on which AD techniques can be applied (Elsheikh and Wiechert 2008). These levels are:

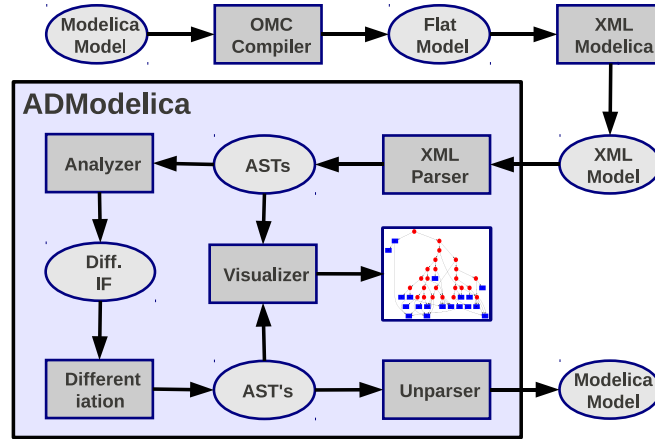


Figure 12.1.: ADModelica basic units

1. *The library level*: represented as high-level descriptive Modelica code. An AD tool relying on this level is given in (Bischof et al. 2005). Other works based on that approach are given in (Elsheikh 2012) and (Elsheikh 201xb)
2. *The flat level*: the underlying DAE system represented in pure equations form
3. *The low-level C code*: the generated simulation code in C. AD tools following this approach are given in (Imslund et al. 2009, Andersson et al. 2010)

Appendix C discusses all these approaches in more details. In this work the flat model approach is adopted due to the minimal efforts required for implementation.

12.2. ADModelica: An AD tool for Modelica models

ADModelica is an AD tool that accepts a Modelica model and dependent base libraries as inputs to compute the corresponding sensitivity equation system represented as a Modelica model (Elsheikh et al. 2008). The tool is implemented in the Java language with which communication with OMC through well-defined API is performed (Sjölund and Fritzson 2009). Figure 12.1 shows the main steps performed for computing parameter sensitivities. These steps are briefly summarized as follows:

- **Flattening**: A high-level model is flattened to pure mathematical equations
- **Transforming to intermediate format**: The ModelicaXML tool (Pop and Fritzson 2003) transforms the mathematical equation system to an easy-to-handle XML format, in which ASTs representation of the equation system is implicitly present
- **Analyzing**: The ASTs are transformed to equation-based intermediate format optimized for differentiation

- The implementation is decomposed into independent packages, each of which corresponds to a functional-unit of the source-code transformation process.

This section illustrates the structure of the generated code by ADModelica, illustrated upon the Spirallus network example from section 20.1. By operating ADModelica on an input model, two modes are available; the CA mode corresponding to symbolic differentiation and the AD mode. In the CA mode, the original model is extended with additional equations for derivatives obtained by symbolic differentiation as follows:

```
model ad_Spirallus
    extends Spirallus;
```

Listing 12.2: Input Jacobian and selection of active parameters

The array *PARINDEX* declares the indices of the active parameters within the input Jacobian. In this case, the generated code simulates parameter sensitivities only w.r.t. $v_{max,1}^{fwd}$, $v_{max,1}^{bud}$ and $K_{I,vupt}$. Usually the input Jacobian is equal to the identity matrix if parameters are independent. Directional derivatives or fixed derivatives can be alternatively considered by altering the above matrix. All parameters not appearing within the

Chapter 12. Overview of ADModelica

input Jacobian are inactive. The generated model declares derivative objects as arrays for each active variable as follows:

Listing 12.3: Derivatives declaration

```
//Declaration of derivatives
Real[NACTPAR] gp_v1_fwd;
Real[NACTPAR] gp_v1_bwd;
```

The equation part describes the sensitivity equation system in a gradient format. Within the CA mode, an equation may look as follows:

Listing 12.4: Sensitivity equation system in symbolic differentiation mode

```
equation
  for ad_i in 1:NACTPAR loop
    ...
    gp_v2_fwd[ad_i] = (((((B_xx_con/v2_km_B)*((E_xx_con/v2_km_E)*
gp_v2_fwd_vmax[PARINDEX[ad_i]])+(v2_fwd_vmax*(gp_E_xx_con[ad_i]/v2_km_E))))+
((v2_fwd_vmax*(E_xx_con/v2_km_E))*(gp_B_xx_con[ad_i]/v2_km_B)))*
((1*(1+(E_xx_con/v2_km_E)))*(1+(B_xx_con/v2_km_B))))-
((((1+(B_xx_con/v2_km_B))*(1*(gp_E_xx_con[ad_i]/v2_km_E)))+
((1*(1+(E_xx_con/v2_km_E))*(gp_B_xx_con[ad_i]/v2_km_B)))*
((v2_fwd_vmax*(E_xx_con/v2_km_E))*(B_xx_con/v2_km_B))))/
(((1*(1+(E_xx_con/v2_km_E)))*(1+(B_xx_con/v2_km_B)))*
((1*(1+(E_xx_con/v2_km_E)))*(1+(B_xx_con/v2_km_B))));
    ...
  end for;
end ad_Spirallus;
```

On the other side, the sensitivity equation system in AD mode is much shorter as follows:

Listing 12.5: Sensitivity equation system in AD mode

```
parameter Real temp3 = 1.0/v2_km_E;
parameter Real temp4 = 1.0/v2_km_B;
equation
  teq3 = 1.0/((1*(1+(E_xx_con*temp3))*(1+(B_xx_con*temp4))));
  for ad_i in 1:NACTPAR loop
    ...
    gp_v2_fwd[ad_i] = (((B_xx_con*temp4)*((E_xx_con*temp3)*
gp_v2_fwd_vmax[PARINDEX[ad_i]])+(v2_fwd_vmax*(gp_E_xx_con[ad_i]*temp3)))+
((v2_fwd_vmax*(E_xx_con*temp3))*(gp_B_xx_con[ad_i]*temp4)))*teq3-v2_fwd*teq3*teq3;
    ...
  end for;
```

Section 13.1 demonstrates the way equations in AD mode are efficiently computed. On both modes, two types of indexes are used:

1. the index ad_i with parameter sensitivities and derivative objects
2. the index $PARINDEX[ad_i]$ for enabling only the selected active parameters

At this stage, it looks as if the sparsity of the input Jacobian is not exploited and that every trivial derivative is represented in the code. In section 13.2, it is shown that high-level Modelica compiler techniques manage to exploit the sparsity.

Chapter 13.

Algorithmic Concepts behind ADModelica

This chapter introduces the algorithmic approach adopted by ADModelica for providing efficient representation of sensitivity equation systems. The classical AD techniques of procedural languages summarized in section 11.2 are not directly applicable for equation-based languages. Therefore, an alternative approach relying on equation-based compiler concepts relevant for the Modelica language is adopted. The algorithmic concepts behind this approach are demonstrated through an example in section 13.1. A complete algorithmic specification of the presented approach is given in (Elsheikh 201xc). Section 13.2 emphasizes Modelica-specific compiler techniques with which generated ADModelica code is subject to optimization.

13.1. AD of equations

13.1.1. Equations vs. assignments

Before diving into algorithmic details of computing an efficient representation of sensitivity equation systems relevant for Modelica, the main differences between equation-based languages and procedural languages like C/Fortran need to be emphasized. Namely, while procedural languages are based on assignments (e.g. $y := f(x)$), Modelica is essentially based on equations (e.g. $f_l(x, y) = f_r(x, y)$, cf. chapter 2). An assignment defines a clear relationship between an output variable y and a set of input variables x . In contrary, an equation is a relation among several variables that needs to be fulfilled concurrently (Fritzson 2003). The notion of causality (i.e. input and output) among these variables is usually absent. This conceptual difference is considered by the way derivatives are computed for Modelica. The main idea is based on the fact that for an implicit equation $f_l(z) = f_r(z)$, the corresponding sensitivity equation w.r.t. a parameter p is the implicit equation $\partial/\partial p [f_l(z)] = \partial/\partial p [f_r(z)]$.

13.1.2. Non-causal equation-based AD

The algorithmic concepts behind the implemented equation-based approach for supporting equation-based languages are illustrated on the following example.

Example 13.1 (AD of equations). Assume that the kinetic equation in (11.4) is viewed as a Modelica equation (i.e. no causality among variables). With the help of the corresponding AST in figure 13.1, a sensitivity equation subsystem can be computed as:

$$\begin{array}{llll}
 v & = & V T_{12} & \implies & v' & = & V' T_{12} + V T'_{12} \\
 T_{12} & = & T_{121} T_{122} & \implies & T'_{12} & = & T'_{121} T_{122} + T_{121} T'_{122} \\
 T_{121} & = & S T_{1212} & \implies & T'_{121} & = & S' T_{1212} + S T'_{1212} \\
 1/T_{1212} & = & (S + K_m) & \implies & T'_{1212} & = & -(S' + K'_m) T_{1212} T_{1212} \\
 T_{122} & = & K_I T_{1222} & \implies & T'_{122} & = & K'_I T_{1222} + K_I T'_{1222} \\
 1/T_{1222} & = & (I + K_I) & \implies & T'_{1222} & = & -(I' + K'_I) T_{1222} T_{1222}
 \end{array}$$

where T_* are intermediate variables computed within intermediate equations and

$$X' = (dX/dp_1, dX/dp_2, \dots, dX/dp_m)^T \quad \forall \text{ variables } X$$

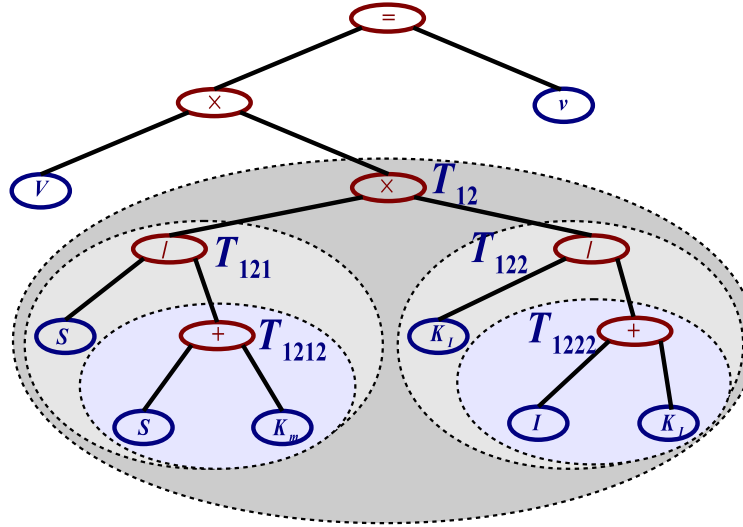


Figure 13.1.: AST of the kinetic term from equation (11.4). Different types of subtrees are considered classifying different potentials of common subexpressions to arise. Some non-leave nodes represent intermediate variables which indices correspond to their position in the AST.

Formal specification for setting up an AST of equations is given in (Elsheikh 201xc). The following facts need to be emphasized:

1. Common sub-expressions are computed by using *intermediate equations* carried out using intermediate variables T_*
2. Reusing of intermediate variables as in example 11.7 is not possible, since the number of identifiers should be equal to the number of equations
3. Intermediate equations are computed by in-order traversal of the AST since the order of equations is irrelevant

In the previous example, not every intermediate computation is performed through a temporary variable. This is done in order to avoid excessive number of intermediate variables, which increases the dimension of the corresponding sensitivity equation system. This is realized by viewing the AST concept a bit differently from the standard case. Each non-leave node is viewed as a root of a subtree. Three types of subtrees are distinguished:

1. subtrees corresponding to a mathematical expression potentially imposing common subexpressions with its derivative
2. subtrees with a root "/"
3. subtrees without any potential for common subexpressions

Each subtree virtually corresponds to an intermediate equation which evaluates the underlying subexpression. As demonstrated in example 13.1, each type of subtree is correspondingly represented by

1. a conventional equation in terms of the next intermediate variables, e.g. $v = V \ T_{12}$
2. symbolically modified intermediate equations for efficient representation, e.g.

$$T_{121} = S \ T_{1212} \quad , \quad 1/T_{1212} = (S + K_m)$$

In this way, the resulting derivatives don't include any divisions

3. conventional subexpressions, e.g. $S + K_m$ is not computed by an intermediate variable

13.1.3. Eliminating derivative objects of intermediate variables

Unfortunately, complete derivative objects need to be declared for intermediate variables. Consequently, the dimension of the underlying DAE system becomes very large. Additionally, intermediate variables are considered as state variables rather than help variables for storing intermediate computations. Therefore, large space of memory is required taking into account that the complete time profile is stored within a simulation. Runtime tests corresponding to such representation are of low performance.

A significant improvement is realized by excluding derivative objects of intermediate variables within sensitivity equation subsystems (i.e. T'_*). Instead, they are rather accumulated within one equation by recursive substitutions until no T'_* is present. For example the sensitivity equation subsystem within example 13.1 can be reduced to the non-simplified equation:

$$\begin{aligned} v' = & V' T_{12} + V [(S' T_{1212} + S (- (S' + K'_m) T_{1212} T_{1212})) T_{122} \\ & + T_{121} (K'_I T_{1222} + K_I (- (I' + K'_I) T_{1222} T_{1222}))] \end{aligned} \quad (13.2)$$

The previous equation can be further optimized by introducing local equations storing common subexpressions as follows:

$$\begin{aligned} l_1 &= V T_{122} T_{1212} \\ l_2 &= l_1 S T_{1212} \\ l_3 &= V T_{121} T_{1222} \\ l_4 &= l_3 K_I T_{1222} \\ v' &= V' T_{12} + l_1 S' - l_2 (S' + K'_m) + l_3 K'_I - l_4 (I' + K'_I) \end{aligned}$$

This representation requires much less FLOPs and is carried out by a reasonable number of intermediate equations.

13.1.4. Efficiency

Taking into account that the Modelica compiler can exploit the sparsity of the input Jacobian (i.e. K', V'), the following table summarizes the number of FLOPs for one evaluation of the sensitivity subsystems w.r.t. m parameters using the presented classical AD approach and the equation-based AD approach:

# Ops	+	-	×	/
Equation (11.4)	2	0	2	2
Classical AD, example (11.7)	$4 + 2m$	m	$4 + 8m$	0
Non-causal equation-based AD	$2 + 2m$	$3m$	$10 + 2m$	0

The table shows that the number of required multiplications is reduced with the equation-based AD approach. However, on the other side, the number of used intermediate variables are increased. Additionally all these intermediate variables are handled from the Modelica simulation environments as real DAE state variables. Consequently, additional memory space is consumed by the equation-based approach. This point is analyzed in more details in section 16.2.

13.2. Modelica-based compiler optimization

The dimension of sensitivity equation systems of typical high-level Modelica models could be so high that space and simulation runtime become not reliable any more (cf. chapter

16. Therefore, ADModelica performs some additional optimization techniques to reduce the dimensionality of the underlying DAE systems. Moreover, the generated code from ADModelica is itself subject to further Modelica-based compiler optimization techniques with which the dimension of automatically generated DAE-systems can be reduced using several methods. This section outlines some of these techniques.

13.2.1. Removal of trivial equations

A typical Modelica model with many connections implicitly represents a DAE system with many trivial equations of the form $u = v$ and $u + v = 0$ at the flat equation level. The number of equations get drastically reduced when only one variable instance for each group of such alias variables is kept (cf. section 3.1). Therefore, a standard Modelica compiler attempts to remove such equations in order to pass an optimized form to the end DAE solver. The derivatives of such equations have also similar alias forms and would be also removed by the compiler. However, associating complete derivative objects to all such alias variables would be memory consuming as also end-results for these variables are tabulated. Therefore, ADModelica removes such equations at the flat model level in a recursive manner as long as such equations further arise after each removal and substitution. This is done until an optimized equation subset is obtained. Consequently derivative objects are associated only to meaningful variables.

13.2.2. Code optimization with the Modelica compiler

The input Jacobian J_{in} from equation (11.6) has usually non-zero elements only at the diagonal elements. The end-generated code declares J_{in} as a two-dimensional array of constants as shown in section 12.3. The sparsity of J_{in} is automatically exploited in two aspects by a standard Modelica compiler:

1. It symbolically simplifies mathematical expressions in which many zero subexpressions are present.
2. Resulting trivial equations are automatically removed from the end optimized DAE systems.

These points are examined by inspecting the end-generated C-code resulting from compiling an ADModelica generated model.

13.2.3. High-level Modelica-based compiler techniques

Figure 13.2 demonstrates an abstract equation system sorted by the techniques illustrated in section 3.3. The directed bipartite graph of the equation system is composed of SCCs. The underlying topological sorting of these SCCs corresponds to the causality among the variables. These sorted equations inherit causality information among variables. Within a typical Modelica simulation environment, such information is not usually utilized for improved solution. Instead, they are usually solved together as a whole ODE/DAE system

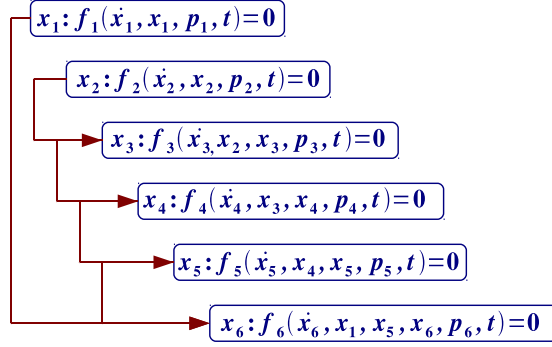


Figure 13.2.: A DAE system in BLT-format

without using a specialized solver exploiting the underlying structure. However, the BLT format is useful for explaining the behavior of parameter sensitivities and providing further optimization opportunities. When computing parameter sensitivities of such systems, there is no need to blindly differentiate all equations w.r.t. all parameters as equation (11.2) suggests. Instead, it is enough to consider the derivatives of the intermediate variables laying in all SCCs of the computational path from the independent variable to the dependent variable (Elsheikh and Wiechert 2008). For example:

- $d\dot{x}_3/dp_2$ can be computed by differentiating only the second and third equations
- $d\dot{x}_4/dp_5$ is zero
- If all parameters are active, the corresponding sensitivity equation system consists of $6 \times 6 = 36$ equations. By exploiting the BLT format, only $1 + 1 + 2 + 3 + 4 + 6 = 17$ equations are needed

Chapter 14.

The Index of Sensitivity Equation Systems

A typical Modelica model is represented by finite number of components and connections. These connections represent algebraic constraints among state variables. Therefore, the underlying DAE system may have a high differential index requiring special handling as illustrated in chapter 4. By applying ADModelica on such a model, sensitivity equations including additional algebraic constraints are augmented to represent parameter sensitivities. Meanwhile, the algebraic constraints remain intact in the augmented model. This gives rise to question about the differential index of sensitivity equation systems and how a Modelica compiler handles such large dimensional systems. This chapter presents a theoretical result concerning the differential index of sensitivity equation systems and its relationship to the differential index of the original DAE system (Elsheikh and Wiechert 201x). Section 14.1 motivates the main idea behind this chapter. Section 14.2 handles the special case of DAE systems of differential index one. Finally, the more general case is handled in section 14.3 using the structural index, introduced in section 4.3.

14.1. Motivation

Given a DAE system generated from a typical Modelica model:

$$F(\dot{x}, x, p, t) = 0, \quad x(t_0) = x_0(p) \quad (14.1)$$

and supposing that this DAE system has a differential index more than one, one way to compute parameter sensitivities can be done by

1. transforming the DAE system into a numerically integrable DAE system of index one or zero (i.e an ODE system) using index reduction techniques (cf. section 4.3)
2. computing the sensitivity equations of the reduced ODE/DAE system. The sensitivity equation system has an identical differential index (cf. section 14.2)
3. integrating the resulting sensitivity equation system

The other way can be done by

1. computing the sensitivity equation system of the DAE system (14.1)

$$\begin{aligned} F(\dot{x}, x, p, t) &= 0, & x(t_0) &= x_0(p) \\ F_{\dot{x}}\dot{x}_p + F_x \cdot x_p + F_p &= 0, & x_p(t_0) &= \frac{\partial x_0(p)}{\partial p} \end{aligned} \quad (14.2)$$

2. reducing the whole sensitivity equation system to a solvable equations system using index reduction techniques
3. integrating the resulting solvable equations system

Apparently, the first way could be preferred because the index reduction algorithm is applied to a smaller system. On the other side, the second way has the advantage that sensitivities are embedded together with the original DAE system in the higher abstraction form similar to the way ADModelica follows, cf. section 12.3. However, the index computation of the sensitivity equation system and the selection process of equations for the task of index reduction become more time-consuming due to the dimension of the sensitivity equation system. Nevertheless, in this chapter it is proven that for a wide subset of DAE systems of the form (14.1), the process of index reduction of the sensitivity equation system (14.2) is similar to the process of index reduction of the original system (14.1) in two aspects:

1. The differential indices of both systems are equal
2. Given which subset of equations is selected for differentiation and algebraic substitution for index reduction of equation (14.1), selection of equations for index reduction of equation (14.2) becomes straightforward.

The next sections clarify for which kind of DAE systems the above arguments are valid.

14.2. DAE systems of differential index one

Two cases regarding the differential index of sensitivity equation systems are distinguished according to the differential index of the original DAE system. In this section it is shown that for a wide class of DAE systems of differential index one, the differential index of the sensitivity equation system is also equal to one. Formally, given the DAE system

$$\begin{aligned} f(\dot{y}(t), y(t), z(t), p, t) &= 0, & y(t_0) &= y_0(p) \\ g(y(t), z(t), p, t) &= 0, & z(t_0) &= z_0(p) \end{aligned} \quad (14.3)$$

where f and g correspond to a set of ODE and a set of algebraic equations, respectively. According to definition (4.5) of the differential index, index reduction is realized by attempting to express \dot{y} as a function in y, z and p . This is done by differentiating the second equation w.r.t. time to obtain:

$$g_y \dot{y} + g_z \dot{z} + \dot{g} = 0 \quad (14.4)$$

If g_y is non-singular, then the given DAE system has the differential index one. It can be shown that the differential index of equation (14.3) together with its sensitivity subsystem

$$\begin{aligned} f_{\dot{y}} \cdot \dot{y}_p + f_y \cdot y_p + f_z \cdot z_p + f_p &= 0, & \frac{\partial y}{\partial p}(t_0) &= \frac{\partial y_0}{\partial p} \\ g_y \cdot y_p + g_z \cdot z_p + g_p &= 0, & \frac{\partial z}{\partial p}(t_0) &= \frac{\partial z_0}{\partial p} \end{aligned} \quad (14.5)$$

is also equal to one. By differentiating the algebraic part, \dot{y}_p and \dot{z}_p can be expressed as an explicit function in y_p, z_p, y, z, p and t as follows

$$\begin{aligned}\dot{y}_p &= -(f_y)^{-1} [f_y \cdot y_p + f_z \cdot z_p + f_p] \\ \dot{z}_p &= -(g_z)^{-1} \left[\frac{d}{dt} (g_z) \cdot z_p + \frac{d}{dt} (g_y \cdot y_p + g_p) \right]\end{aligned}\tag{14.6}$$

Thus, for DAE systems of differential index one, the same numerical techniques mentioned in section 4.4 can be directly applied for integrating the corresponding sensitivity equation systems without applying index reduction techniques.

14.3. DAE systems of arbitrary differential index

This section handles the more general case of DAE systems with arbitrary differential index. The approach taken for the proof is influenced by the way a Modelica compiler handles DAE systems of higher indices. Such DAE systems cannot be solved using common numerical solvers as clarified in section 4.4. They need first to get transformed into integrable DAE systems of differential index zero or one. Attempting to apply index reduction through the standard definition (4.5) of differential index is not practical from a computational point of view. Alternatively, another algorithm based on the structural index definition (4.9) relevant for generalized domain-independent concepts adopted by Modelica is utilized (cf. section 4.3). In this section, it is proven that the DAE system (14.1) and the corresponding sensitivity equation system (14.2) have an identical differential index, in case the structural index successfully estimates the differential index.

14.3.1. Common structural characteristics

In this section, it is shown that the structural index of the sensitivity equation system (14.2) is equal to that of the original system (14.1). This is done by utilizing common structural characteristics between both equation systems. Namely, their corresponding bipartite graph representations from definition (3.1) are isomorphic. Figure 14.1 shows an example of the computational graph representation of the sensitivity equation system of a kinetic equation system of a bi-uni reaction:

$$\begin{array}{ll}\dot{S}_1 &= -v & \partial \dot{S}_1 / \partial \alpha &= -\partial v / \partial \alpha \\ \dot{S}_2 &= -v & \partial \dot{S}_2 / \partial \alpha &= -\partial v / \partial \alpha \\ \dot{P} &= v & \partial \dot{P} / \partial \alpha &= \partial v / \partial \alpha \\ v &= f(S_1, S_2, P, v, \alpha) & \partial v / \partial \alpha &= \partial / \partial \alpha [f(S_1, S_2, P, v, \alpha)]\end{array}$$

The following definition is used to prove this argument for the general case.

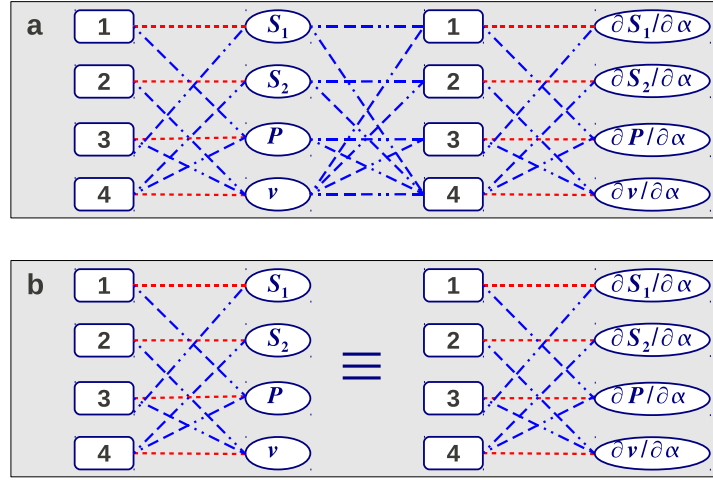


Figure 14.1.: a) The bipartite graph of the sensitivity equation system
 b) The bipartite graph of the original DAE system is isomorphic to the bipartite graph representation of the corresponding sensitivity equations

Definition 14.7 (The induced subgraph). *Given a graph $G = (V, E)$ where V is a set of vertices and E is a set of edges. Suppose that $U \subset V$, then the induced subgraph of G with the vertex set U is defined as:*

$$G[U] = (U, E_U)$$

where $E_U = \{(u, v) \in E : \{u, v\} \subset U\}$

Theorem 14.8. *Let $G = (V, E, w)$ be the bipartite graph representation (cf. definition (3.1)) of the DAE system (14.2) where w is a weight function defined as in definition (4.9) and let $G^* = (V^*, E^*, w)$ be the bipartite graph representation of the sensitivity system (14.2), where*

$$V^* = V \cup V' = V_x \cup V_e \cup V'_x \cup V'_e$$

with V_x, V_e as in definition (3.1) and

$$V'_x = \left\{ \frac{\partial x_1}{\partial p}, \frac{\partial x_2}{\partial p}, \dots, \frac{\partial x_n}{\partial p} \right\}, \quad V'_e = \{e'_1, e'_2, \dots, e'_n\}$$

where e'_i correspond to the derivative of equation e_i w.r.t. p . Consider the graph

$$G' = G[V'] = (V', E', w)$$

then $G' \cong G$.

Proof. Clearly, $\forall x_i \in V_x, i = 1, 2, \dots, n \exists x'_i \in V'$ representing $\frac{\partial x_i}{\partial p}$.

Similarly, $\forall e_i \in V_e, i = 1, 2, \dots, n \exists e'_i \in V'$ representing derivative of equation e_i w.r.t. p .
 $\forall (e_i, x_j) \in E \Leftrightarrow (e'_i, x'_j) \in E'$. Moreover $\forall (e_i, x_j) \in E, w(e_i, x_j) = w(e'_i, x'_j)$. The graph $G' = (V', E', w) = G^*/G \cong G$ by definition \square

The previous theory is in general a way to conclude how Modelica compiler techniques understand index reduction algorithms are applied on sensitivity equation systems. Since the definition (4.9) of structural index is based on its bipartite graph representation from definition (3.1), it is expected that the structural index of the DAE system (14.1) is equal to the structural index of the corresponding sensitivity system (14.2). This is proven in the next subsection.

14.3.2. The structural index of sensitivity equation systems

The structural index is based on the weight of maximal matchings, see definition (4.8). Consequently, it can be concluded that the isomorphic mapping of a maximal matching $M_{max}^n(G)$ is also a maximal matching in G' .

Corollary 14.9.

1. if $M^s(G)$ is a matching $\Rightarrow M'^s(G') = \left\{ (e', \frac{\partial x_i}{\partial p}) : (e, x_i) \in M^s(G) \right\}$ is a matching
2. if $M^n(G)$ is a maximal matching $\Rightarrow M'^n(G')$ is a maximal matching
3. $w(M_{max}^n(G)) = w(M_{max}^n(G'))$.

Proof. Straightforward from theorem 14.8 \square

Remark 14.10. A perfect matching in G^* is necessarily of size $2n$.

It can be shown that a maximal perfect matching $M_{max}^{2n}(G^*)$ is composed of two maximal perfect matchings $M_{max}^n(G)$ and $M_{max}^n(G')$ both of size n in G and G' respectively.

Corollary 14.11. $M_{max}^{2n}(G^*) = M_{max}^n(G) \cup M_{max}^n(G')$.

Chapter 14. The Index of Sensitivity Equation Systems

Proof. \forall edge $v \in M_{max}^{2n}(G^*), v = (e^*, x^*), e^* \in E^*$ and $x^* \in V^*$, there are three cases concerning v :

First case: Either $v \in E$ or $v \in E'$.

In this case, the proof is straight forward using corollary 14.9 and theorem 14.8.

Second case: $e^* \in E$ and $x^* \in V'$.

Then the variable x^* representing a $\frac{\partial x_i}{\partial p}, i \in 1, 2, \dots, n$ is present in an equation $e^* \in E$ in equation (14.2) \Rightarrow (Contradiction)

Third case: $e^* \in E'$ and $x^* \in V$.

Since $M_{max}^{2n}(G^*)$ is a perfect matching, $|V_x'| = |V_e'|$ and any $x' \in V_x'$ must be present exactly in one edge $u \in M_{max}^{2n}(G^*) \Rightarrow \exists u \in M_{max}^{2n}(G^*)$ s.t. $u = (e, x')$, where $e \in V_e, x' \in V_x' \Rightarrow$ (Contradiction according to second case) \square

Corollary 14.12. 1. $w(M_{max}^{2n}(G^*)) = 2 \cdot w(M_{max}^n(G))$

2. $w(M_{max}^{2n-1}(G^*)) = w(M_{max}^n(G)) + w(M_{max}^{n-1}(G))$

Proof. Straight forward using corollary 14.9 and corollary 14.11 \square

Using all results concluded so far, it can be shown that the structural index of the DAE system (14.1) is equal to the structural index of the corresponding sensitivity equation system (14.2).

Theorem 14.13. $ind(G^*) = ind(G)$

Proof. $ind(G^*) = w(M_{max}^{2n-1}(G^*)) - w(M_{max}^{2n}(G^*)) + 1$
 $= w(M_{max}^n(G)) + w(M_{max}^{n-1}(G)) - 2 \cdot w(M_{max}^n(G)) + 1$
 $= w(M_{max}^{n-1}(G)) - w(M_{max}^n(G)) + 1$
 $= ind(G)$ \square

With this proof, it is enough to apply Pantelides algorithm only on the original equation system for selecting the equations to be differentiated. Due to the common structural characteristics of sensitivity system and original system, if an equation $e_i \in V_e$ is selected for differentiation towards structural index reduction, the corresponding equations $e'_i \in V'_e$, obtained by differentiating e_i w.r.t. p , should be also chosen for index reduction of the whole sensitivity system. Note that the above argument applies only to a DAE system which differential index is equal to its structural index.

Chapter 15.

Accuracy of Finite Difference Methods

This chapter is concerned with the accuracy of ADModelica generated models in comparison with FD methods. Section 15.1 analyzes the problematics of standard FD formulas. The following section 15.2 demonstrates a benchmark showing that for the type of models handled in this work, the employment of ADModelica is crucial for providing accurate results in comparison with FD. A more comprehensive treatment of the analysis of FD methods and their comparisons with analytical derivatives can be found (Elsheikh and Wiechert 2012).

15.1. Finite difference methods

One way for numerical computation of parameter sensitivities is to employ FD methods. Assuming that the integration of equation (11.1) leads to the solution:

$$\bar{x}_{i,k}(p) = x_i(t_k, p) + E_{i,k}, \quad i \in \{1, 2, \dots, n\}, \quad k = 0, 1, 2, \dots \quad (15.1)$$

where $|E_{i,k}| < \text{Tol}_R \cdot |x_i(t_k, p)| + \text{Tol}_A$ represents the local error controlled by the given relative tolerance Tol_R and absolute tolerance Tol_A . In the Dymola simulation environment, both tolerances are equal by default. Using Taylor series approximations, the required parameter sensitivities can be explicitly approximated as:

$$\frac{d\bar{x}_i}{dp_j}(t_k, p) = \frac{\bar{x}_{i,k}(p + e_j \delta_j) - \bar{x}_{i,k}(p)}{\delta_j} + O(\delta_j) \quad (15.2)$$

$$\text{for } i = 1, \dots, n \quad \& \quad j = 1, \dots, m \quad \& \quad k = 0, 1, 2, \dots$$

where $e_j \in R^m$ is a unit vector and δ_j is a small scalar value proportional to the value p_j , e.g. $\delta_j = \delta p_j$. The advantage of this way is that the implementation is straightforward and requires $m + 1$ simulations. However, δ_j needs to be carefully selected in order to produce accurate results. From one side, the choice of a big δ_j leads to large numerical errors in the order of $O(\delta_j)$. Therefore, a higher-order *Central Difference* (CD) formula (Eberly 2001, Levy and Lessman 1992) is recommended

$$\begin{aligned} \frac{d\bar{x}_i}{dp_j}(t_k, p) = & \frac{-\bar{x}_{i,k}(p + 2e_j \delta_j) + 8\bar{x}_{i,k}(p + e_j \delta_j) - 8\bar{x}_{i,k}(p - e_j \delta_j) + \bar{x}_{i,k}(p - 2e_j \delta_j)}{12\delta_j} \\ & + O(\delta_j^4) \end{aligned} \quad (15.3)$$

$$\text{for } i = 1, \dots, n \quad \& \quad j = 1, \dots, m \quad \& \quad k = 0, 1, 2, \dots$$

Applying this formula requires $4m + 1$ simulations. From the other side, a very small δ_j results in numerical errors resulting by division. Moreover, a relevant choice of δ_j should consider the used solver tolerance. The approximated derivatives from FD formulas as in equation (15.2) use numerical approximated solutions from equation (15.1). By substituting (15.1) into equation (15.2), the true derivatives values become:

$$\begin{aligned} \frac{d\bar{x}_i}{dp_j}(t_k, p) &= \frac{x_i(t_k, p + e_j \delta_j) - x_i(t_k, p) + O(E_{i,k})}{\delta_j} + O(\delta_j) \\ &= \frac{dx_i}{dp_j}(t_k, p) + O\left(\frac{E_{i,k}}{\delta_j}\right) \end{aligned} \quad (15.4)$$

This equation implies that numerical errors in FD formulas need to be avoided by choosing δ_j which satisfies:

$$\delta_j \gg |E_{i,k}(p)| \quad \forall j = 1, \dots, m \quad (15.5)$$

Hence, an ideal choice for δ_j is different for each variable x_i , time step t_k , the chosen tolerance and the parameter values p . Consequently, there are some situations where the standard FD formula may not accurately reproduce analytical sensitivities since:

- δ_j is constant through the whole simulation time course
- δ_j is constant for all variables $x_i, i = 1, \dots, n$
- For some parameters p_j , a slight modification $p_j \pm \delta_j$ with $\delta_j = p_j \cdot \delta$ (say with $\delta = 0.01$) may cause the corresponding simulation to be not possible due to highly singular Jacobian

This agrees with similar complaints reported in literature (Brenan et al. 1989) concerning the non-accuracy of the computation of the Jacobian by DAE solvers. This is the case although more sophisticated algorithms for FD computation with variable step-sizes δ_j at each time step t_k are performed. Therefore, it is not a surprise to find cases where FD fails to produce acceptable results as shown in the next section.

15.2. Accuracy benchmark

This section compares FD methods against analytical methods for computing parameter sensitivities in terms of accuracy. Figure 15.1 shows a comparison between parameter sensitivities using analytical methods and using FD methods of different order with various settings for the Coryne model from table 6.4. The values of the chosen parameters and variables as well as the used settings corresponding to figure 15.1 are summarized in Table 15.2. The following remarks need to be considered

15.2. Accuracy benchmark

Table 15.1.: The values of the chosen parameters p_j and variables x_i as well as the chosen solver tolerance Tol and the corresponding FD step size δ_j in the benchmark corresponding to figure 15.1

Cases	a	b	c	d	e	f	g	h
Tol	10^{-4}	10^{-4}	10^{-2}	10^{-5}	10^{-12}	10^{-12}	10^{-12}	10^{-12}
δ_j	10^{-2}	10^{-3}	10^{-1}	10^{-4}	10^{-4}	10^{-6}	$10^{-3} * p_j$	$10^{-3} * p_j$
p_j			1.0				$< 10^{-3}$	$> 10^3$
$ x_i \in$			$[10^{-2}, 10^2]$				$> 10^5$	$< 10^{-5}$

- The results using CA technique are not considered since they are identical to parameter sensitivities using AD technique¹
- Some parameter sensitivities were not computable using the formula (15.3). For such cases, specialized handling techniques were performed by inspecting better δ_j or using only forward or only backward FD rather than centred FD formulas

The chosen benchmark reveals some situations where FD results are not accurate as shown in all cases except figure 15.1(e)). This behavior can be explained by equation (15.4) which imply that most of the parameter sensitivities could be reasonably approximated by avoiding:

- big δ_j (e.g. figure 15.1(a,c))
- $\delta_j \approx \text{Tol}_R$ (e.g. figure 15.1(b,d))
- small δ_j (e.g. figure f)

However even with reasonable setting choices, figures 15.1(g,h) still show inaccurate behavior of FD methods with parameters and variables varying within extremely different ranges of values, which is a very typical for large biochemical network models. This implies that within badly-scaled DAE systems, there would be boundary cases where an ideal choice of δ_j which

- does not violate inequality (15.5) for any variable x_i at any time step t_k
- and is not so small or not so large enough for avoiding numerical errors

becomes practically impossible (Elsheikh and Wiechert 2012). This can happen when some variables are highly sensitive to a certain parameter and other variables are almost insensitive to p_j making an ideal choice of δ_j impossible. This can be clarified as follows: for two variables say x_1, x_2 and a parameter p_j where $x_1 \approx O(\log(\log(p_j)))$, a large value for δ_j is required while for $x_2 \approx O(e^{e^{p_j}})$ a small value for δ_j is needed. In some cases, once the physical units of some parameters are relaxed, FD may even behave differently.

¹In general, this has not to be the case since different order of executing arithmetic operations could lead to different results

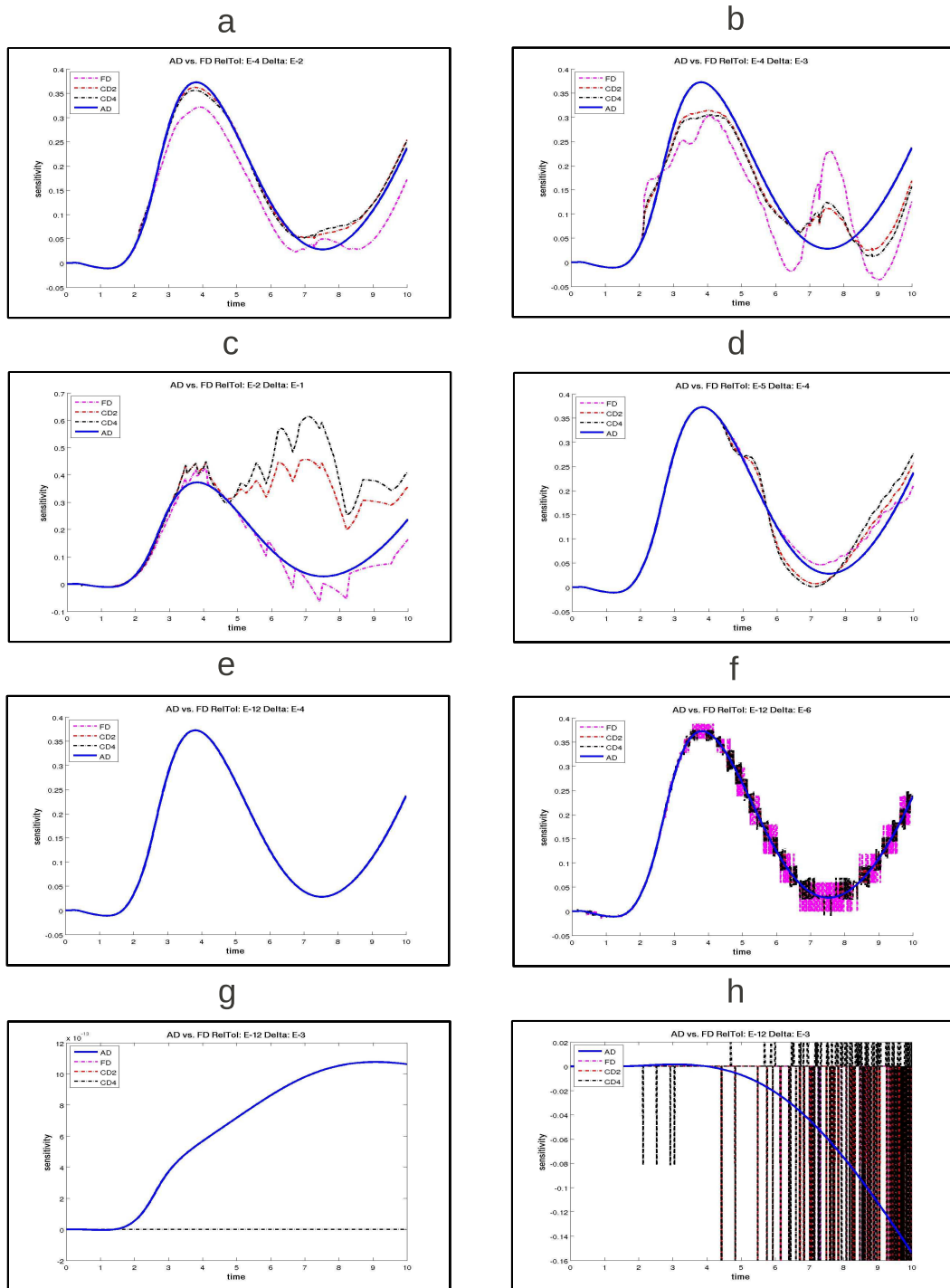


Figure 15.1.: The accuracy of FD methods of order 1,2 and 4 against analytical derivatives with AD, the used settings are summarized in table 15.2

Chapter 16.

Efficient Simulation of Sensitivity Equation Systems

This chapter discusses the performance of ADModelica generated models from a runtime perspective. It shows that the approach adopted by ADModelica is superior to FD methods. In section 16.1, two benchmarks of different size scales for comparing the runtime performance of ADModelica with FD techniques are presented. In section 16.2, the low runtime performance of direct integration of large-size strongly nonlinear sensitivity equation systems is explained through runtime complexity analysis. Such performance drawbacks are overcome by the method described in section 16.3.

16.1. Runtime benchmark

In this section, the so far presented approaches for computing parameter sensitivities of two Modelica models of different size scales are tested for runtime performance. The considered models are the Spirallus model and the Coryne model from table 6.4. Sensitivity equations are computed by ADModelica using SD and AD techniques. The solvers DASSL and LSODAR from section 4.4 are used with different relative tolerances. Root solvers for locating solutions at desired time points were disabled to compare w.r.t. the integration time alone and reduce the influence of I/O overhead. Simulations are performed from $t = 0$ to $t = 10s$. The benchmarks were performed on a Linux machine with Intel Xeon processors with a clock rate of frequency 2.93 GHz and cache size of 4096 KB. The best simulation out of ten simulations for each setting is recorded.

A light benchmark: The Spirallus network

The following table shows the runtime performance of computing parameter sensitivities of the Spirallus model:

Solver	LSODAR				DASSL			
RelTol	10 ⁻⁴	10 ⁻⁶	10 ⁻⁸	10 ⁻¹⁰	10 ⁻⁴	10 ⁻⁶	10 ⁻⁸	10 ⁻¹⁰
CD (s)	0.49	0.52	0.51	0.52	0.51	0.54	0.57	0.64
SD (s)	0.08	0.08	0.09	0.11	0.17	0.26	0.37	0.57
AD (s)	0.05	0.05	0.07	0.08	0.09	0.12	0.18	0.27

Chapter 16. Efficient Simulation of Sensitivity Equation Systems

The size of the corresponding sensitivity equation system is 1525 with 24 active parameters. The performance of AD is 2 – 10 times better than CD and twice better than SD. Given that N_F and N_J correspond to the number of function evaluations and Jacobian evaluations performed by the Newton-like iteration (4.14). Similarly, N'_F and N'_J correspond to the same performed by the extended Newton-like iteration (B.3) for integrating the sensitivity equation system. Each Jacobian evaluation is followed by a factorization. The following statistics justifies the previous table

Solver	LSODAR		DASSL	
	N_F	N_J	N_F	N_J
DAE (11.1)	282	5	365	21
CD (15.3)	$\times 24 \times 4 =$		$\times 24 \times 4 =$	
	27072	480	35040	2016
	N'_F	N'_J	N'_F	N'_J
AD eqs. (11.1) + (11.2)	1251	4	4707	20

These statistics were generated with relative tolerance 10^{-4} . For higher tolerances, similar behavior concerning the relationship between N_F and N'_F as well as N_J and N'_J can be reported. For this benchmark the following remarks are emphasized:

- $N'_F \ll N_{F,CD} \times m$: much less function evaluations for integrating the sensitivity equation system are needed in comparison with function evaluations by CD
- The underlying Newton-like iteration of the sensitivity equation system has not been executed m times more, although the dimension is $m + 1$ times larger
- $N'_J \approx N_J$: the number of Jacobian factorization needed is almost the same

These remarks are used in the analysis given in the next section.

A strongly non-linear large benchmark: The Coryne network

For the second benchmark, 113 active parameters were selected making the dimension of the sensitivity equation system equal to 19732 with runtime performance as follows:

Solver	LSODA				DASSL			
RelTol	10^{-4}	10^{-6}	10^{-8}	10^{-10}	10^{-4}	10^{-6}	10^{-8}	10^{-10}
CD (s)	27.4	27.8	33	38	28.6	31.8	37.6	41.6
SD (s)	∞	∞	∞	∞	∞	∞	∞	∞
AD (s)	95	137	∞	∞	156.9	209	∞	∞

Simulations that took more than 300 seconds were terminated. The table shows that SD is not reliable at all. AD is much worse than CD and is not competitive for small tolerances. The bad runtime performance can be analyzed through the following statistics with relative tolerance equal to 10^{-4} :

16.2. Runtime and space complexity

Solver	LSODAR		DASSL	
	N_F	N_J	N_F	N_J
DAE (11.1)	907	20	1281	33
CD (15.3)	$\times 113 \times 4$		$\times 113 \times 4$	
	409964	1040	579012	14916
	N'_F	N'_J	N'_F	N'_J
AD eqs. (11.1) + (11.2)	81624	23	106243	30

The inefficient runtime performance can be explained by:

1. $N'_F \approx N_{F,CD} \times m$ is very large in comparison with the Spirallus model
2. The factorization of a very large dimensional Jacobian

The first reason implies that the underlying large dimensional Newton-like iteration (B.3) is executed so often when integrating the sensitivity equation system.

16.2. Runtime and space complexity

In this section, an estimation of runtime and space complexity for simulating sensitivity equation systems is presented. Formally, given that:

- $f(n)$ is the average runtime complexity of performing one step of the Newton-like scheme (4.14) for simulating the DAE system (11.1) of dimension n
- Similarly, $f'_{SD}(mn + n)$ and $f'_{AD}(mn + n)$ for simulating the DAE system (11.1) augmented with the corresponding sensitivity equations (11.2) computed by SD and AD techniques for m active parameters, respectively

and under the assumption that the runtime complexity of a DAE system is dominated by f , table 17.1 shows the runtime complexity of all presented approaches for computing parameter sensitivities. Determining the complexity of $f(n)$ is complex and depends on:

1. The used ODE/DAE solver, its underlying sparse nonlinear equations solver and whether excessive re-evaluations and factorizations of the Jacobian are avoided

Table 16.1.: An estimation of runtime and space complexity of sensitivity analysis of DAE systems. N, N' correspond to the number of required iterations by the corresponding Newton-like schemes. l is the number of intermediate variables.

DAE System	Integration Complexity	Space Complexity
DAE (11.1)	$O(N f(n))$	$O(n N)$
CD (15.3)	$O(m N f(n))$	$O(m n N)$
SD DAE (11.1) + (11.2)	$O(N' f'_{SD}(mn + n))$	$O(m n N')$
AD DAE (11.1) + (11.2)	$O(N' f'_{AD}(mn + n))$	$O(m (n + l) N')$

2. The nonlinearity of the underlying DAE system, input parameter values p and tolerance settings determine N as well as how often the Jacobian needs to be re-evaluated and factorized (cf. section 4.4)
3. In case no factorization is done: the cost of DAE expression evaluation dominated by the underlying number of FLOPs
4. Otherwise: the complexity of factorizing the sparse Jacobian

With the assumption that Jacobian factorization dominates functions evaluations, the following relation is assumed to hold

$$N f(n) = N_F g(n) + N_J (g(n) + h(n)) \approx N_F g(n) + N_J h(n) \quad (16.1)$$

where N_F, N_J : # of steps with/without factorization respectively

g : # the average cost of evaluating DAE rhs-expressions/iteration

h : # the average cost of factorizing a sparse Jacobian of size $n \times n$

Similarly, the complexity of direct integration of the sensitivity equation system becomes:

$$\begin{aligned} N' f'_*(mn + n) &\approx N' f'_*(mn) = N'_F g'_*(mn) + N'_J (g'_*(mn) + h(mn)) \\ &\approx N'_F g'_*(mn) + N'_J h(mn) \quad , \quad * \in \{AD, SD\} \end{aligned} \quad (16.2)$$

From table 17.1, a scenario that makes the complexity of integrating sensitivity equation systems be of the same order of CD can be formulated as:

$$\begin{aligned} O(N' f'_*(mn + n)) &\equiv O(m N f(n)) \Leftarrow \\ N' \in O(N) \quad \wedge \quad [O(f'_*(mn)) &\equiv O(m f(n))] \quad , \quad * \in \{AD, SD\} \end{aligned} \quad (16.3)$$

That is, N' should be in the same order of N and executing the Newton-like iteration for a sensitivity equation system should be in the order of executing the iteration m times for the original DAE system. In general, the ratio of the number of FLOPs of a differentiated equation to that of the original equations is linear for AD¹. Since,

$$g'_{AD}(mn) \leq 4mg(n) \in O(mg(n))$$

that is, evaluating the expression of a sensitivity equation system computed by forward AD method is in the same complexity order of evaluating the original DAE system m times. Thus, using equations (16.1) and (16.2), and assuming $N_J = N'_J$, equation (16.3) can be relaxed to:

$$\begin{aligned} O(N' f'_{AD}(mn + n)) &\equiv O(m N f(n)) \Leftarrow N'_F \in O(N_F) \wedge \\ [O(h(mn)) &\equiv O(m h(n)) \vee N_J h(mn) \in O(m N_F g(n))] \end{aligned} \quad (16.4)$$

That is, given that $N'_F \in O(N_F)$, if any of the following two cases is valid:

¹By decomposition of an equation into a set of binary equations, the derivative of a binary operator includes at most 4 operations (i.e. division)

16.3. Runtime performance improvement

- either the integration complexity is dominated by DAE expression evaluations rather than the few iterations N_J where the Jacobian get factorized
- or the factorization complexity of the extended Jacobian is linear in terms of m

then, direct integration of sensitivity equation system becomes at least as efficient as CD. The presented benchmarks show different behavior: In the Spirallus network benchmark, $N_F < N'_F \ll m N_F$ and the performance shows that at least one of the above conditions is satisfied. For models like the coryne model, $N'_F \approx m N_F \gg N_F$, the performance becomes uncompetitive. This is also the type of models where the performance of some state-of-the-art advanced integrators suffer as explained in appendix B (Li et al. 2000).

For space complexity analysis, assume that $s(n)$ expresses the space complexity required for one evaluation of a DAE system of dimension n . Then the space complexity of one evaluation of the corresponding sensitivity equation system with the equation-based approach becomes $s(mn + n + ln)$ where l is the average number of required local variables per equation for intermediate expressions. The following conclusion concerning the space complexity of equation-based AD approach can be implied:

$$s(mn + n + ln) \in O(s(mn)) \quad \Leftarrow \quad m > l \vee l \in O(1) \quad (16.5)$$

Formally, with increasing number of active parameters m , the space complexity of the equation-based AD approaches the order of the space complexity of FD methods as long as $N'_F \approx N_F$ can be assumed.

16.3. Runtime performance improvement

The analysis in the last section shows that the runtime performance of direct integration of sensitivity equation systems suffers from high-dimensional Jacobian as well as from high number of iterations for the underlying Newton-like scheme. To avoid or reduce these drawbacks, the method described in (Dickinson and Gelinas 1976) is implemented. In this method, the original DAE system (11.1) is simulated m times, each time together with one sensitivity subsystem from equation (11.2). Although the original DAE system is integrated m times in this way, smaller Jacobians need to be factorized and the number of the required Newton-like iterations gets decreased. Further improvement is done in this work by simply allowing more sensitivity subsystems to be simulated together rather than only one sensitivity subsystem. The implementation can be easily realized by the parameterized code generated from ADModelica in which the set of active parameters can be specified (cf. section 12.3). The following table shows the improved performance for the Coryne benchmark:

Solver	LSODA				DASSL			
RelTol	10^{-4}	10^{-6}	10^{-8}	10^{-10}	10^{-4}	10^{-6}	10^{-8}	10^{-10}
CD	27.4	27.8	33	38	28.6	31.8	37.6	41.6
AD	95	137	∞	∞	156.9	209	∞	∞
1 Pars/Sys	43	43.3	50.8	54.1	41.9	44	45.5	57.7
2	25.1	25.6	30.6	35.1	23.5	26.5	30.5	37.6
4	14.4	15.4	21.4	25	15.8	16.1	23.3	24.1
8	12.1	13.7	18.8	23	12.4	14.2	20.2	24.3
16	15	18.5	32.2	44.3	15.3	20.1	29	40.5

The result shows that AD is twice better than CD and that simulating 8 sensitivity subsystems together was the best setting for this model. Under the assumption that $N'_F \in O(N_F)$ motivated by the underlying statistics, equation (16.4) concerning the runtime complexity of this way can be relaxed to :

$$O(N' f_{AD}(mn + n)) \equiv O(m N f(n)) \quad \Leftarrow \\ [m/k O(h(kn)) \equiv O(m h(n)) \quad \vee \quad N_J m/k h(kn) \in O(m N_F g(n))] \quad (16.6)$$

where k is the number of sensitivity equation subsystems. With a small value of k , small dimensional Jacobian need to be factorized, whereas for a large value of k less repetitive computations are performed although the number of iterations of the underlying Newton-like increases. Hence, with a relevant choice of k , integrating sensitivity equation becomes at least as efficient as the performance of CD if

- the complexity of factorizing the extended Jacobian of sensitivity equation systems of smaller dimension has a linear behavior
- the number of DAE expression evaluations dominates the low factorization cost

Consequently, equation (16.6) implies that the performance gap between the adopted approach and using advanced solvers based on utilizing the structure of the Jacobian (cf. section B) vanishes with increasing number of active parameters m . However, the main drawback is that the structure of the Jacobian of a sensitivity equation system, shown in equation (B.4), is not utilized for cheap factorization. However, ADModelica benefits from the advanced solvers used by which Jacobian factorization is performed only for few iterations. In this case the runtime cost of DAE expression evaluations may dominate the factorization.

Another advantage of the presented method is that it is easily parallelizable. In (Ke 2009), the sensitivity equation system of the model EMP from table 6.4 with 334 active parameters and nearly 500 000 equations were tested. Conventional direct simulation was not possible any more. By parallelizing the simulation of sensitivity subsystems with one parameter with 8 processors, it took about two minutes on the mentioned hardware. The main drawback is that the simulation time required usually differs for each sensitivity subsystem making the distribution of computationally identical tasks for each processor non possible. However parallelization behavior is linearly scalable up to 8 processors.

Part V.

Applications of Sensitivity Analysis

Chapter 17.

Large-scale Parameter Estimation

In (Mauch et al. 1997), it was stated that the practical realization of sensitivity analysis of large complex metabolic systems might be virtually impossible if the necessary differential equations were to be created manually due to the tremendous efforts of differentiation. In this part, it is shown that not only this is possible through automatic differentiation, but also heavy computational tasks based upon sensitivity analysis and related to ill-posed inverse problems can be reliably performed. For instance, an example is model selection introduced in chapter 10 where the best subset of "valid" models need to be identified out of combinatorially high number of model candidates. The identified subset of models needs to best describe available data measurements as well as to accurately predict further experiments. Consequently, the key issue for ensuring a successful model identification of high quality is an efficient parameter estimation process in terms of good ratio of accuracy to runtime performance as well as good parameter identifiability. Nevertheless, the underlying parameter estimation problems is known to be ill-conditioned and multimodal so that traditional optimization algorithms fail to yield satisfactory solutions.

This chapter addresses the main difficulties behind such problems (section 17.1) and efficient strategies for tackling them are presented. Namely, the equivalent optimization problem of identifying semi-optimal start values with which derivative-based optimization strategies is considered (section 17.2). Targeting the solution of this problem, standardized hybrid derivative-based strategies are developed and evaluated against naive derivative-based multistart strategies (section 17.3). A comprehensive specification of derivative-based hybrid heuristics is given in (Elsheikh 201xa).

17.1. Common problems behind large-scale parameter estimation

Typical parameter estimation problems aim at minimizing the distance $r : R^m \rightarrow R$ between simulation results $x(p, t_j) \in R^N$ of a given DAE system of the form (11.1) and measurement data $\tilde{x}(t_j) \in R^N$ at discrete time points t_j with $j = 1, \dots, S$ in the sense of a least square problem:

$$\begin{aligned} r(p) &= (Q^T Q) / 2 , \\ Q &= [q_{1,1}, q_{1,2}, \dots, q_{1,S}, q_{2,1}, \dots, q_{N,S}]^T \in R^{NS \times 1} , \\ q_{i,j} &= w_{i,j} (\tilde{x}_i(t_j) - x_i(p, t_j)) \end{aligned} \tag{17.1}$$

where $w_{i,j}$ is a weight factor derived from standard deviation of errors in the measurements $\tilde{x}_i(t_j)$. In order to assist statistical analysis, these errors are assumed to be independent and follow a normal distribution. For this type of computational problems, where $x(p, t)$ represents the simulation results of DAE systems described in the Modelica language, many difficulties exist of both technical and algorithmic nature. Moreover, the reliability of the results provided by the underlying parameter estimation algorithm needs to be questioned and analyzed.

At the algorithmic level the considered parameter estimation problem is hard to solve due to the nonlinear dynamics of the underlying models. Standard global optimization algorithms, to which open-source implementations are available, can not claim to solve it with certainty in finite time (Moles et al. 2003). While some deterministic global optimization methods (e.g. branch and bound (Leyffer 2001)) have sound theoretical convergence properties, the associated computational effort increases exponentially with the problem size (Schweissgut and Wiechert 2010). Stochastic methods and other metaheuristics (Talbi 2009) (cf. appendix D.2) which have weak theoretical guarantees of convergence, require excessive computations for finding a nearly global optimum (Mendes 2001). On the other side, derivative-based local optimization methods are not reliable. They have instable behavior with DAE systems corresponding to reaction networks of practical size if no constraints are given, i.e. for start-values chosen even very close to the optimum the algorithm may diverge. This makes the results of constrained optimization very doubtful as usually optima at boundaries are found. Moreover, in contrast to the popular common belief, naive multistart strategies of local derivative-based methods (e.g. Levenberg-Marquardt) are unsatisfactory (Boender and Romeijn 1995).

Most of these problems can be explained by the nature and the structure of the problem as well as the inaccuracy of data measurements and high correlation among parameters. Consequently, most of the parameters are poorly identifiable, i.e. they can not be uniquely estimated. Even for DAE systems corresponding to small reaction networks with optimal set of well-identifiable parameters and data, derivative-based methods require very good start values close enough to the global optimum in order to converge as have been done in Biegler (Tjoa and Biegler 1991). In contrary, if start-values are chosen in a fair way, without knowing the global optimum a priori, only local optima are going to be identified (Mendes and Kell 1998). This deficiency to converge to a global minimum may have dramatic consequences in the context of reaction kinetics (Michalik et al. 2009).

At the technical level, a practical realization of parameter estimation for Modelica models was hindered by many difficulties. For instance, parameter values suggested by any optimization algorithm may cause the Jacobian of the underlying DAE system to become numerically singular that the simulation time is excessively enlarged. In this way, a single simulation may hinder the whole parameter estimation process. Without providing solutions to these problems, implementation of strategies for successful parameter estimation

would not be possible. Appendix D.1 briefly summarizes these technical difficulties and presents some solutions to them.

17.2. Identification of optimal start values

For the type of models this work is concerned with, derivative-free methods are not reliable at all in terms of computational power needed. Therefore, derivative-based optimization methods should be employed in a way or another in order to achieve acceptable results in a reasonable amount of time. However, derivative-based methods require very good start values for identifying good optima. The problem of identifying optimal start values is formally summarized as follows: Given a derivative-based optimization method (e.g. by a Newton-like scheme method), and assuming that $S_i \subset R^m$ correspond to all subspaces such that any start point $p_0^i \in S_i$ converges by the given method to a local optima p_{loc}^i and $S_i \cap S_j = \emptyset \forall j \neq i$, then in order to locate the global optimum¹:

$$\left\{ p^* \in R^m : r(x(p^*, t), \tilde{x}(t)) \leq r(x(p_{loc}^i, t), \tilde{x}(t)) \text{ for } i = 1, 2, \dots \right\}$$

optimal start values $p_0^* \in S_{p^*}$ converging to p^* need to be identified. The determination of such suitable start values p_0^* for high dimensional parameter estimation problems that are located within the global convergence non-convex area S_{p^*} of a derivative-based optimization algorithm is not trivial because:

1. The space $S_p \subset R^m$ of physically admissible parameter values is typically large.
2. The sensitivity system whose solution is required for computing parameter sensitivities is usually solvable only in a subspace $S_{sol} \subseteq S_p$ according to the chosen solver and the tolerance input settings.
3. Some parameter values can cause numerical difficulties that are associated with the numerical solver. The subspace covering such parameter values is denoted by $S_{diff} \subseteq S_{sol}$. For instance, it is often the case that suggested parameter values make simulation time very long due to semi-singular Jacobians.
4. Ideally, global convergence is theoretically guaranteed for start values $p_0 \in N_\epsilon(p^*)$ from a convex subspace $S_{N_\epsilon(p^*)} \subseteq S_p^*$ around the global optimum p^* .
5. Nonlinear optimization problems usually have numerous local optima. If the underlying parameters set is not well-identifiable (i.e. high correlation among parameters exist), then there are infinite number of local optima.

Hence, start values should ideally be chosen within the subspace:

$$S_{conv} = S_{N_\epsilon(p^*)} \cap (S_{sol}/S_{diff})$$

¹or a global optimum in a numerical sense

in order to efficiently locate the global optimum p^* . Since S_{conv} is notably smaller than S_p , naive multistart optimization strategies are not practical for high dimensional problems. Most arbitrarily chosen start values either diverge or converge only to a local optimum as it will be shown in the next section. Moreover, computational efforts are significantly increased by the computation of parameter sensitivities. The key issue behind global optimization methods presented in this work is directed towards locating reasonably good start values. The trick is to view the problem of identifying the global optimum as an optimization problem of identifying optimal start values in a reasonable amount of time with which a local derivative-based method converges to a good solution. In the following sections, different strategies for improving the search within the space S_p are presented.

A further fundamental improvement for this type of high-dimensional badly-scaled models was to apply parameter scaling techniques illustrated in standard literatures. As shown in the next sections, such techniques show significant improvements in the results since the local convergence area is enlarged (i.e. $S_{N_\epsilon(p^*)}$). Moreover, the behavior of such methods show improved stability and even unconstrained optimization becomes also possible (Nocedal and Wright 2006). Although applying such techniques is straightforward to implement (cf. appendix D.3), professional software such as LEVMAR (Lourakis Jul. 2004), IPOPT (Wächter and Biegler 2006) and the Matlab Optimization toolbox (see www.mathworks.com) don't provide such opportunity.

17.3. Hybrid heuristics

As already mentioned, there is no global optimization algorithm that can find optimal solutions in acceptable amount of time from a practical perspective. For the considered parameter estimation problem, it would be advantageous to collectively apply several types of algorithms in order to utilize the advantages of each algorithm and avoid the disadvantages of each one as much as possible. This can be achieved by employing hybrid heuristic strategies which require as inputs the considered optimization problem together with a set of optimization algorithms and produce best possible solutions as outputs. Then, this is realized by solving the input optimization problem according to various mechanisms for hybridizing the given input optimization algorithms. Hybrid heuristics have been shown to provide the best solutions of many practical and known academic combinatorial optimization problems (Talbi 2009). This section presents a first study of the effectiveness of employing hybrid heuristics for the considered type of problems. It can be considered as an extension of the study presented in (Talbi 2002) in two aspects:

- The same techniques exclusively applied to combinatorial optimization problems (i.e. discrete search space) are examined with nonlinear programming problems
- The input optimization algorithms may contain non-metaheuristics like derivative-based methods (cf. appendix D.2 for the notion of metaheuristics)

Here derivative-based methods cannot be completely considered as metaheuristics since additional information (i.e. derivatives) is required and hence are not applied as black-boxes to input problems. Two modes for applying hybrid heuristics are demonstrated:

1. Relay mode: a set of optimization methods applied in a pipelined way where the output solutions of one method are the inputs of another
2. Low-level cooperative mode or co-evolutionary hybridization: a cooperative optimization model where an input method performs a search in the solution space of another method

For the set of input algorithms, out of two families of classes, the following methods are selected as input methods to the above modes:

- Naive multistart derivative-based methods: Newton-like method with trust-region approach (e.g. Levenberg-Marquardt)
- Evolutionary population-based methods: Differential evolution method known to be among the best derivative-free methods in academical competitions (Storn and Price 1997)

17.3.1. Relay mode: Scatter search

Instead of performing naive derivative-based multistart strategy, that is applying computationally expensive derivative-based methods to a randomly generated population of start values, it is less costly to apply a derivative-based method to start-values generated by an evolutionary method. Namely, the space S_p is first cheaply explored by the chosen evolutionary-based algorithm. The resulting population of parameters are distributed along the space S_p and are taken as start-values for a derivative-based optimization algorithm. In this way, the resulting start-values are already nearby local optima to which convergence is fast and computation of sensitivities are not consumed on badly located start values. If the evolutionary-algorithm finds a solution within the convergence area of the global optimum p^* for a derivative-based method, the global optima (or at least very nearby) would be found by the derivative-based method.

The method derived by this mode already resembles an existing published method in literature called Scatter method (Rodriguez-Fernandez et al. 2006). However, the published method is more sophisticated and includes more strategies such as generation of start-values using a logarithmic distribution (to intensify the search near boundaries) and handling flat regions with deterministic exact strategies (e.g. branch and bound). The overall method with all these sophisticated strategies can still be characterized as a hybrid method since the initialization of populations can be controlled by the initialization strategy of population-based methods and handling flat regions can be done by inserting additional methods in the methods pipeline of the relay mode.

17.3.2. Co-evolutionary hybridization

The scatter search philosophy is to find start points with quality function of low values for derivative-based algorithms. This implies that such strategy may not be able to find global optima belonging to very narrow convergence areas not necessarily identifiable through start values with quality function of low values. The strategy resulting by the co-evolutionary hybridization mode can overcome such difficulties as it is optimizing the generation of start values for derivative-based method in the space S_p . This is realized by letting the objective function of an evolutionary-based method start a derivative-based optimization for few iterations. Namely, for each population of start values, only parameter vectors with which derivative-based optimization shows a fast convergence behavior are used for generating the next set of start values. This would be more or less a directed search in S_p . In the first couple of populations, the behavior remains the same as by a naive multistart strategy. In advanced populations, parameter vectors are generated out of various mixture of parameter components that contributed to a fast convergence. As shown by the benchmark in the next subsection, the resulting strategy is significantly better than naive multistart strategy in terms of computational cost and results quality.

17.3.3. Benchmark

As a benchmark for both modes, the Spirallus model from table 6.4 was chosen to test the efficiency of parameter estimation against the naive multistart strategy by which derivative-based optimization is performed on randomly generated start values. The following configuration was used by the benchmark:

- For each kinetic parameter a value in the interval $[0, 1000]$ according to a uniform distribution is generated²
- Simulated data are generated out of forward simulation from time $t = 0s$ to $t = 15s$ using the default parameter values as published in the dissertation (Wahl 2007)
- The simulation results are extracted at equidistant discrete points at every $0.2s$
- Each entry in the simulated data is randomly generated according to a normal distribution whose mean value is equal to the corresponding simulation result and standard deviation equal to 0.1
- $w_{i,j}$ in equation 17.1 were scaled in a way to make r not so small

The implementation of both modes is straightforward using the Matlab optimization toolbox together with an open-source implementation of the differential evolution algorithm provided by the author of this method. Figure 17.1 shows a comparison between hybrid heuristics and naive multistart strategy with and without parameter scaling. The following table shows also some statistics of the results.

²In (Moles et al. 2003), some subsets parameters referred to as hill coefficients are allowed to vary only in a smaller interval

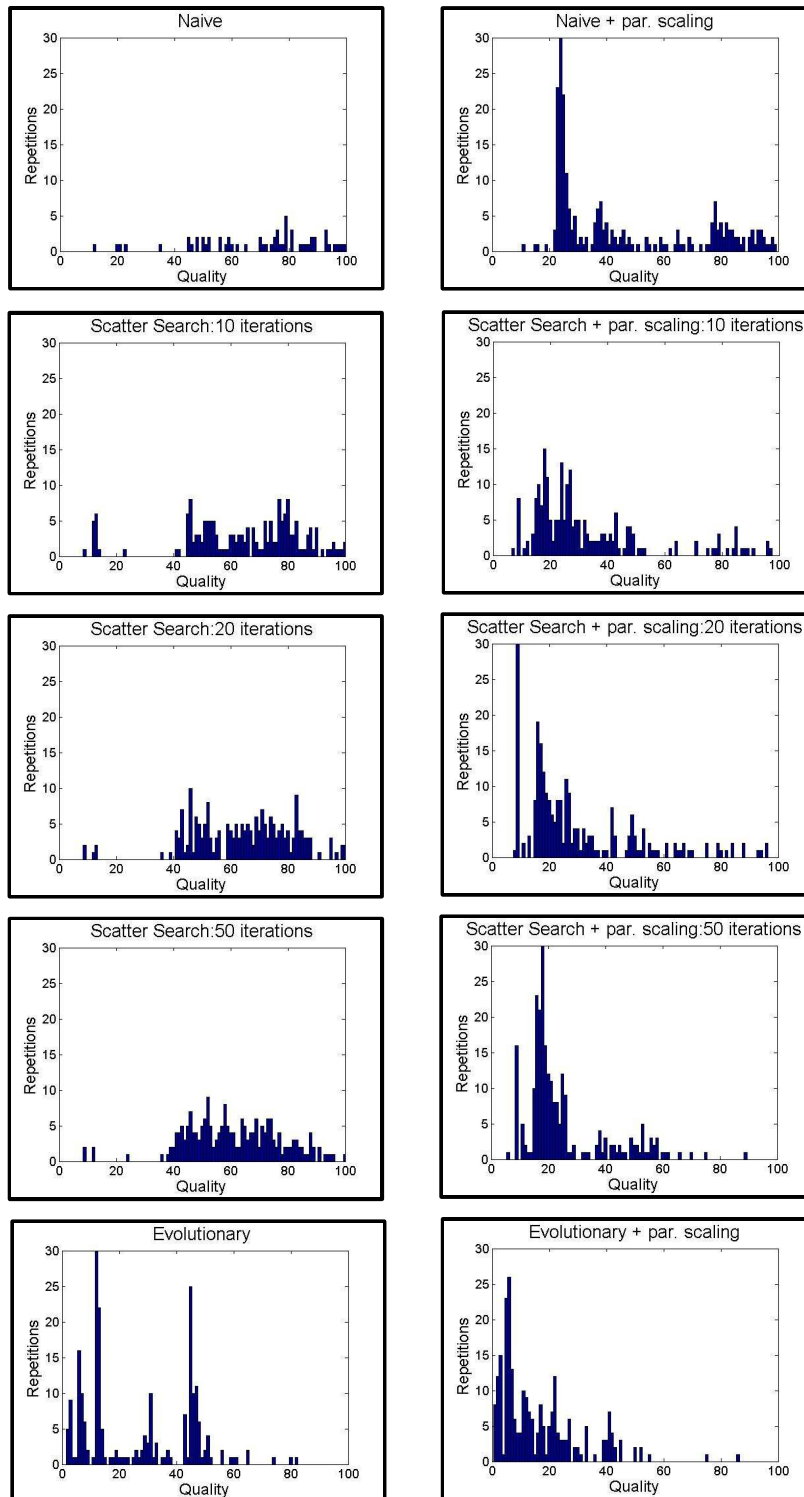


Figure 17.1.: The performance of different multistart strategies. The resulting qualities are scaled for illustration purposes. The left hand (right hand) side corresponds to multistart without (with) parameter scaling, respectively 109

Chapter 17. Large-scale Parameter Estimation

Table 17.1.: Outputs of the presented multistart strategies. The term population refers to the initial number of parameter vectors generated. Only simulations of sensitivity equations are counted. Runtime of each method is dominated by the number of required simulations.

Methods	Naive	Scatter Search					Evolutionary
# of populations	1000	250					250
# of DE iteration	-	1	10	20	50	100	20
best quality (r)	12.28	20.58	8.84	8.74	8.62	8.62	1.75
# of pts < 100	60	18	174	207	206	225	250
# of simulations	57424	9829	7525	6210	5647	5253	c.a. 75000
with parameter scaling							
best quality (r)	11.33	18.51	7.49	8.42	5.71	5.71	1.25
# of pts < 100	247	55	213	243	248	249	250
# of simulations	50353	7555	7386	7059	7180	7305	c.a. 65000

Based on the figure and the statistics, the following conclusions are emphasized:

1. Parameter scaling significantly improves the quality and the stability of parameter estimation independent of the method used
2. In scatter search, with very few simulations much better results can be achieved
3. In evolutionary-based multistart, with nearly the same number of simulations the best results can be achieved.

In this work, direct comparisons between scatter search and evolutionary methods is avoided due to many reasons. First of all, only one type of evolutionary-based strategy has been examined (i.e. differential evolution). Other various evolutionary algorithms have not been examined in this work. Moreover, for scatter search only so few simulations have been performed that a direct comparison is not fair any more. For the used settings, increasing the number of populations does increase the number of simulations but does not improve the end results. This behavior might not be necessarily the same if other settings or different type of evolutionary algorithms are used. For these reasons, this study does not draw any conclusions concerned with direct comparisons of both hybrid modes.

Appendix D.4 introduces further global optimization methods that have been implemented and examined in this work. Nevertheless, hybrid heuristics are the recommended one due to the simplicity of implementation, applicability to this type of models and the quality of the results.

Chapter 18.

Statistical Analysis of Parameter Estimation

Identifiability of model parameters

Determining the parameter values that minimizes a least-square function is only a part of the parameter estimation problem. An equally important part is to supply measures for the reliability of the estimated parameters (Johnson 1992). Namely, desired is to determine the identifiability of model parameters and their precisions. This can be done by examining the presence of dependencies among model parameters and determining confidence regions corresponding to the propagation of measurements errors on the estimated parameters.

A widely used method for computing such confidence regions is based on an approximation of the covariance matrix using Fischer Information Matrix (FIM) of the linearized model in the neighborhood of the best parameter estimates (Marsili-Libelli et al. 2003). This is done under the assumptions that data measurements errors are independent and follow a normal distribution. Dependencies among model parameters can be also detected using the approximated covariance matrix. While this method is known to work precisely with linear models, however the resulting confidence regions might be unrealistic for nonlinear models. In order to overcome the doubts behind using the linearized statistical analysis on nonlinear models, more realistic bounds can be determined using the computationally expensive bootstrap method (Efron 1979). In this method, Monte Carlo simulations of parameter estimation using variated data samples are performed and the best estimates are recorded in histograms which correspond to preciser confidence regions.

Case study

Figure 18.1 shows a sample distribution of 20 estimated parameters using the bootstrap method with 256 samples of variated data for the Coryne-based models with and without labellings for describing the central Metabolism from table 6.4. Additionally, for emphasizing the influence of parameter dependencies on parameter identifiability, the corresponding correlation among parameters based on linearized statistical analysis is shown in figure 18.2. The following configuration was used by the bootstrap method:

- In-silico data was generated from time $t = 0s$ to $t = 30s$ for only known measurable metabolites
- For each metabolite (or a label of a metabolite), 5 measurements per second with standard deviation equal to 0.1 were computed
- For each sample, a multistart derivative-based optimization is performed using the LEVMAR software written in C and implementing Levenberg-Marquardt algorithm (Lourakis Jul. 2004)
- 8 start-values were randomly generated in the small interval $[0.9p^*, 1.1p^*]$ in order to attempt avoiding multiple-local optima, where p^* is the parameters set used for computing in-silico data
- For each data sample, the optimized parameter set corresponding to the best quality as well as the corresponding linearized variances were recorded in the result

In order to reduce the computational cost, the expensive computations were performed on the supercomputer JUROPA. This supercomputer includes 2208 computing nodes each of which has 2 Intel Xeon X5570 (Nehalem-EP) quad-core processors of frequency 2.93 GHz and 24 GB of memory. The total number of cores is 17664 with total peak performance of 207 Teraflops making this supercomputer in the 10th-23rd position within the top-500 list of supercomputers within the years 2010 and 2011 (see <http://www.top500.org>). For the non-labeled coryne-based model (cf. Table 6.4), about 2 hours and a half using 32 processors were needed to perform Monte Carlo simulation with the mentioned configuration whereas the most expensive computations with the labeled Coryne-based model with 120 kinetic parameters required about 11 hours using 128 processors.

In order to support performing identifiability tests, a base software supporting parameter estimation as an incremental iterative process has been implemented. This open-source software, available under <https://github.com/AtiyahElsheikh/DecTrees>, provides an implementation of decision trees with which complex hierarchical computational tasks can be assisted. The overall role of this software in supporting parameter estimation and identifiability analysis is illustrated in Appendix E.2 (Elsheikh 2013).

Analysis of the result

According to the histograms of figure 18.1 resulted from the bootstrap method, different groups of parameters can be classified:

1. Well-identifiable parameters with small length of confidence intervals
2. Poorly-identifiable parameters with large length of confidence intervals
3. Insensitive parameters which don't influence the parameter estimation

By a close investigation on the correlation among parameters in figure 18.2, the non-identifiability of some parameters can be explained by the presence of strong correlation among parameters. In the considered examples, about one third of the parameters have strong correlation among them selves. For two or more correlated parameters, the influence of a change in parameter value can be eliminated by a change in another parameter value making the underlying parameters set poorly identifiable. Improved identifiability of parameters can be achieved when some of these parameters are fixed to constants. Consequently, prior to the parameter estimation process, identifiability analysis aiming at identifying non-uniquely estimated parameters should be performed in order to achieve reliable confidence regions as well as to overcome many of the troubles arising in the parameter estimation process.

Another important observation is that the identifiability of kinetic parameters of the labeled coryne-based model get improved due to the availability of more data measurements. This can be investigated through the following observations:

- For well-identifiable parameters, the length of their confidence intervals from the bootstrap method get decreased up to the half
- The shape of some distribution of estimated parameters become similar to normal distributions
- Less presence of dependencies among parameters

The same results have been reported in (Wahl 2007), however using small models like the *Spirallus* model.

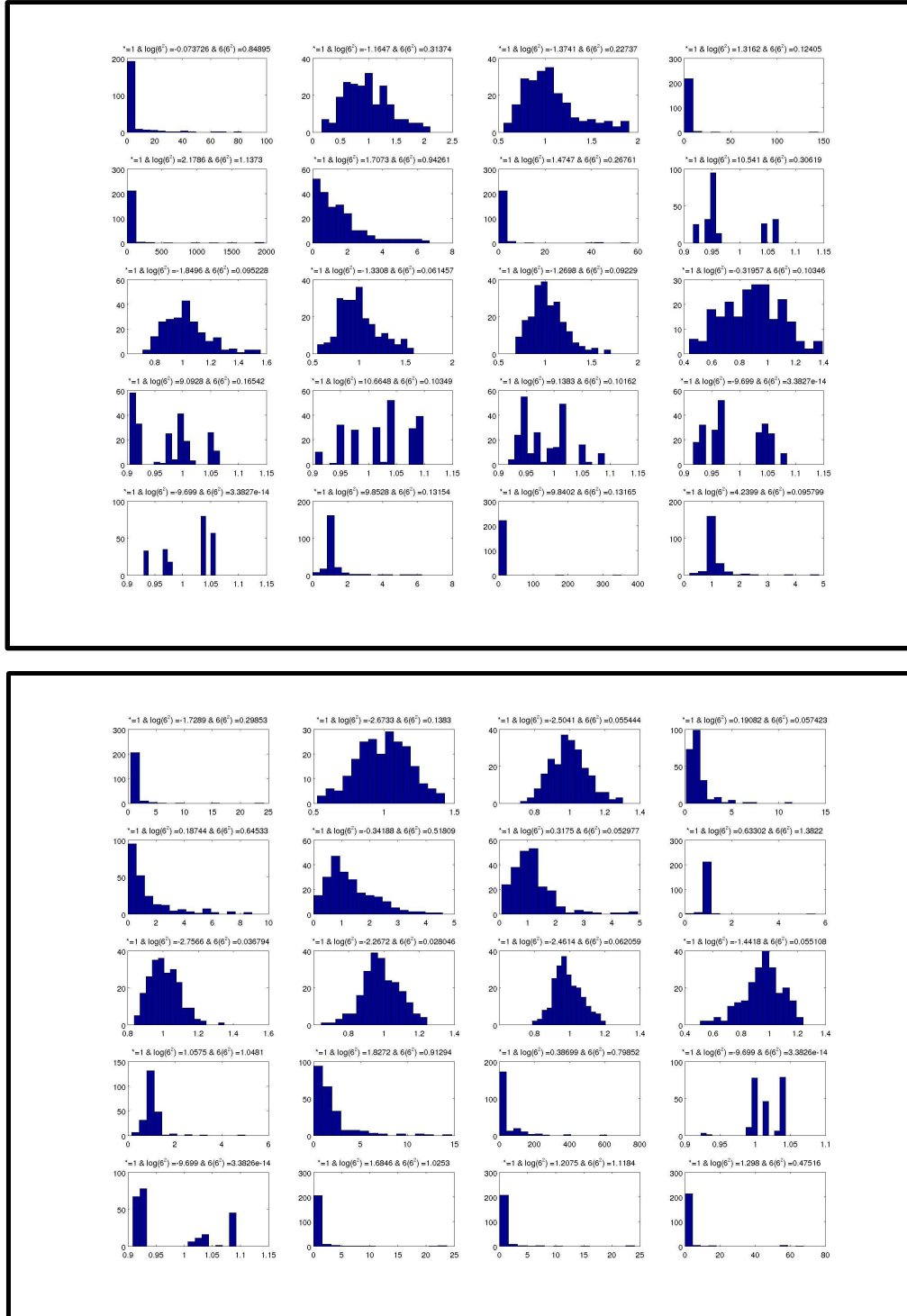


Figure 18.1.: Distribution of a sample of estimated parameters
 Upper: non-labeled coryne-based model
 Lower: labeled coryne-based model

Chapter 19.

Visualization of Parameter Sensitivities

In this chapter, a visualization tool that has been developed within a supervised master thesis for supporting primary applications in the field of Systems Biology is employed (Ke 2009). With this tool, dynamic control coefficients computed with the help of AD are visualized. These coefficients can be viewed as a help tool which gives the opportunity for straightforward identification of significant model parameters subject to modification and experimentation.

19.1. Motivation

For cellular processes, it is very practical to quantify the impact of e.g. environmental conditions, substrate concentration, enzyme amounts or kinetic properties (controlled by genetic modifications) on the amount of product or waste, certain cellular activities, side effects and others. From one side, understanding a biochemical reaction network only from data measurements is very limited due to the high complexity of a large number of effectors and hidden back loops (cf. chapter 5). From the other side, even if the application of a specific modification was partially successful, it cannot be completely explained whether the targeted change is the real cause or not. The success might be raised to global effects caused by a minor change in another hidden variable that is thought to be insignificant.

A way to quantify the influence of changes of enzyme quantities on fluxes and intracellular metabolites with no or little impact of global effects is to perform small changes to them and analyze the response of the system. With the availability of a mathematical model, the influence of infinitesimal changes to inputs can be examined by elementary simulations. Figure 19.1 demonstrates an example using the Spirallus network of figure 6.1. The simulation shows that a slight increment on the value of the maximal reaction rate $v_{2,max}$ results in a decrease of v_6 and hence a decrease of the production of E_{ex} . Similarly, if engineering methods are viewed as a mean for controlling the concentration of intracellular metabolites, simulation shows that a slight increase in the concentration of C results in an increase of the production of F_{ex} .

Instead of employing elementary simulations, it would be more practical for large complex network models to employ the tools of sensitivity analysis and visualization tech-

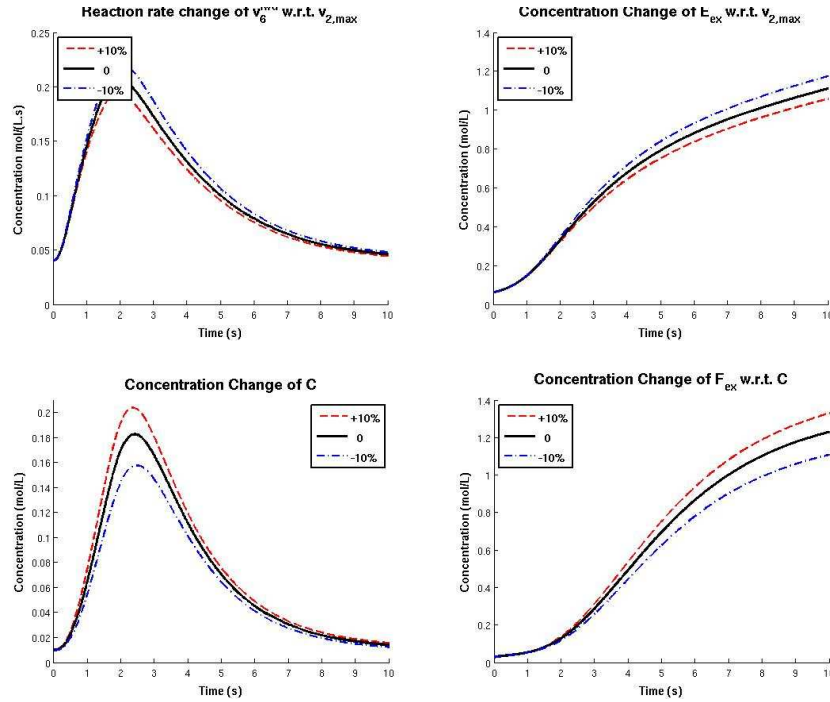


Figure 19.1.: Above: the impact of $\pm 10\%$ change of $v_{2,max}$ on v_6 and E_{ex}
 Bottom: the impact of concentration change of C on F_{ex}

niques to provide means for quantifying the impact of all important parameters in the form of comparable visual pools of information. In the field of Systems Biology, these means are realized by some tools from the field of Metabolic Control Analysis (MCA) presented in the next section.

19.2. Control coefficients

The base for a quantitative analysis of biochemical reaction networks was constructed by MCA through quantitative measures referred to as *control coefficients* for measuring the response of system variables to relative infinitesimal small parameter changes (Kacser and Burns 1973, Heinrich and Rapoport 1974). Initial attempts were done by experimentally altering enzyme activities and measuring the resulting change in fluxes (Fell 1992). However, achieving this at experimental level is usually inaccurate since genetic modifications have usually large scope of influence on various enzyme quantities and large effect on the regulation of effectors (Ehlde and Zacchi 1996).

An ideal alternative for quantifying the impact of input changes to a whole biochemical reaction network is promised by computing *control coefficients* via a model-based approach. Such coefficients can be directly derived from solving the sensitivity equations of the network mathematical equation (6.1):

$$\frac{d}{dt} \frac{\partial c}{\partial \alpha} = N \cdot \left(\frac{\partial v}{\partial c} \times \frac{\partial c}{\partial \alpha} + \frac{\partial v}{\partial \alpha} \right) \quad (19.1)$$

$$\text{with } \frac{\partial c}{\partial \alpha}(0) \quad \text{satisfying :} \quad \frac{\partial v}{\partial c}(0) \times \frac{\partial c}{\partial \alpha}(0) + \frac{\partial v}{\partial \alpha}(0) = 0$$

The initial conditions guarantee that the computed sensitivities expresses the influence of inputs around known steady-state conditions (Mauch et al. 1997). By eliminating the physical units in resulting derivatives, dynamic *Concentration Control Coefficients* (CCC) are explicitly computed as:

$$C_j^{S_i}(t) = \frac{\alpha_j}{S_i} \cdot \frac{\partial S_i}{\partial \alpha_j}(t) \quad \forall i = 1, 2, \dots, m \quad \& \quad j = 1, 2, \dots \quad (19.2)$$

These coefficients can be fairly compared against each other. In addition to CCCs, dynamic *Flux Control Coefficients* (FCCs) are implicitly present in equation (19.1) and can be computed as:

$$C_j^{v_k}(t) = \frac{\alpha_j}{v_k(t)} \cdot \sum_{i=1}^m \left[\frac{\partial v_k}{\partial S_i}(t) \cdot \frac{\partial S_i}{\partial \alpha_j}(t) + \frac{\partial v_k}{\partial \alpha_j}(t) \right] \quad (19.3)$$

The last equation quantifies the impact of a change in α_j on the reaction rate v_k . This magnitude imposes two types of influences:

1. the local influence of the change in α_j
2. the global influence¹ of the resulting change in a metabolite S_i on v_k

The resulting coefficients serve as a pool of detailed comparable information from which the most suitable candidates can be selected for metabolic manipulation as they reveal vital information not available in data measurements alone like:

- which factors (effectors, parameters, etc.) impose the most influence on desired outputs and should be considered as modification targets at experimental level?
- which model parameters are insignificant and could be excluded?
- how far the network model describes the considered biochemical system?

¹This should not however be mixed with global sensitivity analysis.

19.3. Visualization of control coefficients

It would be very practical to be able to:

- compare control coefficients simultaneously against each other
- animate the dynamic FCCs and CCCs along the time axis

through a visualization tool. Techniques and algorithms for visualizing sensitivities have been comprehensively discussed (Noack et al. 2008, Qeli et al. 2004). Within (Ke 2009), a prototype tool for visualizing pure Modelica simulation time-dependent results has been implemented. The tool is based on well-defined architecture based on UML and provide standardized plug-ins mechanisms for extending further functionalities. The tool is an ideal base implementation for developing further advanced visualization features:

1. Scaling: emphasizing small sensitivities against dominant ones
2. Filtering: removing variables and parameters to concentrate on important targets
3. Interactivity: performing previous operations interactively without explicit recomputation from the user side

Within this work, the implemented tool has been employed for visualization of control coefficients of any differentiated Modelica models. The following are some samples of dynamic FCCs and CCCs.

As a benchmark, the *Spirallus* model from table 6.4 is considered. Parameter sensitivities are computed using ADModelica and scaled to control coefficients in Matlab. Visualization of pure control coefficients is inappropriate as only the most significant control coefficients are emphasized due to the large discrepancies among control coefficients. In order to obtain meaningful results, more scaling techniques are applied to emphasize important information and handle possible singularities in the visualization process.

The implemented visualization tool can animate dynamic control coefficients. Figure 19.2 shows snapshots of animated dynamic FCCs w.r.t. maximal reaction rates v_{max} . The most influential parameter w.r.t. all variables along the simulation time axis have the darkest color (corresponding to either of value 1 or -1). The rest of the sensitivities are scaled according to the darkest color. Additionally, the tool provides the feature of excluding and/or inserting additional parameters (i.e. filtering) and variables as shown in figure 19.3 which shows FCCs w.r.t. all kinetic parameters.

19.3. Visualization of control coefficients

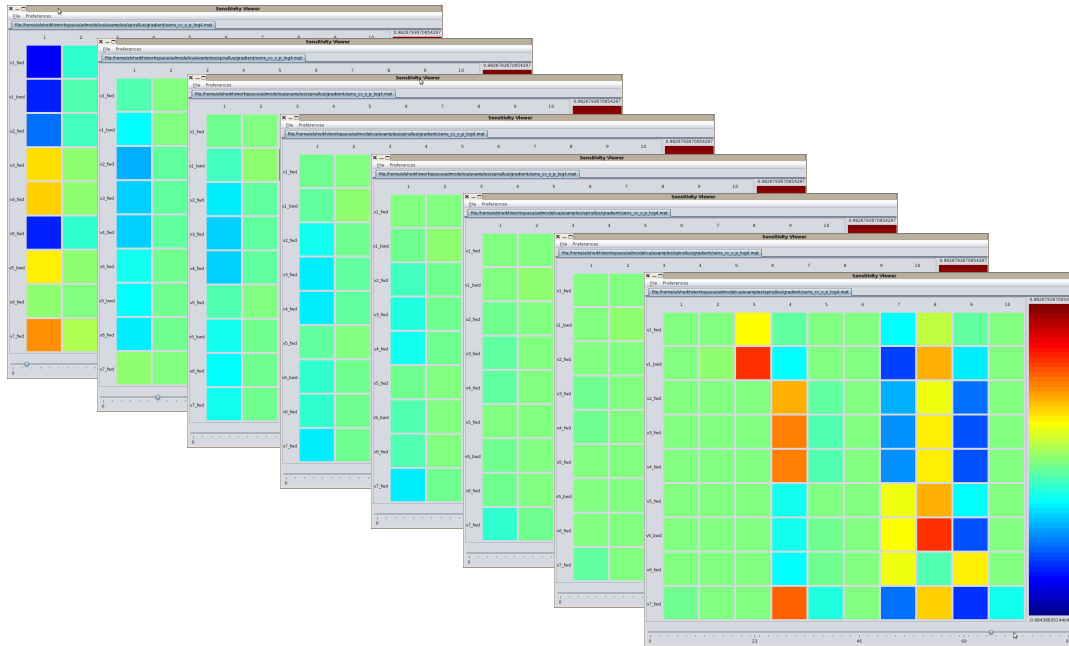


Figure 19.2.: Snapshots of time-dependent flux control coefficients w.r.t. maximal reaction rates

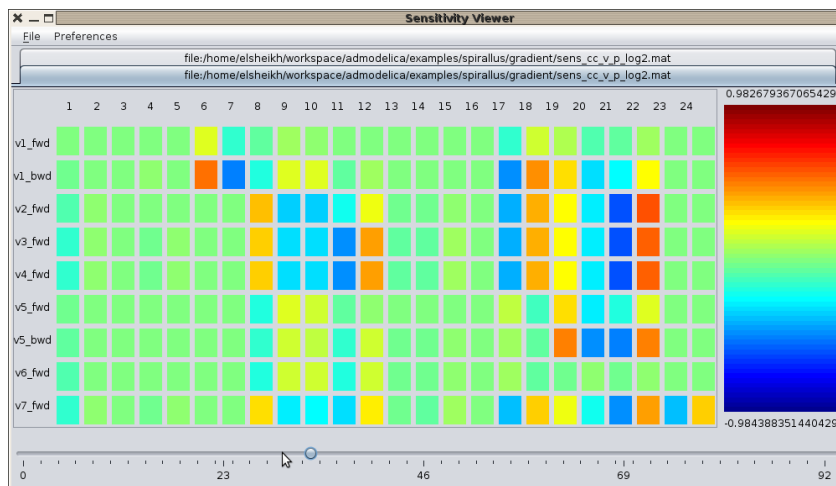


Figure 19.3.: flux control coefficients w.r.t. 24 kinetic parameters

Chapter 19. Visualization of Parameter Sensitivities

Another visualization perspective is to reserve one axis for the time as shown in figure 19.4 in which time-dependent concentration coefficients are provided.

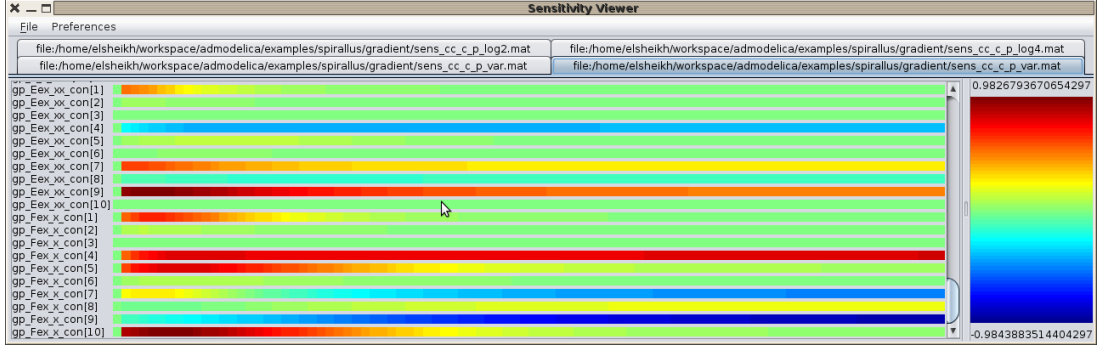


Figure 19.4.: time-course of the concentration control coefficients of the products F_{ex} , E_{ex}

The same concepts have been applied in (Noack 2009) for the vertical model from table 6.4 for identifying targets for metabolic engineering, however using another visualization tool (Qeli et al. 2005).

Chapter 20.

Summary, Outlook and Future Perspectives

20.1. Summary

Along this work, many computational tools, libraries and conceptual notions have been implemented. This section highlights the most promising achievements for the Modelica community within this work. At the level of AD, it is worth to mention that:

1. ADModelica, the first AD tool for Modelica models, has been implemented. It is used for generating Modelica models in which sensitivity equations are efficiently represented. By simulating the generated models parameter sensitivities are computed. These quantities are required by a wide range of applications of sensitivity analysis. The tool is implemented with minimal efforts making use of an already existing open-source compiler. The AD concepts invented within this work fully readopt Modelica-based compiler techniques and notions forming a new AD approach for equation-based languages. The resulting tool supports a subset of Modelica grammar and has been successfully applied on high-level Modelica models in the field of Systems Biology.
2. It has been shown that computing parameter sensitivities analytically via AD is superior to numerical finite difference methods in terms of runtime performance and accuracy. From one side, it is shown that the performance of simulating the automatically differentiated sensitivity equations achieves the expected theoretical runtime performance of forward sensitivity analysis. From the other side, analytical parameter sensitivities of nonlinear badly-scaled DAE systems, a common characteristic of large models, are more accurate and they don't inherit common problems arising in finite difference methods.
3. The absolute advantage of having an AD tool for Modelica models, in addition to the mentioned reasons, is that the generated models are completely platform independent and syntactically represented by legal Modelica grammar. Hence, they can be compiled and simulated within any existing Modelica simulation environment.

Being able to compute parameter sensitivities efficiently, heavy computational tasks have been performed such as:

4. The task of realizing parameter estimation of large-scale Modelica models is typically of a challenging nature due to the poor practicability of conventional methods, nonlinearity of the underlying DAE systems and the presence of measurement errors. Aiming at achieving stable process of parameter estimation as well as high result quality, many efforts both at technical and algorithmic levels have been performed. This includes the implementation of parameter scaling techniques found in standard literature but not usually adopted by common optimization software. Additionally, many non conventional global optimization methods like multiple-shooting and cluster algorithms have been examined. The performed study reveals that among the most promising global optimization methods in terms of high result quality and low implementation efforts are hybrid heuristics. In this work, some heuristics, shown to submit the best results to known combinatorial problems in academic and industry, have been extended to nonlinear optimization problems. The results are significantly better than those produced by naive derivative-based multistart optimization strategies.
5. In addition to the task of locating a global minimum of least-square functions, an equally important task is to examine the validity of the estimated parameters. With the help of linear and nonlinear statistical methods via FIM and Monte Carlo bootstrap method correspondingly, the identifiability of parameters and their confidence regions can be studied. The heavy computational costs have been reduced through the utilization of high performance computing resources for parallel computing. This has been done by implementing software with which communication with Modelica models within C/C++ programs is possible. In this way, it is possible to obtain and manipulate parameter sensitivities of ADModelica generated models within C/C++ programs and link them with open-source optimization software on supercomputer clusters.

Additionally some AD-based tools and further contributions have been implemented some of which are listed as follows:

6. A tool for visualizing scaled parameter sensitivities has been implemented within a master thesis (Ke 2009)
7. Applications of sensitivity analysis in the domain of Systems Biology has been performed in several previous works (Noack 2009, Tillack et al. 2009) as well as currently running PhD theses

Within this work some domain-dependent contributions have been achieved as follows:

8. A compact library for various simplified and generalized kinetic formats for mathematical modeling of highly complex biochemical reactions have been implemented. This library is adequate for applications of automatic model generation.
9. A highly Modelica-based specialized editor for modeling biochemical networks through cooperative works with colleagues has been implemented.

Both previous mentioned achievements can be viewed as an extension to a comprehensive comparative study between the domain-independent Modelica language and the specification language SBML, the standard language established by the Systems Biology community. It has been shown that efficient employment of powerful Modelica language constructs significantly simplifies the implementation of highly specialized tools in the domain of Systems Biology, in a way SBML may not provide. For instance, model families can be completely specified by utilizing the power of Modelica language constructs, which is not the case with SBML. Finally, this work also covers a theoretical result concerning the differential index.

10. The differential index of sensitivity equation systems is equal to the differential index of the original DAE system in case the structural index of both systems is identical.

20.2. Outlook

Running and future works concerning Modelica-based oriented research should consider the fact that the Modelica community is continuously getting larger that common problems not going to remain for ever and standardized solutions will be provided to them. Open compiler tools are going to get improved in terms of compiler enhancements , enhanced numerical solvers and compiler run-time , efficient execution of compiled code on multi-core platforms and tool interoperability (Blochwitz et al. 2011). Various Modelica compilers are already subject for comparisons according to standard benchmarks (Frenkel et al. 2011). This will simplify the task of developing compiler-based and Modelica-oriented tools and enhance collaborative works among the community. Consequently, more specialized Modelica-based tools providing elegant service for special tasks are emerging, e.g. JModelica as a framework environment for rapid prototyping of optimization problems.

Figure 20.1¹ summarizes some achievements throughout this work as well as further future perspectives realizable by implemented tools and already established frameworks.

¹The figure is inspired by a presentation of Prof. Wolfgang Wiechert in FH-Bielefeld on January 2010 with the title simulation tools for biochemical networks

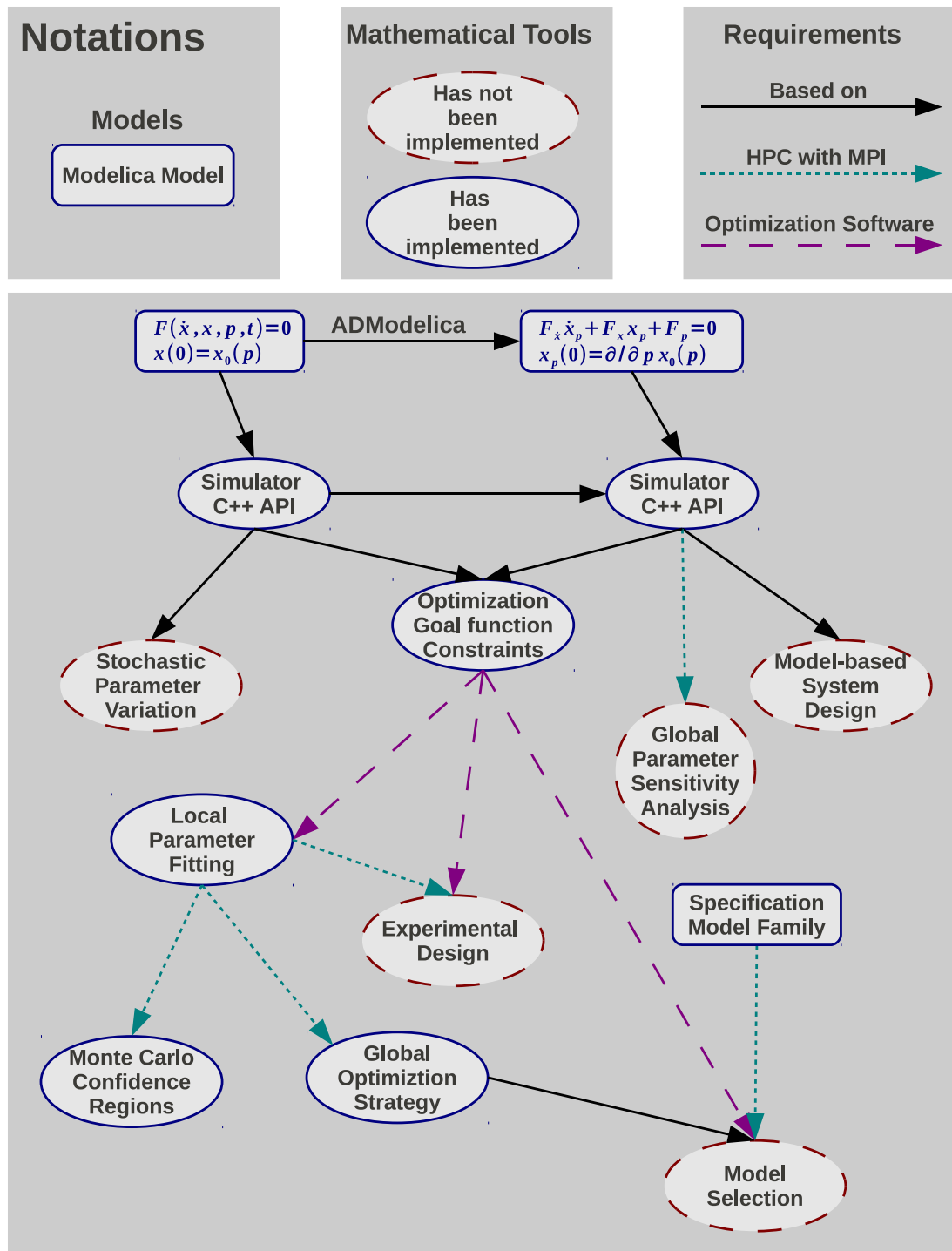


Figure 20.1.: Summary of implemented tools and outlook to future perspectives.

Many of these tools correspond to optimization problems of different classification, e.g. by their mathematical formulation and constraints. Each type of problem needs to be solved using its own optimization algorithm possibly using different optimization software. In order to link a Modelica-based optimization goal function with optimization software, it should be possible to simulate and process the results of Modelica models ideally within the same language with which that optimization software is implemented.

While the so far used simulation environment Dymola provides an elementary Matlab function for this purpose, the underlying optimization problems are typically of computationally expensive nature. The utilization of high performance computing resources for some of these computational tasks is not only a luxury service, but also a requirement for significantly improving the quality of achievable solutions, e.g. model selection. Thus, the computation processes performed for solving such problems need to ideally utilize computational resources for high performance computing, for which Matlab is not necessarily the most ideal available programming language.

Communicating with common Modelica simulation environments within classical languages has been a problem to the whole community due to the absence of standards and software. Realizing meaningful heavy computational tasks was usually achieved through individual efforts by which this work is not an exception. Within this work, a compact but extensible software for communicating with Dymola executables through a well-defined API suited for the performed tasks has been implemented. Only recently and as a an expected result of the progressive evolution of the continuously growing Modelica community, standard solutions for communicating with Modelica models within the C language have been suggested and lately realized by the newest versions of some common Modelica simulation environments (Blochwitz et al. 2011). First successful experiences from different parties with these standards have been published (Andersson et al. 2011, Bastian et al. 2011).

The implemented C++ software within this work is compacter than the standardized solution. On one side, its design was oriented for the few completed tasks. On the other hand, the standardized solution is comprehensive and provides a lot of functionalities that most probably not need by the scope of planned applications, at least in the short-term sense. Nevertheless, due to many software engineering reasons, it is recommended to

1. integrate the standardized solution required into the already established framework

From one side, this eliminates the maintenance efforts required for self-implemented software. From the other side, relying on standardized software solutions enlarges the life cycle of implemented software especially within academical institutions where established tools are largely realized through ad-hoc individual efforts. In this way, the implemented tools are more tractable along generation of developers due to the availability of well-written documentation.

Chapter 20. Summary, Outlook and Future Perspectives

A distinguished criteria for the self-implemented C++ API realized in this work is that special handling of Modelica models generated by ADModelica is provided for processing parameter sensitivities. This simplifies the implementation of derivatives of optimization goal functions required by derivative-based optimization algorithms and software. Therefore, another task is to

2. extend the standardized API with further functionalities for handling ADModelica generated models with the help of the already implemented API

With the availability of a specialized API for handling generated ADModelica models and with the availability of high performance computation powers, the way is paved for

3. realizing further useful mathematical tools such as experimental design, identifiability analysis, global sensitivity analysis and model selection

For instance, the task of realizing model selection can be implemented by using already available tools and following common practice performed throughout this work such as:

- model family specification (cf. chapter 10) and model parsing tools (cf. section 9.5)
- the formulation of an optimization goal function as in equation (10.3) by which a model space is browsed according to a discrete optimization strategy
- global optimization strategies with hybrid heuristics (cf. chapter 17)
- utilization of supercomputers for parallel computing (cf. chapter 18)

Last but not least, the software appendix E.2 provides another future work perspectives concerned with

4. supporting parameter estimation as an incremental process through dynamic interactive menus based on decision systems towards identifiability analysis

Part VI.

Technical Notes

Appendix A.

Semantical Advantages of Modelica against SBML

One of the tasks of a programming language is to enable programmers detecting errors at the early phase of design. Using official SBML language documents, the following items show and give some examples, where Modelica is superior to SBML in such contexts.

Impact of the declaration on the Mathematics

Mathematically, as already shown, kinetic formulas impose structural information that can be utilized for implementing templates for generalized kinetics. These templates can be then specialized according to the number of reactants, products, activators, effectors as well as the reaction reversibility and stoichiometry. These structural information can be utilized in Modelica to implement hierarchies of generalized and specialized kinetic formulas. On the other hand, there is no mean in SBML to utilize such structures. Additionally, reaction declaration has no impact on the mathematical structure. For example, the attributes of *Reaction* like *reversible* attribute has no impact on the construction of the kinetic equations of that reaction which can be freely constructed. The same argument applies for the *stoichiometry* attribute. All these attributes are not used in the corresponding kinetic law.

The completeness of equations

Over- or under-determined system of equations is semantically not a correct SBML-model, nevertheless such systems remain syntactically a valid SBML model. This requires that extra/additional analysis from SBML-based simulation tools to examine the solvability of a system. The same argument applies for system of equations with algebraic loops or invalid equations. A Modelica compiler needs to perform such tasks as well. However, Modelica provides many classifications of components where incomplete equation systems can be specified. For example *partial model* allows abstract types to correspond to under-determined equation systems. These abstract types can be then specialized by complete models. The keyword *model* represent components which must correspond to a solvable equation system. If not, then this represents a violation against the Modelica language and is not syntactically correct model.

Constants and parameters

The use of the term parameter in SBML sometimes leads to confusion among readers who have a particular notion of what something called "parameter" should be. In SBML *parameter* is used for defining both constants and (additional) variables in a model. In SBML the state of a parameter is set through the attribute *constant*. Depending on the value of this attribute, *true* or *false*, an identifier is declared to be constant or a variable. Nevertheless, it is the mathematical structure of the equation which sets an identifier to be a constant or a variable. It is possible to set a constant identifier, which values changes over time according to the system of equations set in the SBML document. Such a conflict can not happen in Modelica. Syntactical constructs of Modelica does not allow such situations to appear. Moreover Modelica introduces two types of constants, one is called *parameter* (i.e. constant value through a single simulation but can change its value from a simulation to another) and the other is *constant*.

Limitation of descriptive power

From the expressibility point of view, the following is an example of using *outside* to model a cell membrane. To express that a compartment with identifier “B” has a membrane that is modeled as another compartment “M”, which in turn is located within another compartment “A”, one would write:

```
<listOfCompartments>
  <compartment id="A"/>
  <compartment id="M" spatialDimensions="2" outside="A"/>
  <compartment id="B" outside="M"/>
</listOfCompartments>
```

This however neither reflects a real topological hierarchies in the same way Modelica does nor does the *outside* attribute affect the mathematical structure of the model. There is a mechanism in SBML for representing hierarchies of compartment types. One *CompartmentType* object instance cannot be the subtype of another *CompartmentType* object; SBML provides no means of defining such relationships.

Start simulation time

Simulation by SBML Model is assumed to begin at time $t = 0$. This is also the time point where initial conditions takes place. If the simulation needs to begin before $t = 0$, this is only possible using a work-around solution, by resetting a dummy symbol representing the time to be equals to time – the start time point of the simulation. In Modelica, initial conditions take place at the user-defined start time of the simulation and not necessarily at $t=0$.

Naming conflicts

An identifier of a local parameter in a reaction can have the same name of an identifier of a global parameter. This introduces the potential for a local parameter definition to shadow a global identifier other than a parameter. In SBML's simple symbol system, there is no separation of symbols within classes and objects; consequently, within the kinetic law mathematical formula, the value of a local parameter having the same identifier as a specie or compartment or other global model entity will override the global value. Modelers and software developers may wish to take precautions to avoid this happening accidentally. In Modelica, such situation is naturally avoided. Each identifier has a globally different dotted notations than other identifiers.

Equation formulation

Rule sections, which enable the listing of additional equations, have a very restrictive standard explicit form of equations. In Modelica, it is up to the user how equations can be formulated, and the matter of transforming these equations into standard formats is left to the compiler. One reason why *AssignmentRule* was introduced in SBML is to enable the user to reform the system of equations in a way that make certain numerical solvers be able to handle the underlying DAE-system. In Modelica, implicit equations are syntactically valid. The task of transforming the system of equations to a solvable system of equations is completely left to the compiler. The Modelica user makes his focus remain on the physical level of system design rather than paying attention at tiny mathematical details.

Redundancy and conflict among rules

Usually, Species, Variables, Parameters etc. can be initialized to literal values using special attributes in the corresponding node definitions. However, if the initial value is represented by a formula, then *InitialAssignment* construct should be used. Although, the value computed by the *InitialAssignment* construct overrides the value stated in the object definition, this way can be a source of redundant and contradictory information in a syntactically valid SBML document. Similarly, the value calculated by an *AssignmentRule* or *InitialAssignment* object overrides the value assigned to the given symbol by the object defining that symbol. This kind of redundant modeling, i.e. the presence of information conflicts is a bad practice and enhance the chance of making errors.

Further restrictions, dependencies, redundancies and overlapping

A wide set of restrictions, dependencies and overlapping rules exist among SBML-constructs. Some can be checked syntactically and some are imposed by the semantics. Some examples are but not limited to:

1. A model must not contain more than one *AssignmentRule* or *RateRule* object having the same value of variable. Similarly, a model must also not contain both an

Appendix A. Semantical Advantages of Modelica against SBML

AssignmentRule and an *InitialAssignment*. The reason is that the semantics of an *AssignmentRule* assumes that it this assignment takes place for $t \geq 0$, while the *InitialAssignment* takes place for $t = 0$

2. The optional attributes *initialAmount* and *initialConcentration* of the *Species* construct, both having a data type of double, are used only exclusively to set the initial quantity of the species in the compartment where the species is located. The construct *InitialAssignment* override these attributes
3. Parameters local to a reaction (i.e. those defined within a reaction's *KineticLaw* object) cannot be changed by rules and therefore are implicitly always constant; thus, parameter definitions within Reaction objects should not have their constant attribute set to *"false"*
4. Any species appearing in the mathematical formula of the *kineticLaw* of a Reaction instance must be declared in at least one of that *Reaction's* lists of reactants, products, and/or modifiers. Put another way, it is an error for a reaction's kinetic law formula to refer to species that have not been declared for that reaction
5. The *stoichiometry* attribute is used to set the stoichiometry of a reaction using. However when the stoichiometry is a rational number, or when it is a more complicated formula, *stoichiometryMath* must be used.

Appendix B.

Specialized ODE/DAE Solvers for Parameter Sensitivities

Unfortunately, direct integration of sensitivity equation systems could be inefficient, specially those corresponding to strongly nonlinear DAE systems and large number of active parameters, w.r.t. which sensitivities are desired. In such cases, the number of iterations performed within the integration process of sensitivity equation systems get exponentially increased (see section 16.1). Alternatively, many other methods for efficient integration of sensitivity equation systems exist. The presented methods can be classified into two categories in which

1. the sensitivity equation system are externally decoupled into smaller DAE systems for fast successive solution (De Pauw and Vanrolleghem 2003)
2. the Jacobian structure of the whole sensitivity equation system is exploited for improved integration and factorization. Decoupling, if any, is done internally at iteration level

An intuitive method from the first category is addressed in (Atherton et al. 1975). Initially, the solution of the DAE system 11.1 is stored in a table followed by successive solutions of the independent sensitivity equation subsystems. Interpolated values are used at times without tabulated values. Another way is to solve the DAE system 11.1 together with only one sensitivity equation subsystem (Dickinson and Gelinas 1976). This requires the solution of DAE of size $2n$ for m times. A modification of this method is reused for efficient simulation in section 16.3.

For a quick overview of methods from the second category, w.l.g. the explicit ODE system:

$$\dot{v} = h(v, p, t) \quad , \quad v(t_0) = v_0 \quad (\text{B.1})$$

is considered. The corresponding parameter sensitivities can be computed by integrating the original ODE system together with the corresponding sensitivity subsystems:

$$\dot{s}_i = \frac{\partial h}{\partial v} s_i + \frac{\partial h}{\partial p}, \quad s_i(t_0) = \frac{\partial v_0(p)}{\partial p_i} \quad (\text{B.2})$$

Appendix B. Specialized ODE/DAE Solvers for Parameter Sensitivities

$$\text{where} \quad s_i = \frac{\partial v}{\partial p_i} \quad \text{for} \quad i = 1, 2, \dots, m$$

As illustrated in section 4.4, the problem of solving an ODE system using common solvers employed by common Modelica simulation environments is transformed to the problem of solving a nonlinear equation system using the iteration scheme of (4.14). By solving the sensitivity equation system, this scheme is extended to:

$$\hat{M} [\hat{v}_n^{m+1} - \hat{v}_n^m] = -\hat{G}(\hat{v}_n^m) \quad (\text{B.3})$$

$$\text{where} \quad \hat{v}_n = \begin{pmatrix} v_n \\ s_1 \\ \vdots \\ s_m \end{pmatrix}, \quad \hat{G}(\hat{v}_n) = \hat{v}_n - \delta_n \beta_{n,0} \hat{h}_n - \hat{a}_n, \quad \hat{h}_n = \hat{h}(\hat{v}_n, p, t_n) = \begin{pmatrix} h(v_n, p, t_n) \\ \frac{\partial h}{\partial v} s_1 + \frac{\partial h}{\partial p_1} \\ \vdots \\ \frac{\partial h}{\partial v} s_m + \frac{\partial h}{\partial p_m} \end{pmatrix}$$

and \hat{a}_n analogously. The Jacobian of \hat{G} becomes:

$$\hat{M} \approx \frac{\partial \hat{G}}{\partial \hat{v}} = I - \delta_n \beta_{n,0} \frac{\partial \hat{h}}{\partial \hat{v}}, \quad \frac{\partial \hat{h}}{\partial \hat{v}} = \begin{pmatrix} M & \dots & 0 \\ \gamma J_1 & M & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ \gamma J_m & \dots & M \end{pmatrix} \quad (\text{B.4})$$

$$\text{where} \quad \gamma = \delta_n \beta_{n,0} \quad \text{and} \quad J_i = \frac{\partial}{\partial v} \left[\frac{\partial h}{\partial v} s_i + \frac{\partial h}{\partial p_i} \right]$$

Common methods attempt to provide efficient implementation by exploiting some of the following characteristics:

- the linearity and structure of sensitivity system (B.2) for problem decomposition, sequential solution and parallelization
- the structure of the Jacobian \hat{M} for cheap factorization as only one diagonal block M need to be factorized when convergence conditions are not satisfied

The following is a list of some these methods and summary of their basic differences:

1. Direct Staggered (Caracotsios and Stewart 1985): The iteration (4.12) is solved first until feasible solution is achieved with a δ_n that satisfies convergence conditions, then the iterations for sensitivities is performed. The disadvantage of this method is that the structure of \hat{M} is not utilized for cheap factorization.
2. Simultaneous (Maly and Petzold 1995): The whole resulting nonlinear system (B.4) is solved directly without making use of the linearity of the sensitivity system. However \hat{M} is fairly approximated for cheap factorization.

-
3. Corrected Staggered (Feehery et al. 1997): A combination of both methods above in the sense that linearity is exploited for sequential solution and meanwhile, a cheap approximation of the full Jacobian is maintained without the need to perform factorization at each time step.

Simultaneous and corrected staggered method are considered to be more efficient than direct staggered since the Jacobian needs not to be evaluated at each time step. This is apparently clear for large DAE Systems resulting from discretization of time-dependent PDE/PDAE where the number of unknowns is much more than the number of parameters. However, in (Li et al. 2000) it is shown that for strongly nonlinear DAE systems where the number of parameters is more than the number of unknowns (i.e. state variables), the staggered direct method outperforms the other methods. This is not surprising since the Jacobian needs to be factorized at many time steps due to the nonlinearity of the DAE system, and hence the bookkeeping of the Jacobian and its factorization becomes an excessive load to the computational complexity. In this thesis, the type of considered models (kinetic modeling) are also similar to those preferable by the direct staggered method. Software implementing these methods provides different modes for error controls:

- only at state variable level
- at both levels of state variables and sensitivities together
- at both levels but independently (i.e. each s_i is controlled independently)

While robust error control leads to more accurate results, performance may suffer as many backtracking steps with decreasing step sizes δ_n are done. The structure of the sensitivities provides parallelization opportunities, despite of the potentially serial part of the state variable iteration. Several parallelization modes can be found in literature:

- with repetitive computation and minimal communication overhead
- with minimal repetitive computation and more communication overhead
- A mixture of both

The second and third modes are better when communication is done among processors within the same core (Hartwich et al. 2011). The first mode is more scalable by not being limited to the number of processors and even further speed up is achieved by cache effects due to complexity reduction. The third approach has been adopted in the supervised master thesis (Ke 2009).

Appendix C.

Possible Approaches for Differentiating Modelica Models

C.1. On which level should AD be applied?

Using OMC, it is not only possible to parse a high-level library-based model but also to access its underlying solvable optimized mathematical formulation. Consequently there are different abstraction levels of Modelica models each of which have a different representation on which AD techniques can be applied. These levels are: high-level descriptive Modelica code, the underlying pure mathematical formulation as a DAE system (flat code) and the generated low-level simulation code in C. In order to justify the chosen approach in this work, all possibilities are first discussed and compared according to the following criteria:

1. Implementation efforts required
2. The compilation complexity of the augmented model
3. The runtime complexity of the augmented model

AD at library level

Given an input Modelica model based on a high-level Modelica library (e.g. section 7.2), it is possible to augment the library components with generalized gradient equations. By importing the augmented library within the given input model, parameter sensitivities automatically become present using additional minimal declaration for specifying the input Jacobian. Appendix C.2 gives an example of this approach for the Modelica language. The library ADGenKinetics (chapter 8) is another example of an algorithmically differentiated library. Detailed guidelines for modeling parameter sensitivities within ADGenKinetics are given in (Elsheikh 201xb). In (Bischof et al. 2005), a similar approach has been adopted for differentiating the XML specification of library components of DAE-based modern languages. This approach is pretty much similar to the operator overloading approach for classical languages in the sense that augmented components are overloaded with the gradients. However, the structure of the underlying DAE system is not available so that specific code optimization cannot be done. Moreover, the complexity of the

Appendix C. Possible Approaches for Differentiating Modelica Models

Modelica language requires large implementation efforts due to the variety of language constructs that are used by typical Modelica libraries.

AD at flat level

Another alternative is to apply AD at flat equation level. It is straightforward to access the flattened equations using the OMC compiler. The whole DAE system is accessible for symbolic manipulation and code optimization. The disadvantage is that the equations at this level don't impose any causality information among variables. This makes traditional AD techniques for classical languages such as C/Fortran not directly applicable. Section 13.1 presents an equation-based approach for computing an efficient representation of sensitivity equation systems. Another disadvantage is that the compilation complexity of typically high dimensional sensitivity equation systems is enlarged. However, as it is shown in several contexts, high-level Modelica compiler techniques can be utilized for optimizing the generated sensitivity equation systems for improved runtime performance.

AD at C-code level

Another approach is to operate on the lowest level, the generated C-code, by available AD tools (Andersson et al. 2010). Specialized solvers introduced in section B can be utilized for achieving maximal possible runtime performance (Imsland et al. 2009). In this case, sensitivities would be computed at background and are not explicitly present as Modelica code. There are many problems using this approach. First of all, generated C-code does not only differ from one Modelica vendor to another, but may also vary from one version to another. Moreover, the generated C-code may utilize some commercial libraries, that are not easily reachable as a third-party tool as it is the case with the Dymola tool (section 2.1). In this case, the approach is either to self implement a Modelica compiler which requires a lot of efforts. Another way is to extend existing developer-oriented compiler tools. However, this makes the end product reliable on a specific compiler and simulation environment that are used by a smaller part of the Modelica community.

The following table shows a comparison between the possible approaches according to the chosen criteria.

Criteria - Levels	C-code	Flat	Library
Implementation efforts	--	+	--
Vendor-Independence	--	++	++
Potentials for code optimization	++	+	--
Compilation Complexity	++	-	--
Runtime Complexity	++	+	-

In summary, the most reasonable approach for the Modelica community is to represent the derivatives within a Modelica model. This makes the generated Modelica model independent from Modelica compilers and accessible for common simulation environments.

In this work the flat model approach is favoured due to the minimal efforts required for implementation. From one side, compilation complexity is rather much higher than low-level approaches. From the other side, as shown in chapter 16 the runtime complexity of the adopted approach lies in the region of the expected runtime performance of forward sensitivity analysis is not more expensive than the runtime complexity of FD approach. Applying AD at the library level provides an attractive alternative. Namely, once individual components of a library are algorithmically differentiated, parameter sensitivities of base models can be provided forever.

C.2. Automatic differentiation of Modelica libraries

This subsection presents a different approach for computing parameter sensitivities of Modelica models. The idea behind this approach is to differentiate the internal models of the library (eg. Resistor, Capacitor, etc) instead of the top-level model (e.g. a model of an electrical circuit). This is done by augmenting the internal models with equations for generalized derivative formulas, while the top-level model remains (virtually) unchanged. Once a library is differentiated, all models based on this library can represent parameter sensitivities using minimal changes. In order to illustrate this approach, a simplified version of the standard electrical Modelica library is taken. All independent components and connectors of the library are augmented with code for representing the derivatives. For example, connectors are augmented as follows:

Listing C.1: Connection for electrical circuits

```
connector ElectricalPin
  parameter Integer NG = 0; // augmented
  Real v "Voltage";
  Real g_v[NG]; // augmented
  flow Real i "Current";
  flow Real g_i[NG]; // augmented
end ElectricalPin;
```

The gradient of potential and flow variables in a connector are also potential and flow variables, respectively. Similarly, components are augmented as follows:

Listing C.2: Implementation of a capacitor

```
model Capacitor
  parameter Integer NG = 0; // augmented
  parameter Real C = 1e-6; // augmented
  parameter Real g_C[NG];
  Real v "Voltage";
  Real g_v[NG]; // augmented
  flow Real i "Current";
  flow Real g_i[NG]; // augmented
  ElectricalPin p
    (ElectricalPin.NG=this.NG); // augmented
  ElectricalPin n
```

Appendix C. Possible Approaches for Differentiating Modelica Models

```

(ElectricalPin.NG=this.NG); // augmented
equation
  p.i + n.i = 0;
  p.v - n.v = v;
  p.i = i;
  p.i = C * der(v);
  // augmented
  for ad_i in 1:NG loop
    p.g_i[ad_i] + n.g_i[ad_i] = 0;
    p.g_v[ad_i] - n.g_v[ad_i] = g_v[ad_i];
    p.g_i[ad_i] = g_i[ad_i];
    p.g_i[ad_i] = g_C[ad_i] * der(v) + C * der(g_v[ad_i]);
  end for;
end Capacitor;

```

For the electrical circuit model shown in figure 2.1, minimal changes to its implementation shown in figure 2.3 are performed to specify the active parameters as follows:

Listing C.3: Implementation of a simple electrical circuit

```

model SimpleCircuit
import ADElectrical; // the differentiated library
  Resistor R1(R=15,NG=3,g_R={0,0,0});
  Resistor R2(R=50,NG=3,g_R={0,0,0});
  Resistor R3(R=20,NG=3,g_R={1,0,0});
  Capacitor C(C=0.01,NG=3,g_C={0,1,0});
  Inductor L(L=0.1,NG=3,g_L={0,0,1});
  VoltageSource VS(NG=3,g_V0={0,0,0});
  Ground G(NG=3);
equation
  connect(VS.p,R1.p);
  connect(R1.p,L.p);
  connect(R1.n,R2.p);
  connect(R2.p,C.p);
  connect(C.n,G.p);
  connect(L.n,R2.n);
  connect(R2.n,R3.p);
  connect(R3.n,G.p);
  connect(G.p,VS.p);
end ADSimpleCircuit;

```

The top model is minimally changed as follows:

1. The augmented library is imported
2. For each declared component, the number of gradients is passed by name as well as the input Jacobian

By this way the whole configuration assembles the original DAE system together with the sensitivities w.r.t. the parameters R_3 , R , C and L .

Appendix D.

A Survey of Optimization Algorithms

D.1. Technical difficulties of large-scale parameter estimation

The following issues represented a great obstacle for a practical implementation of parameter estimation:

- Modelica simulation environments provide no or little common developer-oriented tools for communication within classical languages by which common optimization algorithms are implemented
- The Jacobian of DAE systems may often become very ill-conditioned by parameter values suggested through common optimization algorithms. The simulation of the corresponding DAE systems is so time-consuming that parameter estimation becomes not reliable at all
- Large-scale parameter estimation requires a lot of settings and configuration procedures that would be tedious if done at code level

Successful parameter estimation was not possible before resolving such problems. Without going into technical details, the following tools have been implemented for overcoming such difficulties:

- An API in C++ to Modelica programs with which sensitivities of ADModelica generated code can be also accessed and simulated in parallel, (cf. appendix E.1)
- Breakable simulators with the help of low-level thread programming
- A dynamic interactive user interface with which large-scale optimization problems can be easily configured (Elsheikh 2013). The resulting user interfaces can be used for supporting parameter estimation as an iterative process (cf. appendix E.2).

Evenworse, such problems are also common for the Modelica community that a lot of individual efforts are performed for Modelica-based computational tasks. Only recently, more standardized protocols have been proposed by the Modelica community for providing a unified interface by which communication with simulation environments and other Modelica-based tool is possible within classical programming languages (Blochwitz et al. 2011). Nevertheless, the implemented high-level API are exactly relevant for the type of tasks needed within this work and can be easily modified for adopting standardized efforts from the community.

D.2. Metaheuristics

Metaheuristics¹ (Glover and Kochenberger 2003, Talbi 2009) are a kind of approximate algorithms that follow an iterative master process for guiding and modifying the exploration of a search space of an optimization problem for finding near optimal solutions. At each iteration, a single solution or a population of solutions are manipulated according to search heuristics (Voss et al. 1999). Metaheuristics can be applied as a black box to optimization problems as they are not problem-specific.

Metaheuristic algorithms can be classified into two basic categories: evolutionary algorithms and local search methods. Local search methods aim at intensifying the search in promising regions while evolutionary algorithms aim at better exploration of the search space by a population of solutions (Holland 1975). At each iteration a population is initialized, selected, paired and recombined in order to generate new solutions that replace other ones and so on. Metaheuristics can be also further characterized according to whether they (Blum and Roli 2003)

- are nature-inspired or non-nature inspired
- are single point or population-based
- have one or various neighborhood structures
- have dynamic or static objective function
- are memory-based or memory-less

Examples of popular metaheuristics are simulated annealing (Metropolis et al. 1953), tabu search (Glover 1986) and genetic algorithms (Holland 1975). Metaheuristics have been basically invented for solving combinatorial optimization problems. However, metaheuristics versions for optimization problems of real-valued search space also exist. These include particle swarm optimization (Kennedy and Eberhart 1995), differential evolution (Storn and Price 1997) and evolution strategies (Bäck et al. 1991). Despite of being widely used, there is no theoretical basis for convergence analysis of metaheuristics, though some mathematical analysis has been presented in literature, e.g. Holland's schema theorem for genetic algorithms (Holland 1975) and (Zaharie 2002) for analysis of differential evolution. For the type of optimization problems handed in this work, metaheuristics are computationally intensive and hence they are not practical for large models.

¹The term metaheuristic is a Greek compound word. The prefix Heuristic comes from *heuriskein* which means "search", while the suffix meta means "beyond"

D.3. Gradient-based optimization algorithms

Given that the considered Modelica model is represented by the parametrized DAE system formally defined by:

$$F(\dot{x}, x, p, t) = 0, \quad x(0) = x_0 \quad (\text{D.1})$$

where the function $F : R^{2N+M+1} \rightarrow R^N$ is sufficiently smooth with respect to the state variables $x \in R^N$ and parameters $p \in R^M$. Typical parameter estimation problems aim at minimizing the distance between simulation results $x(p, t)$ and measurement data $\tilde{x}(t_j) \in R^N$ at discrete time points t_j with $j = 1..T$ in the sense of least squares:

$$\begin{aligned} r : R^M &\rightarrow R, \quad r(p) = \frac{1}{2} \|Q\|_2^2 \\ Q &= [q_1, \dots, q_T] \in R^{N \cdot T} \\ q_j &= \tilde{x}(t_j) - x(p, t_j), \quad j = 1, \dots, T. \end{aligned} \quad (\text{D.2})$$

For start values $p^0 \in R^M$ that are chosen sufficiently close to a local optimum $p_{loc}^* \in R^M$, Gauss-Newton algorithm (Antoniu and Lu 2007, Nocedal and Wright 2006) converges to p_{loc}^* by iterating over the following scheme:

$$\begin{aligned} p^{i+1} &= p^i - (r_{pp})^{-1} r_p, \\ &\approx p^i - \left(\begin{bmatrix} x_p^{i,j} \end{bmatrix}^T x_p^{i,j} \right)^{-1} \begin{bmatrix} x_p^{i,j} \end{bmatrix}^T Q \end{aligned} \quad (\text{D.3})$$

where

$$r_p = \frac{\partial r}{\partial p}(p^i), \quad r_{pp} = \frac{\partial^2 r}{\partial p^2}(p^i)$$

and

$$x_p^{i,j} \in R^{N \cdot S \times M}, \quad x_p^{i,j} = \frac{\partial x}{\partial p}(p^i, t_j) \quad \forall j = 1, \dots, T$$

represent parameter sensitivities at discrete time points $t_j = 1, \dots, T$. The Hessian r_{pp} is usually approximated by $\left(\begin{bmatrix} x_p^{i,j} \end{bmatrix}^T x_p^{i,j} \right)$ which is often semi-singular. In this case the inverse can not be exactly computed but is usually approximated by a pseudo-inverse using singular value decomposition. x_p can be solved by solving the sensitivity equations computed by ADModelica.

Heuristics for Gauss-Newton

The singularity of the Hessian may lead to further negative consequences. The goal function can increase by the suggested Gauss-Newton step, instead of getting decreased. Additionally, the new parameter values may lay outside the convergence region, and consequently causing divergence of Gauss-Newton. The following heuristics can overcome the mentioned difficulties

Appendix D. A Survey of Optimization Algorithms

Line Search Algorithm

Even if the suggested Gauss-Newton step does not minimize the goal function, it is recommended to follow the suggested direction, induced by the Jacobian, though, with a suitable step length. Common line-search algorithms aim at estimating a $\lambda \in (0, 1]$ s.t.

$$S_N^{p^i} = p^{i+1} - p^i = \lambda_i \left(\begin{bmatrix} x_p^{i,j} \end{bmatrix}^T x_p^{i,j} \right)^{-1} \begin{bmatrix} x_p^{i,j} \end{bmatrix}^T Q \quad (\text{D.4})$$

minimizes the goal function.

Levenberg-Marquardt

In order to avoid an extreme sloppily direction of optimization, Levenberg-Marquardt suggests manipulation of the diagonal elements of the approximated Hessian. The algorithm aims at finding $\lambda \in \mathbb{R}$ s.t.

$$S_N^{p^i} = p^{i+1} - p^i = (J_{x_p}^T J_{x_p} + \lambda_i I)^{-1} J_{x_p}^T Q \quad (\text{D.5})$$

minimizes the goal function.

Scaling

Physical units of various variables and parameters are usually different. Moreover, pure magnitudes and values of parameters do not necessarily indicate the respective importance. Unfortunately, although in theory it does not matter which physical units are used, the scales used may influence the singularity of the Hessian, and hence, the behavior of Gauss-Newton. To improve this behavior, dummy physical units can be introduced. This does not only improve the scaling of the Jacobian, but also improves the corresponding convergence area of Gauss-Newton. Results within a multistart derivative-based optimization show that more start points converge with scaled parameters.

Parameter Scaling

Scaling a parameter vector $p \in \mathbb{R}^M$ to $q \in \mathbb{R}^M$ can be realized as follows:

$$q = D_p p, \quad \text{where } D_p \in \mathbb{R}^{M \times M}, D_p = [d_{ij}] = \{1/\max(|p_i|, \epsilon) \text{ if } i = j \text{ else } 0\} \quad (\text{D.6})$$

Here, D_p get changed at each iteration (dynamic scaling) and ϵ serves for not being divided by 0. In this case the scaled objective function (D.2) and their first and second derivatives required by typical optimization software are formulated as follows:

$$\begin{aligned} r(q) &= \frac{1}{2} \|Q((D_p)^{-1} q)\|_2^2 = \frac{1}{2} \|Q(p)\|_2^2 \\ \nabla_q[r(q)] &= (D_p)^{-1} x_p^T Q(p) = (D_p)^{-1} \nabla_p[r(p)] \\ \nabla_q^2[r(q)] &\approx x_p^T (D_p)^{-1} (x_p^T (D_p)^{-1})^T = ((D_p)^{-1})^2 \nabla_p^2[r(p)] \end{aligned} \quad (\text{D.7})$$

and the corresponding Gauss-Newton step becomes

$$S_N^q = (x_p^T(D_p)^{-1}(x_p^T(D_p)^{-1})^T)^{-1}x_p^T(D_p)^{-1}Q(p) \approx D_p S_N^p \quad (\text{D.8})$$

In this work, two ways of parameter scaling have been applied:

1. Dynamic scaling which lets all parameter equal to one
2. Logarithmic scaling for positive parameters vector $p \in \mathbb{R}^M$, by letting $q_i = \log(p_i + 1)$ i.e. take the logarithm of each entry added by one.

D.4. Some global optimization algorithms

In the context of this work, prototypes of two further global optimization algorithms have been implemented each has its own way of tackling the complexity of exploring the large search space of the optimization problem.

Cluster algorithms

The first algorithm, illustrated in figure D.1 is based on cluster methods in which a large parameter estimation problem is decomposed into independent smaller clusters (i.e. sub-problems with less number of parameters and variables), each of which requires fewer start values and less computation. The resulting local minima are taken as start values for enlarged subproblems, and so forth until good start values for the original problem are found. This approach serves to improve global convergence and computational speed of multi-start optimization strategies independent of the optimization algorithm used.

Usually, the decomposition into clusters can be heuristically performed based on parameter sensitivities in which strongly sensitive parameters and variables can be clustered together. Another way is to use domain-knowledge to provide a natural decomposition based on topological information of the model. For instance in (Hadlich et al. 2009) start values for estimating large biochemical network models was provided by first performing local parameter estimation for each reaction alone. The locally estimated parameters were used together as start values for the larger optimization problem.

Alternatively (Elsheikh et al. 2009) presents a multistart recursive clustering strategy that utilizes DAE decomposition algorithms, in particular Tarjan's and tearing algorithms, cf. Chapter 3. In this way, a natural decomposition with well-defined dependencies among clusters is provided. In ideal models with little or no cyclic structure, these heuristics is much more efficient than the presented hybrid heuristics in Chapter 17. However, they become more difficult to apply with biochemical networks with typically many effectors. For this reason, they have been excluded from the presented benchmark. Nevertheless, they still remain an ideal choice for initial models suggested by model selection strategies. The estimated parameters of such initial models serve as ideal start values for further complicated models generated through a model selection strategy.

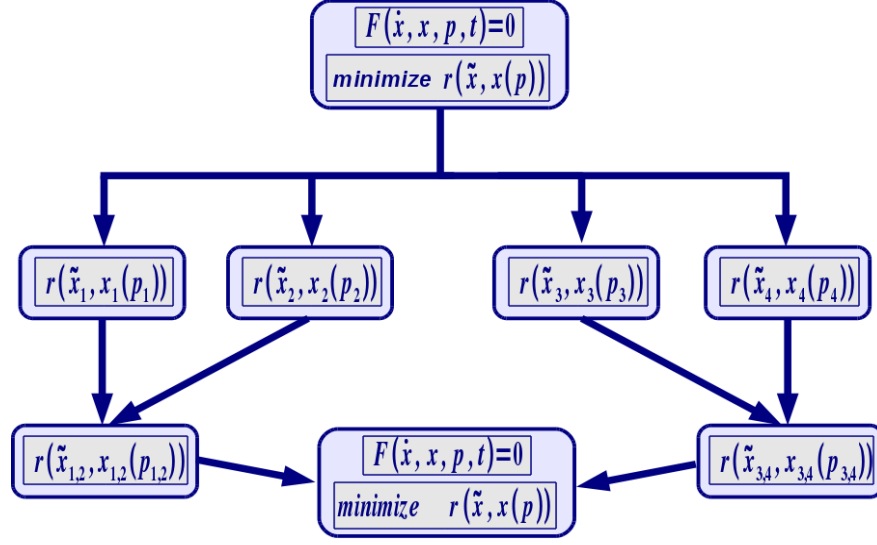


Figure D.1.: General schema of a cluster optimization method

Multiple-shooting

In this work a prototype implementation of multiple-shooting algorithms (Bock and Plitt 1984, Bock et al. 2007) has been implemented. The global convergence area of Gauss-Newton scheme is effectively improved by reducing the nonlinearity of r by solving $x(t, p)$ through the DAE-systems:

$$\begin{aligned} F(\dot{x}, x(y, z), p, t) = 0, \quad y(t_i) = s_{y,i}, \quad z(t_i) = f(s_{y,i}) \\ \text{where } t \in [t_i, t_{i+1}], i = 1, \dots, N_s \text{ and } t_1 = 0 \\ f \text{ guarantees the consistency of initial conditions} \end{aligned} \quad (\text{D.9})$$

In this case, $x(t) = x^i(t, p, s_{y,i})$, $i = 1, \dots, N_s$ and hence extra parameters $s_{y,i}$ are introduced to the minimization problem, i.e. the task becomes to solve

$$\min_{p, s_{y,i}} r(x(t, p, s_{y,i})), \quad i = 1, \dots, N_s \quad (\text{D.10})$$

In the optimal case, measurements corresponding to all y are available and hence suitable initial guesses for $s_{y,i}$ can be provided. Consistent initial conditions for z (i.e. $f(s_{y,i})$) can be automatically computed by the software Dymola. A solution of the above problem physically makes sense if the resulting trajectories $x(t_i, p, s_{y,i})$ are continuous. This can be enforced by attaching the following constraint to the problem (D.10):

$$C = \frac{1}{2} \sum_{i=1}^{N_s-1} \|s_{y,i+1} - x^i(t_{i+1}, p, s_{y,i})\|_2^2 = 0 \quad (\text{D.11})$$

The solution requires the partial derivatives:

$$\begin{aligned}\frac{\partial r}{\partial(p,s)} &= -\Sigma^{-1} \sum_{i=1}^{N_s-1} \left[\frac{\partial x}{\partial(p,s)}(t_{i+1}) \right]^T \sqrt{r}, \\ \frac{\partial C}{\partial(p,s)} &= \sum_{i=1}^{N_s-1} \left[[0|I] - \frac{\partial x}{\partial(p,s)}(t_{i+1}) \right]^T \sqrt{C}\end{aligned}\tag{D.12}$$

where

$$\frac{\partial x}{\partial(p,s)} = \begin{pmatrix} \frac{\partial x^1}{\partial p} & \frac{\partial x^1}{\partial s_{y,1}} & 0 & \cdots & 0 \\ \frac{\partial x^2}{\partial p} & 0 & \frac{\partial x^2}{\partial s_{y,2}} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial x^{N_s}}{\partial p} & 0 & 0 & \cdots & \frac{\partial x^{N_s}}{\partial s_{y,N_s}} \end{pmatrix}\tag{D.13}$$

and $\sqrt{r} = \sum_{i=1}^{N_s-1} [s_{y,i+1} - x^i(t_{i+1}, p, s_{y,i})]$, \sqrt{C} similarly defined

These partial derivatives are directly computed via ADModelica by letting start values of state variables be explicitly declared as parameters. First implementation is done using nonlinear constrained solver from the Matlab optimization toolbox. Figure D.2 shows the trajectories of a multiple-shooting goal function before and after the estimation. Limitations with respect to constraint satisfaction arises with inexact data. Nevertheless, the resulting optimum can be directly used as a start value for a single shooting optimization. The main drawback of the implementation is the efficiency of the algorithm. In order to make multiple-shooting as fast as single-shooting, sparse techniques need to be employed rather than the standard implementation.

Appendix D. A Survey of Optimization Algorithms

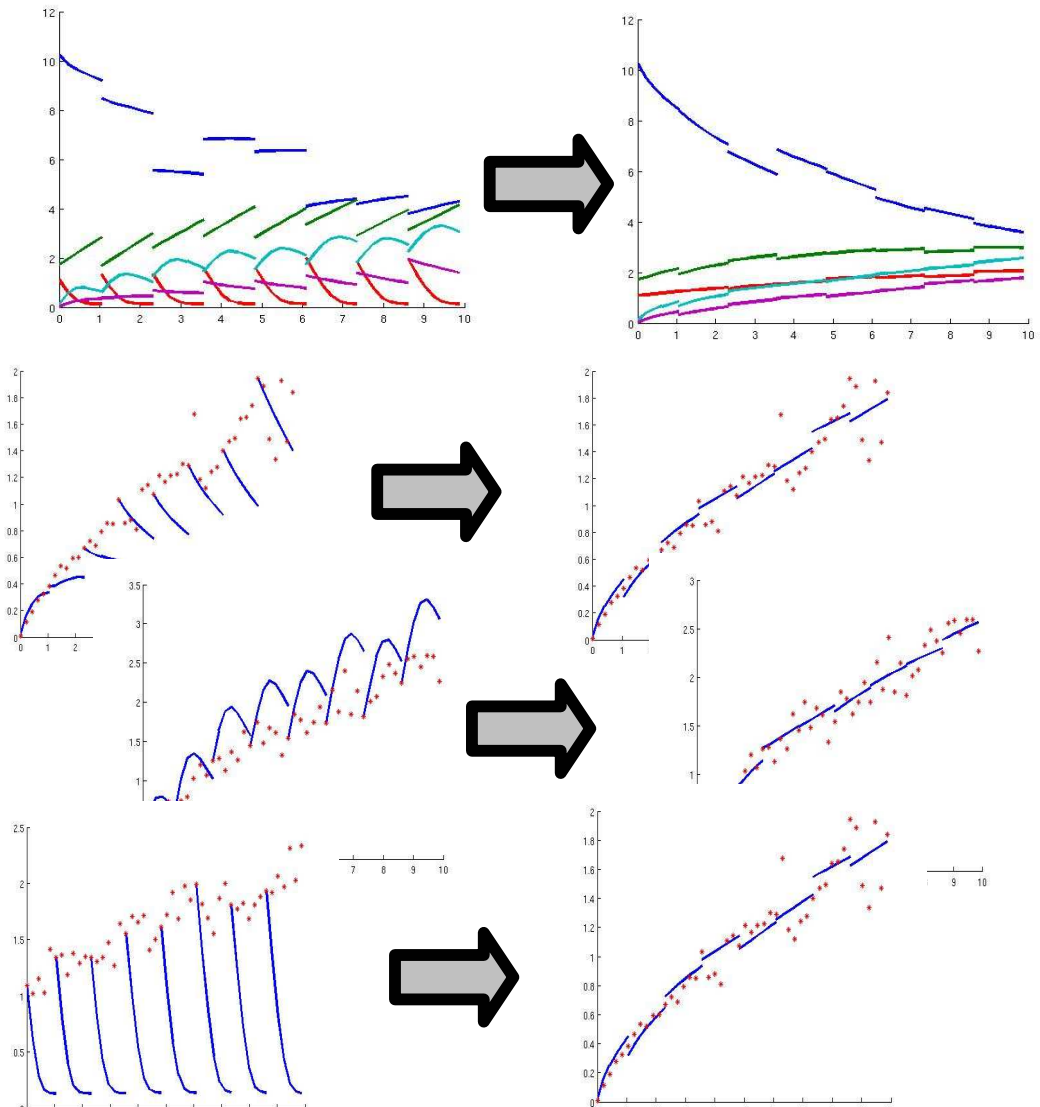


Figure D.2.: The trajectories of state variables before and after optimization

Appendix E.

Implemented Software

E.1. C++ software for communicating with Dymola

In order to link Modelica models with external optimization software (e.g. IPOPT, LEVMAR, NLOPT, PARADISEO etc.), the process of modifying Modelica models (e.g. with other parameter values) needs to be automated. A software fulfilling this requirement is implemented in a way that a client is able to perform the following tasks for anonymous Modelica models through a well-defined API:

- assign default values or control simulation parameters (e.g. start time, stop time, tolerance, solver used, the time points at which solution is desired etc.)
- set model parameters and start values
- extract the results of required simulation variables
- generate derivatives w.r.t. desired parameters
- extract parameter sensitivities (including sensitivities w.r.t. start values)

With the help of the mentioned software, further classes expressing Modelica-based goal functions (eg. weighted sum of squares) were implemented. Optimization with the software LEVMAR (a Levenberg-Marquardt implementation in C) was successfully applied.

Listing E.1: A sample of the C++/Dymola Inteface

```
dym.compileGradient({"p1" ; "p2"});  
dym.setParameter("p1",10.0);  
dym.setStopTime(5.0);  
dym.simulate({"v1"; "v2"; "v3"});  
X = dym.getData();  
gx = dym.getGradData();
```

Listing E.2: A sample of the C++/Dymola Inteface

```
ParDymSim dym({"p1","p2"},{"x1","x2","x3"});  
dym.setParameter("p1",10.0);  
dym.setStopTime(5.0);  
dym.simulate();  
x = dym.getData();
```

Appendix E. Implemented Software

```
// Gradient model must be already compiled
GradDymSim gdym({"p1", "p2"}, {"x1", "x2", "x3"});
gdym.setParameter("p1", 10.0);
gdym.setStopTime(5.0);
gdym.simulate();
X = gdym.getData();
gx = gdym.getGradData();
```

As a result, software with well-defined API for setting up, compiling, simulating and extracting results of Modelica models compiled by Dymola has been implemented. Modelica models for describing the dynamics of a TCA-cycle and isotopic Coryne-based models were used as test benchmarks. First simulations on Juropa were done. Software for parameter estimation of Modelica model was used with an open-source implementation of Levenberg-Marquardt method (LEVMar).

E.2. Interactive dynamic menus

A dynamically interactive user interface for configuring the optimization framework based on decision trees concept has been implemented. The easily extendable user interface assists handling large optimization problems (eg. parameter estimation with large number of parameters) by supporting:

- Automatic selection of variables and parameters of large models consisting of 100s of parameters and variables.
- Incremental in-silico investigation by adding/fixing parameters or reinvolving/removing measurements of an optimization to the next.
- Detecting the most optimal problem-dependent settings of the optimization-software since optimization results are very sensitive to these settings (eg. stopping conditions, different heuristical strategies).
- Specifying box-constraints of the boundaries of start-values / optimization variables.
- Easily extending further configuration aspects, eg. when new optimization problems, goal functions, algorithms, software are considered.

The platform is used to assist the quantification of the influence of measurement errors on parameter identifiability. Comparison of the quality of kinetic parameter estimation with different:

1. Norms or quality functions
2. Optimization software/optimization algorithm settings s.a. parameter scaling strategies
3. Set of parameters and reference variables (eg. Measurements)

4. values of fixed parameters, start values, ranges of start values, standard deviation of data samples etc.

Contributions of the Interactive Dynamic User Interface

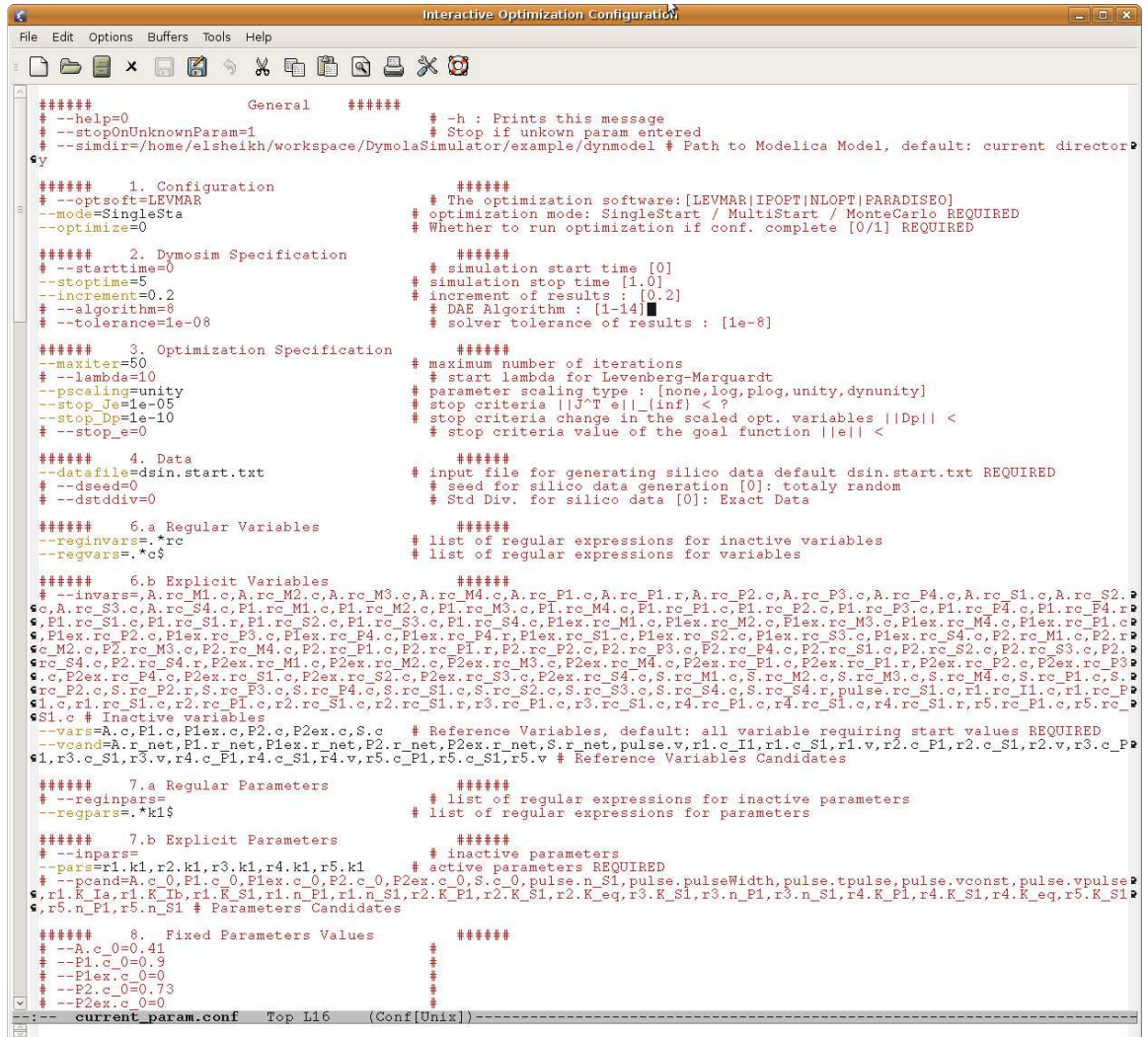
With the help of the above criteria: the following tasks can be easily investigated:

- Comparison of the optimization results of the dynamics of a metabolic network with/without marking carbons using the same configuration files.
- Detection of a functioning configuration of the optimization problem under consideration with respect to optimal parameter set, optimization settings ..etc. For manual setting, this is a very tedious / time-consuming task.
- Flexible switching between different configurations of the same problem, eg. active parameters, reference variables, goal functions, optimization software, boundary specification and Dymola-specific configuration, DAE-solver issues etc.

From an optimization algorithm perspective, the following have been concluded with the help of the established platform:

- Parameter scaling is a fundamental issue (independent of the optimization software used). Parameter scaling is done on gateway interface between user input model and the optimization software. Without parameter scaling, no stable results could be achieved with the used software so far.
- Parameter estimation results are very sensitive to stopping conditions. False settings lead to unstable results. Detecting the optimal setting manually is tedious and time consuming.

Appendix E. Implemented Software



```

##### General #####
# --help=0 # -h : Prints this message
# --stopOnUnknownParam=1 # Stop if unknown param entered
# --simdir=/home/elsheikh/workspace/DymolaSimulator/example/dynmodel # Path to Modelica Model, default: current directory

##### 1. Configuration #####
# --optsoft=LEVMAR # The optimization software: [LEVMAR|IPOPT|NLOPT|PARADISEO]
# --mode=SingleStart # optimization mode: SingleStart / MultiStart / MonteCarlo REQUIRED
# --optimize=0 # Whether to run optimization if conf. complete [0/1] REQUIRED

##### 2. Dymosim Specification #####
# --starttime=0 # simulation start time [0]
# --stoptime=5 # simulation stop time [1.0]
# --increment=0.2 # increment of results : [0.2]
# --algorithm=8 # DAE Algorithm : [1-14]
# --tolerance=1e-08 # solver tolerance of results : [1e-8]

##### 3. Optimization Specification #####
# --maxiter=50 # maximum number of iterations
# --lambda=1 # start lambda for Levenberg-Marquardt
# --pscaling=unity # parameter scaling type : [none, log, plog, unity, dynunity]
# --stop_d=1e-05 # stop criteria ||J'|| || [inf] < ?
# --stop_Dp=1e-10 # stop criteria change in the scaled opt. variables ||Dp|| <
# --stop_e=0 # stop criteria value of the goal function ||e|| <

##### 4. Data #####
# --datafile=dsin.start.txt # input file for generating silico data default dsin.start.txt REQUIRED
# --dseed=0 # seed for silico data generation [0]: totally random
# --stddiv=0 # Std Div. for silico data [0]: Exact Data

##### 6.a Regular Variables #####
# --reginvars=.*rc # list of regular expressions for inactive variables
# --regvars=.*c$ # list of regular expressions for variables

##### 6.b Explicit Variables #####
# --invars=A.rc.M1.c,A.rc.M2.c,A.rc.M3.c,A.rc.M4.c,A.rc.P1.c,A.rc.P1.r,A.rc.P2.c,A.rc.P3.c,A.rc.P4.c,A.rc.S1.c,A.rc.S2.c,A.rc.S3.c,A.rc.S4.c,P1.rc.M1.c,P1.rc.M2.c,P1.rc.M3.c,P1.rc.M4.c,P1.rc.P1.c,P1.rc.P2.c,P1.rc.P3.c,P1.rc.P4.c,P1.rc.S1.c,P1.rc.S2.c,P1.rc.S3.c,P1.rc.S4.c,Plex.rc.M1.c,Plex.rc.M2.c,Plex.rc.M3.c,Plex.rc.M4.c,Plex.rc.P1.c,Plex.rc.P2.c,Plex.rc.P3.c,Plex.rc.P4.c,Plex.rc.P1.r,Plex.rc.P2.r,Plex.rc.P3.r,Plex.rc.P4.r,Plex.rc.S1.c,Plex.rc.S2.c,Plex.rc.S3.c,Plex.rc.S4.c,P2.rc.M1.c,P2.rc.M2.c,P2.rc.M3.c,P2.rc.M4.c,P2.rc.P1.c,P2.rc.P2.c,P2.rc.P3.c,P2.rc.P4.c,P2.rc.S1.c,P2.rc.S2.c,P2.rc.S3.c,P2.rc.S4.c,P2.rc.S1.r,P2.rc.S2.r,P2.rc.S3.r,P2.rc.S4.r,P2ex.rc.M1.c,P2ex.rc.M2.c,P2ex.rc.M3.c,P2ex.rc.M4.c,P2ex.rc.P1.c,P2ex.rc.P1.r,P2ex.rc.P2.c,P2ex.rc.P3.c,P2ex.rc.P4.c,P2ex.rc.S1.c,P2ex.rc.S2.c,P2ex.rc.S3.c,P2ex.rc.S4.c,S.rc.M1.c,S.rc.M2.c,S.rc.M3.c,S.rc.M4.c,S.rc.P1.c,S.rc.P2.c,S.rc.P3.c,S.rc.P4.c,S.rc.S1.c,S.rc.S2.c,S.rc.S3.c,S.rc.S4.c,S.rc.S1.r,S.rc.S2.r,S.rc.S3.r,S.rc.S4.r,pulse.rc.S1.c,r1.rc.II.c,r1.rc.II.r,r1.rc.II.v,r1.rc.II.v,r2.rc.S1.c,r2.rc.S1.v,r2.rc.S1.v,r2.rc.S1.v,r3.rc.S1.c,r3.rc.S1.v,r3.rc.S1.v,r3.rc.S1.v,r4.rc.S1.c,r4.rc.S1.v,r4.rc.S1.v,r4.rc.S1.v,r5.rc.S1.c,r5.rc.S1.v,r5.rc.S1.v,r5.rc.S1.v # Reference Variables, default: all variable requiring start values REQUIRED
# --vrand=A.rc.net,P1.rc.net,Plex.rc.net,P2.rc.net,P2ex.rc.net,S.rc.net,pulse.v,r1.c.II,r1.c.S1,r1.v,r2.c.P1,r2.c.S1,r2.v,r3.c.P1,r3.c.S1,r3.v,r4.c.P1,r4.c.S1,r4.v,r5.c.P1,r5.c.S1,r5.v # Reference Variables Candidates

##### 7.a Regular Parameters #####
# --reginpars=.*k1$ # list of regular expressions for inactive parameters
# --regpars=.*k1$ # list of regular expressions for parameters

##### 7.b Explicit Parameters #####
# --inpars= # inactive parameters
# --pars=r1.k1,r2.k1,r3.k1,r4.k1,r5.k1 # active parameters REQUIRED
# --pcand=A.c_0,P1.c_0,Plex.c_0,P2.c_0,P2ex.c_0,S.c_0,pulse.n_S1,pulse.pulseWidth,pulse.pulseWidth,pulse.vconst,pulse.vpulse,r1.K_Ia,r1.K_Ib,r1.K_S1,r1.n_P1,r1.n_P1,r2.K_P1,r2.K_S1,r2.K_eq,r3.K_S1,r3.n_P1,r3.n_S1,r4.K_P1,r4.K_S1,r4.K_eq,r5.K_S1,r5.n_P1,r5.n_S1 # Parameters Candidates

##### 8. Fixed Parameters Values #####
# --A.c_0=0.41 #
# --P1.c_0=0.9 #
# --Plex.c_0=0.73 #
# --P2.c_0=0 #
# --P2ex.c_0=0 #

```

Figure E.1.: An interactive user interface for configuring large Modelica-based optimization problems. It is divided into dependent dynamic menus (i.e. the existence of a menu depends on the value of another). A menu consists again of dependent dynamic flags.

Bibliography

- Akaike, H. 1980. Likelihood and the bayes procedure. In J. M. Bernardo, et al. (Eds.) *Bayesian statistics*. Valencia University Press.
- Alberts, B., Johnson, A., Lewis, J., Raff, M., Roberts, K., and Walter, P. 2002. *Molecular Biology of the Cell*. Garland Science.
- Andersson, C., Åkesson, J., Führera, C., and Gäfvert, M. 2011. Import and export of functional mock-up units in JModelica.org,. In *Modelica'2011: The 8th International Modelica Conference*. Dresden, Germany.
- Andersson, J., Houska, B., and Diehl, M. 2010. Towards a computer algebra system with automatic differentiation for use with object-oriented modelling languages. In *Proceedings of 3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*.
- Antonioni, A., and Lu, W.-S. 2007. *Practical Optimization - Algorithms and Engineering Applications*. Springer.
- Astroem, K. J., Elmqvist, H., and Mattsson, S. E. 1998. Evolution of continuous-time modeling and simulation. In *ESM 1998: The 12th European Simulation Multiconference - Simulation - Past, Present and Future*. Manchester, United Kingdom.
- Atherton, R. W., Schainker, R. B., and Ducot, E. 1975. On the statistical sensitivity analysis of models for chemical kinetics. *AIChE Journal*, 21(3), 441–448.
- Bastian, J., Clauß, C., Wolf, S., and Schneider, P. 2011. Master for co-simulation using FMI. In *Modelica'2011: The 8th International Modelica Conference*. Dresden, Germany.
- Bäck, T., Hoffmeister, F., and Schwefel, H.-P. 1991. A survey of evolution strategies. In *ICGA 1991: The 4th International Conference on Genetic Algorithms*. San Diego, CA, USA: Morgan Kaufmann.
- Bischof, C. H., Bücker, H. M., Marquardt, W., Petera, M., and Wyes, J. 2005. Transforming equation-based models in process engineering. In H. M. Bücker, G. Corliss, P. Hovland, U. Naumann, and B. Norris (Eds.) *Automatic Differentiation: Applications, Theory, and Implementations*, Lect. Notes in Comp. Sc. and Eng., (pp. 189–198). Springer.
- Bisswanger, H. 2002. *Enzyme Kinetics, Principle and Methods*. WILEY-VCH Verlag, Weinheim, Germany.

Bibliography

- Blochwitz, T., Otter, M., Arnold, M., Bausch, C., C. Clauß, H. E., Junghanns, A., Mauss, J., Monteiro, M., Neidhold, T., Neumerkel, D., Olsson, H., Peetz, J.-V., and Wolf, S. 2011. The functional mockup interface for tool independent exchange of simulation models. In *Modelica'2011: The 8th International Modelica Conference*. Dresden, Germany.
- Blum, C., and Roli, A. 2003. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3), 268–308.
- Bock, H., Kostina, E., and Schlöder, J. 2007. Numerical methods for parameter estimation in nonlinear differential algebraic equations. *GAMM Mitteilungen*, 30(2), 376–408.
- Bock, H., and Plitt, K. 1984. A multiple shooting algorithm for direct solution of optimal control problems. In *Proceedings of the 9th IFAC World Congress*. Budapest, Hungary.
- Boender, C. G. E., and Romeijn, H. E. 1995. *Stochastic methods*. Dordrecht: Kluwer.
- Braun, W., and Bachmann, L. O. B. 2011. Symbolically derived jacobians using automatic differentiation - enhancement of the openmodelica compiler. In *Proceeding of the 8th International Modelica Conference*. Dresden, Germany.
- Brenan, K. E., Campbell, S. L., and Petzold, L. R. 1989. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. New York, USA: North-Holland.
- Brown, P. N., and Hindmarsh, A. C. 1986. Reduced storage matrix methods in stiff ODE systems. *Applied Mathematics and Computation*, 31, 40–91.
- Campbell, S. L. 1994. Numerical methods for unstructured higher index DAEs. *Annals of Numerical Mathematics*, 1, 265–278.
- Caracotsios, M., and Stewart, W. E. 1985. Sensitivity analysis of initial value problems with mixed ODEs and algebraic equations. *Computers and Chemical Engineering*, 9(4), 359–365.
- Cellier, F. E. 1991. *Continuous System Modeling*. Springer Verlag.
- Claeys, F., Vanrolleghem, P., and Fritzson, P. 2006. A generalized framework for abstraction and dynamic loading of numerical solvers. In *EMSS2006: The 2nd European Modelling and Simulation Symposium*. Barcelona, Spain.
- Cohen, S. D., and Hindmarsh, A. C. 1996. CVODE, a stiff/nonstiff ODE solver in C. *Computer in Physics*, 10(2), 138–143.
- De Pauw, D. J., and Vanrolleghem, P. A. 2003. Practical aspects of sensitivity analysis for dynamic models. In *MATHMOD 2003: The 4th Vienna International Conference on Mathematical Modelling*. Vienna, Austria.

- Dickinson, R., and Gelinas, R. 1976. Sensitivity analysis of ordinary differential equation systems - a direct method. *Journal of Computational Physics*, 21, 123–143.
- Droste, P., Noack, S., Noh, K., and Wiechert, W. 2009. Customizable visualization of multi-omics data in the context of biochemical networks. In *VIZ 2009: The 2nd International Conference on Information Visualisation*. Barcelona, Spain.
- Eberly, D. 2001. Derivative approximation by finite differences. Tech. rep., Geometric tools, LLC.
- Efron, B. 1979. Bootstrap methods: another look at the jackknife. *Annals of Statistics*, 7(1), 1–26.
- Ehlde, M., and Zacchi, G. 1996. Influence of experimental errors on the determination of flux control coefficients from transient metabolite concentrations. *Biochemical Journal*, 313(3), 721–727.
- Ellson, J., Gansner, E. R., Koutsofios, E., North, S. C., and Woodhull, G. 2003. Graphviz and Dynagraph – static and dynamic graph drawing tools.
- Elmqvist, H. 1978. *A structured model language for large continuous systems*. Ph.D. thesis, Lund Institute of Technology, Lund, Sweden.
- Elmqvist, H. 1993. Object-oriented modeling and automatic formula manipulation in Dymola. In *SIMS 1993: The 34th Scandinavian Conference on Simulation and Modeling*. Kongsberg, Norway.
- Elmqvist, H., and Mattsson, S. E. 1997. Modelica - the next generation modeling language: An international design effort. In *ESS97: The 9th European Simulation Symposium*. Passau, Germany.
- Elmqvist, H., and Otter, M. 1994. Methods for tearing systems of equations in object-oriented modeling. In *ESM'94: The 8th European Simulation Multiconference - Simulation - Past, Present and Future*. Barcelona, Spain.
- Elsheikh, A. 2012. ADGenKinetics: An algorithmically differentiated library for biochemical networks modeling via simplified kinetics formats. In *Modelica'2012, the 9th International Modelica Conference*. Munich, Germany.
- Elsheikh, A. 2013. Assisting identifiability analysis of large-scale dynamical models with decision trees: DecTrees and InteractiveMenus. In *The 8th EUROSIM Congress on Modelling and Simulation*. Cardiff, Wales, UK.
- Elsheikh, A. 201xa. Derivative-based hybrid heuristics for continuous-time simulation optimization. Submitted.
- Elsheikh, A. 201xb. Modeling control coefficients in ADGenKinetics via equation-based algorithmic differentiation techniques. Submitted.

Bibliography

- Elsheikh, A. 201xc. A novel equation-based algorithmic differentiation technique for sensitivity analysis of differential algebraic equations. Submitted.
- Elsheikh, A., Nöh, K., and von Lieres, E. 2009. Improving convergence of derivative-based parameter estimation with multistart parameter clustering based on DAE decomposition. In *Modelica'2009: The 7th International Modelica Conference*. Como, Italy.
- Elsheikh, A., Noack, S., and Wiechert, W. 2008. Sensitivity analysis of Modelica applications via automatic differentiation. In *Modelica'2008: The 6th International Modelica Conference*. Bielefeld, Germany.
- Elsheikh, A., Widl, E., and Palensky, P. 2012. Simulating complex energy systems with Modelica: A primary evaluation. In *DEST'2012, the 6th IEEE International Conference on Digital Ecosystems and Technologies*. Campione d'Italia, Italy.
- Elsheikh, A., and Wiechert, W. 2008. Automatic sensitivity analysis of DAE-systems generated from equation-based modeling languages. In C. H. Bischof, H. M. Bücker, P. D. Hovland, U. Naumann, and J. Utke (Eds.) *Advances in Automatic Differentiation*, (pp. 235–246). Springer.
- Elsheikh, A., and Wiechert, W. 2012. Accuracy of parameter sensitivities of DAE systems using finite difference methods. In *MATHMOD'2012, The 7th Vienna International Conference on Mathematical Modelling*. Vienna, Austria.
- Elsheikh, A., and Wiechert, W. 201x. A structure-preserving approach for sensitivity analysis of higher index differential algebraic equations. Submitted.
- Feehery, W. F., Tolsma, J. E., and Barton, P. I. 1997. Efficient sensitivity analysis of large-scale differential-algebraic systems. *Applied Numerical Mathematics*, 25(1), 41–54.
- Fell, D. 1992. Metabolic control analysis: a survey of its theoretical and experimental development. *Biochemocal Journal*, 286(2), 313–330.
- Frenkel, J., Kunze, G., and Fritzson, P. 2012. Survey of appropriate matching algorithms for large scale systems of differential algebraic equations. In *Modelica'2012, the 9th International Modelica Conference*. Munich, Germany.
- Frenkel, J., Schubert, C., Kunze, G., Fritzson, P., Sjölund, M., and Pop, A. 2011. Towards a benchmark suite for Modelica compilers: Large models. In *Modelica'2011: The 8th International Modelica Conference*. Dresden, Germany.
- Fritzson, P. 2003. *Principles of Object-Oriented Modeling and Simulation with Modelica*. Wiley-IEEE Computer Society Pr.
- Fritzson, P., Aronsson, P., Bunus, P., Engelson, V., Saldami, L., Johansson, H., and Karström, A. 2002a. The Open Source Modelica Project. In *Proceedings of the 2nd International Modelica Conference*. Munich, Germany.

- Fritzson, P., Ounnarsson, J., and Jirstr, M. 2002b. MathModelica - An extensible modeling and simulation environment with integrated graphics and literate programming. In *Proceeding of the 2nd International Modelica Conference*. Munich, Germany.
- Gear, C. W. 1971. Simultaneous numerical solution of differential algebraic equations. *IEEE Transactions on Circuit Theory*, 18(1), 89–95.
- Glover, F. 1986. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5), 533–549.
- Glover, F., and Kochenberger, G. (Eds.) 2003. *Handbook of Metaheuristics*. International Series in Operations Research and Management Science. Springer.
- Griewank, A., and Walther, A. 2008. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. No. 105 in Other titles in Applied Mathematics. Philadelphia, PA: SIAM, 2nd ed.
- Hadlich, F., Noack, S., and Wiechert, W. 2009. Translating biochemical network models between different kinetic formats. *Metabolic Engineering*, 11(2), 87 – 100.
- Hartwich, A., Stockmann, K., Terboven, C., Feuerriegel, S., and Marquardt, W. 2011. Parallel sensitivity analysis for efficient large-scale dynamic optimization. *Optimization and Engineering*, 12(4), 489 – 508.
- Haunschild, M. D. 2006. *Metabolische Stimulus-Response-Experimente: Werkzeuge zur Modellierung, Simulation und Auswertung*. Ph.D. thesis, Siegen University, Germany.
- Haunschild, M. D., Wahl, S. A., Freisleben, B., and Wiechert, W. 2006. A general framework for large-scale model selection. *Optimization Methods and Software*, 21(6), 901–917.
- Heijnen, J. J. 2005. Approximative kinetic formats used in metabolic network modeling. *Biotechnology and Bioengineering*, 91(5), 534–545.
- Heinrich, R., and Rapoport, T. A. 1974. A linear steady-state theory of enzymatic chains: general properties, control and effector strength. *European Journal of Biochemistry*, 42(1), 89–95.
- Hindmarsh, A. C. 1983. ODEPACK, a systematized collection of ODE solvers. *IMACS Transactions on Scientific Computation*, 1, 55–64.
- Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems*. Ann Arbor, Michigan: The University of Michigan Press.
- Hucka, M., and et al 2003. The Systems Biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4), 524–531.

Bibliography

- Imsland, L., Kittilsen, P., and Schei, T. S. 2009. Using modelica models in real time dynamic optimization – gradient computation. In *Proceeding of the 7th International Modelica Conference*. Como, Italy.
- Johnson, M. L. 1992. Why, when and how biochemists should use least squares. *Analytical Biochemistry*, 206(2), 215 – 225.
- Kacser, H., and Burns, J. A. 1973. The control of flux. *Symposia of the Society for Experimental Biology*, 27, 65–104.
- Ke, X. 2009. *Tools for Sensitivity Analysis of Modelica Models*. Master’s thesis, Siegen University, Germany.
- Kennedy, J., and Eberhart, R. 1995. Particle swarm optimization. In *ICNN95: IEEE International Conference on Neural Networks*. Perth, Western Australia.
- Kitano, H. 2002. Systems Biology: A Brief Overview. *Science*, 295(5560), 1662–1664.
- Klipp, E., Herwig, R., Kowald, A., Wierling, C., and Lehrach, H. 2005. *Systems Biology in Practice: Concepts, Implementation and Application*. Wiley-VCH.
- Kron, G. 1963. *Diaoptics: The Piecewise Solution of Large-Scale Systems*. MacDonald.
- Leitold, A., and Hangos, K. M. 2001. Structural solvability analysis of dynamic process models. *Computers and Chemical Engineering*, 25, 1633–1646.
- Levy, H., and Lessman, F. 1992. *Finite Difference Equations*. Dover Publications.
- Leyffer, S. 2001. Integrating SQP and branch-and-bound for mixed integer nonlinear programming. *Computational Optimization and Applications*, 18, 295–309.
- Li, S., Petzold, L., and Zhu, W. 2000. Sensitivity analysis of differential-algebraic equations: A comparison of methods on a special problem. *Applied Numerical Mathematics: Transactions of IMACS*, 32(2), 161–174.
- Liebermeister, W., and Klipp, E. 2006. Bringing metabolic networks to life: convenience rate law and thermodynamic constraints. *Theoretical Biology and Medical Modelling*.
- Lourakis, M. Jul. 2004. LEVMAR: Levenberg-Marquardt nonlinear least squares algorithms in C/C++. [web page] <http://www.ics.forth.gr/~lourakis/levmar/>.
- Maffezzoni, C., Girelli, R., and Lluka, P. 1996. Generating efficient computational procedures from declarative models. *Simulation Practice and Theory*, 4(5), 303–317.
- Maly, T., and Petzold, L. R. 1995. Numerical methods and software for sensitivity analysis of differential-algebraic systems. *Applied Numerical Mathematics*, 20, 57–79.
- Marsili-Libelli, S., Guerrizio, S., and Checchi, N. 2003. Confidence regions of estimated parameters for ecological systems. *Ecological Modelling*, 165(2-3), 127 – 146.

- Mattsson, S. E., and Elmqvist, H. 1997. Modelica - an international effort to design the next generation modeling language. In *CACSD'97: The 7th IFAC Symposium on Computer Aided Control Systems Design*. Gent, Belgium.
- Mattsson, S. E., and Söderlind, G. 1993. Index reduction in differential-algebraic equations using dummy derivatives. *SIAM Journal on Scientific Computing*, 14(3), 677–692.
- Mauch, K., Arnold, S., and Reuss, M. 1997. Dynamic sensitivity analysis for metabolic systems. *Chemical Engineering Science*, 52(15), 2589 – 2598. Mathematical modelling of chemical and biochemical processes.
- Mendes, P. 2001. Modeling large scale biological systems from functional genomic data: parameter estimation. (pp. 163–186). Cambridge, MA: MIT Press.
- Mendes, P., and Kell, D. 1998. Non-linear optimization of biochemical pathways: applications to metabolic engineering and parameter estimation. *Bioinformatics*, 14(10), 869–883.
- Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., and Teller, A. H. 1953. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6), 1087–1092.
- Michalik, C., Hannemann, R., and Marquardt, W. 2009. Incremental single shooting—a robust method for the estimation of parameters in dynamical systems. *Computers and Chemical Engineering*, 33(7), 1298 – 1305.
- Moles, C. G., Mendes, P., and Banga, J. R. 2003. Parameter estimation in biochemical pathways: A comparison of global optimization methods. *Genome Research*, 13(11), 2467–2474.
- Nagel, L. W., and Pederson, D. 1973. SPICE (simulation program with integrated circuit emphasis). Tech. rep., EECS Department, University of California, Berkeley.
- Naumann, U. 2012. *The art of Differentiating Computer Programs, an Introduction to Algorithmic Differentiation*. SIAM.
- Naumann, U., Utke, J., and Walther, A. 2004. An introduction to developing and using software tools for automatic differentiation. In P. Neittaanmäki, T. Rossi, S. Korotov, E. O. nate, J. Périaux, and D. Knörzer (Eds.) *ECCOMAS 2004: The 4th European Congress on Computational Methods in Applied Sciences and Engineering*. Jyväskylä, Finland: University of Jyväskylä.
- Nilsson, E. L., and Fritzson, P. 2005. A Metabolic Specialization of a General Purpose Modelica Library for Biological and Biochemical Systems. In *Proceeding of the 4th International Modelica Conference*. Hamburg, Germany.
- Noack, S. 2009. *Integrative Auswertung von Multi-Omix-Daten aus dem Zentralstoffwechsel von Corynebacterium Glutamicum*. Ph.D. thesis, Research Centre Jülich, Germany.

Bibliography

- Noack, S., Wahl, A., Haunschild, M. D., Qeli, E., Freisleben, B., and Wiechert, W. 2008. Visualizing regulatory interdependencies and parameter sensitivities in biochemical network models. *Mathematics and Computers in Simulation*, 79(4), 991–998.
- Nocedal, J., and Wright, S. J. 2006. *Numerical Optimization*. Springer series in operations research. Springer Verlag.
- Nöh, K., and Wiechert, W. 2004. Parallel solution of cascaded ODE systems applied to 13C-labeling experiments. In M. Bubak, G. D. van Albada, and P. M. A. Sloot (Eds.) *Lecture Notes in Computer Science*, vol. 3037, (pp. 594–597).
- Nytsch-Geusen, C., and et al. 2005. MOSILAB: Development of a Modelica based generic simulation tool supporting model structural dynamics. In *Proceeding of the 4th International Modelica Conference*. Hamburg, Germany.
- Oldiges, M., and Takors, R. 2005. Applying metabolic profiling techniques for stimulus-response experiments: Chances and pitfalls. In U. Kragl (Ed.) *Technology Transfer in Biotechnology*, Advances in Biochemical Engineering/Biotechnology. Springer Berlin / Heidelberg, Germany.
- Olsson, H., Tummescheit, H., and Elmqvist, H. 2005. Using automatic differentiation for partial derivatives of functions in Modelica. In *Proceeding of the 4th International Modelica Conference*. Hamburg, Germany.
- Pantelides, C. C. 1988. The consistent initialization of differential-algebraic systems. *SIAM Journal on Scientific and Statistical Computing*, 9(2), 213–231.
- Pop, A., and Fritzson, P. 2003. ModelicaXML: A Modelica XML representation with applications. In *Proceeding of the 3rd International Modelica Conference*. Linköping, Sweden.
- Qeli, E., Wiechert, W., and Freisleben, B. 2004. Visualizing time-varying matrices using multidimensional scaling and reorderable matrices. In *IV 2004: The 8th International Conference on Information Visualisation*. London, UK.
- Qeli, E., Wiechert, W., and Freisleben, B. 2005. Visual exploration of time-varying matrices. In *IV 2005: The 9th International Conference on Information Visualisation*. London, UK.
- Reissig, G., Martinson, W. S., and Barton, P. I. 1999. Differential-algebraic equations of index one may have an arbitrarily high structural index. *SIAM Journal on Scientific Computing*, 21(6), 1987–1990.
- Rodriguez-Fernandez, M., Egea, J. A., and Banga, J. R. 2006. Novel metaheuristic for parameter estimation in nonlinear dynamic biological systems. *BMC Bioinformatics*, 7:483.

- Saad, Y. 2003. *Iterative Methods for Sparse Linear Systems*. Philadelphia, Pennsylvania, USA: SIAM Press.
- Saltelli, A., Tarantola, S., Campolongo, F., and Ratto, M. 2004. *Sensitivity Analysis in Practice: A Guide to Assessing Scientific Models*. New York, NY, USA: Halsted Press.
- Schweissgut, O., and Wiechert, W. 2010. Global exploration of optimization landscapes for nonlinear ill posed parameter estimation problems. In *EUROSIM 2010: The 7th European Congress on Modeling and Simulation*. Prague, Czech Republic.
- Sjölund, M., and Fritzson, P. 2009. An openmodelica java external function interface supporting metaprogramming. In *Proceeding of the 7th International Modelica Conference*. Como, Italy.
- Stephanopoulos, G. N., Aristidou, A. A., and Nielsen, J. 1998. *Metabolic Engineering : Principles and Methodologies*. Academic Press.
- Storn, R., and Price, K. 1997. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4), 341–359.
- Talbi, E.-G. 2002. A taxonomy of hybrid metaheuristics. *Journal of Heuristics*, 8(5), 541–564.
- Talbi, E.-G. 2009. *Metaheuristics - From Design to Implementation*. Wiley.
- Tarjan, R. 1972. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2), 146–160.
- Tillack, J. 2008. *Kinetische Modellierung und Simulation isotopisch instationärer metabolischer Netzwerke*. Diploma thesis, Research Centre Jülich, Germany.
- Tillack, J., Noack, S., Nöh, K., Elsheikh, A., and Wiechert, W. 2009. A software framework for modeling and simulation of dynamic metabolic and isotopic systems. In *MATHMOD*. Vienna, Austria.
- Tiller, M. 2001. *Introduction to Physical Modeling with Modelica*. Norwell, MA, USA: Kluwer Academic Publishers.
- Tjoa, I. B., and Biegler, L. T. 1991. Simultaneous solution and optimization strategies for parameter estimation of differential-algebraic equation systems. *Industrial and Engineering Chemistry Research*, 30(2), 376–385.
- Uno, T. 2001. A fast algorithm for enumerating bipartite perfect matchings. In *ISAAC 2001: the 12th Annual International Symposium on Algorithms and Computation*, LNCS 2223, (pp. 367–379). Springer-Verlag.

Bibliography

- Visser, D., and Heijnen, J. J. 2003. Dynamic simulation and metabolic re-design of a branched pathway using linlog kinetics. *Metabolic engineering*, 5(3), 164–176.
- Voss, S., Osman, I. H., and Roucairol, C. (Eds.) 1999. *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Norwell, MA, USA: Kluwer Academic Publishers.
- Wächter, A., and Biegler, L. T. 2006. On the implementation of an interior-point filter, line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1), 25–57.
- Wahl, S. A. 2007. *Methoden zur integrierten Analyse metabolischer Netzwerke unter stationären und instationären Bedingungen*. Ph.D. thesis, Research Centre Jülich, Germany.
- Wiechert, W. 2001. ^{13}C metabolic flux analysis. *Metabolic Engineering*, 3(3), 195 – 206.
- Wiechert, W., and de Graaf, A. A. 1997. Bidirectional reaction steps in metabolic networks: I. modeling and simulation of carbon isotope labeling experiments. *Biotechnology and Bioengineering*, 55(1), 101–17.
- Wiechert, W., Noack, S., and Elsheikh, A. 2010. Modeling languages for biochemical network simulation: Reaction vs equation based approaches. *Advances in Biochemical Engineering / Biotechnology*.
- Wiechert, W., and Takors, R. 2004. Validation of metabolic models: Concepts, tools, and problems. In H. V. Westerhoff, and B. Kholodenko (Eds.) *Metabolic Engineering in the Post Genomic Era (Horizon Bioscience)*. Horizon Scientific Press.
- Zaharie, D. 2002. Critical values for the control parameters of differential evolution algorithm. In *MENDEL 2002: The 8th International Conference on Soft Computing*. Brno, Czech Republic.