Generierung vereinfachter Modelle mechatronischer Systeme auf Basis symbolischer Gleichungen

Von der Fakultät für Ingenieurwissenschaften,
Abteilung Maschinenbau und Verfahrenstechnik der
Universität Duisburg-Essen
zur Erlangung des akademischen Grades

DOKTOR-INGENIEUR

genehmigte Dissertation

von

Lars Mikelsons
aus
Duisburg

Referent: Prof. Dr.-Ing. Dieter Schramm
Korreferent: Prof. Dr. Hans-Bernd Knoop
Tag der mündlichen Prüfung: 18.01.2012

Danksagung

Die vorliegende entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Lehrstuhl für Mechatronik der Universität Duisburg-Essen. Mein ganz besonderer Dank gilt Herrn Prof. Dieter Schramm für die Unterstützung bei der Durchführung dieser Arbeit und die mir gewährten Freiräume. Ebenso danken möchte ich Herrn Prof. Hans-Bernd Knoop für die Übernahme des Korreferats und die kritische Durchsicht der Arbeit.

Ich danke den Kollegen am Lehrstuhl für Mechatronik, die mir stets mit Rat und Tat zur Seite standen. Ganz besonders möchte ich Herrn Dr.-Ing. Tobias Bruckmann für seinen wertvollen Beitrag zu dieser Arbeit danken. Ebenso möchte ich Herrn Prof. Thorsten Brandt und Herrn Dr.-Ing. Oliver Lenord für die zahlreichen und fruchtbaren fachlichen Diskussionen sowie die Förderung dieser Arbeit danken. Ferne danke ich allen Studenten deren Ergebnisse zur Entstehung dieser Arbeit beigetragen haben.

Schließen möchte ich mit einem ganz besonderen Dank für das Verständnis und die Unterstützung meiner Familie.

Kurzfassung

Für die Entwicklung, Regelung und Optimierung technischer Systeme stellt die modellbasierte Simulation ein wichtiges Hilfsmittel dar. Dabei sind in vielen Anwendungen niedrige Simulationszeiten essentiell. Eine Simulation besteht zum einen aus dem Modell des technischen Systems und zum anderen aus einem numerischen Lösungsverfahren. Für eine effiziente Simulation sollte das verwendete Modell so einfach wie möglich sein, um den interessierenden physikalischen Effekt noch abbilden zu können, jedoch nicht einfacher.

Die Modellbildung technischer Systeme geschieht häufig in objektorientierten Simulationsumgebungen. Diese erlauben eine komfortable Modellierung per *drag&drop* mit Hilfe grafischer Benutzeroberflächen. Außerdem ist die Modellbildung weniger fehleranfällig, da die meisten Komponenten vorgefertigten Standard-Bibliotheken entnommen werden können. Darüber hinaus sind die generierten Modelle leicht wiederverwendbar. Diese Vorteile führen insbesondere dazu, dass die Erstellung komplexer Modelle technischer Systeme in objektorientierten Simulationsumgebungen in einfacher Art und Weise geschehen kann.

In der Regel werden von einem technischen System während des Entwicklungsprozesses mehrere Modelle benötigt. Die einfache Verfügbarkeit komplexer Modelle legt es nahe, die Modellbildung lediglich für das komplexeste Modell zu betreiben und alle weiteren Modelle durch Modellreduktion aus diesem zu generieren. In dieser Arbeit wird ein geeignetes Modellreduktionsverfahren entwickelt und in eine objektorientierte Simulationsumgebung integriert.

Das verwendete Modellreduktionsverfahren basiert darauf, ausgehend von einem vorgegebenen Szenario den Einfluss der in den Modellgleichungen enthaltenen mathematischen Terme zu schätzen. Anhand ihres geschätzten Einflusses auf das Simulationsergebnis werden die Terme anschließend sortiert und manipuliert (beispielsweise linearisiert oder vernachlässigt). Da die Schätzung des Einflusses szenariobasiert ist, hängt die Güte des Modells in einer Simulation von der Ähnlichkeit des Reduktionsszenarios zum Simulationsszenario ab. In dieser Arbeit werden zwei Ansätze

vorgestellt, um den Gültigkeitsbereich der reduzierten Modelle zu vergrößern. Darüber hinaus wird ein neues, äußerst effizientes Verfahren zur Schätzung des Einflusses der Terme präsentiert.

Ein besonderer Schwerpunkt liegt weiterhin auf der Generierung von Modellen für die Echtzeitsimulation. Für die Echtzeitsimulation muss jeder Integrationsschritt innerhalb einer vorgegebenen Zeitspanne berechnet werden können. Die zur Verfügung stehende Rechenleistung hängt dabei zum einen von der Zielhardware und zum anderen von dem gewünschten Takt ab. In dieser Arbeit wird ein Algorithmus vorgestellt, um ausgehend von einem komplexen Modell ein vereinfachtes Modell zu generieren, welches (falls überhaupt möglich) in Echtzeit auf der Zielhardware in dem gewünschten Echtzeittakt simuliert werden kann.

Die Leistungsfähigkeit des Verfahrens wird anhand bekannter Modelle aus der Fahrdynamik eindrucksvoll gezeigt. Darüber hinaus wurde im Rahmen der Arbeit ein komplexes Modell einer Baumaschine generiert und anschließend für Echtzeitsimulationen reduziert. Außerdem dient das Modellreduktionsverfahren als Basis für einen Ansatz zur Erstellung generischer Modelle.

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Problemstellung . . . 1
- 1.2 Literaturübersicht . . . 4
- 1.3 Inhalt und Ziel dieser Arbeit . . . 8

2 Grundlagen **11**
- 2.1 Objekorientierte Modellbildung . . . 11
  - 2.1.1 Grundkonzepte . . . 12
  - 2.1.2 Symbolische Manipulation objektorientierter Modelle . . . 19
  - 2.1.3 Kausalität . . . 22
  - 2.1.4 Die Modelica-Sprache . . . 23
- 2.2 Differential-algebraische Gleichungen . . . 25
  - 2.2.1 Indexbegriffe für DAEs . . . 27
  - 2.2.2 Indexreduktion . . . 31
  - 2.2.3 Numerische Integrationsverfahren für DAEs . . . 38

3 Gleichungsbasierte Modellreduktion **53**
- 3.1 Einleitung . . . 53
- 3.2 Reduktionstechniken . . . 55
  - 3.2.1 Termreduktion . . . 56
  - 3.2.2 Termsubstitution . . . 57
  - 3.2.3 Linearisierung von Termen . . . 57
  - 3.2.4 Ähnlichkeitsbetrachtungen . . . 58
- 3.3 Ranking-Verfahren . . . 58
  - 3.3.1 Residuenranking . . . 59
  - 3.3.2 Einschrittranking . . . 60
  - 3.3.3 Sensitivitätsranking . . . 61
  - 3.3.4 Güte der Ranking-Verfahren . . . 63
- 3.4 Reduktion für eine vorgegebene Fehlertoleranz . . . 66

3.4.1 Approximationsalgorithmus 66
3.4.2 Erweiterung für multiple Szenarien 71
3.4.3 Erweiterung für intervallwertige Szenarien 74
3.5 Reduktion für Echtzeitsimulationen 83
3.5.1 Echtzeitsimulation 85
3.5.2 Approximationsalgorithmus 91
3.6 Szenarienwahl 96

4 Anwendungsbeispiele **99**
4.1 Implementierung 99
4.2 Anwendung auf Beispiele aus der Fahrzeugdynamik 101
4.2.1 Verwendete Fahrzeugmodelle 101
4.2.2 Reduktion für eine vorgegebene Fehlertoleranz 106
4.2.3 Reduktion für Echtzeitsimulationen 109
4.2.4 Reduktion für intervallwertige Szenarien 112
4.3 Anwendung auf das Modell einer Baumaschine 116
4.3.1 Modellierung des Skid-Steer-Loaders 117
4.3.2 Reduktion des Skid-Steer-Loaders für Echtzeitsimulationen . . 120
4.4 Ansatz zur Erstellung von generischen Modellen 123
4.4.1 Dimensionsanalyse 124
4.4.2 Dimensionsloses Fahrzeugmodell 128
4.4.3 Anwendung der gleichungsbasierten Modellreduktion 131

5 Zusammenfassung und Ausblick **137**

Anhang **141**

Literaturverzeichnis **157**

KAPITEL 1

Einleitung

Abschnitt 1.1 erläutert die im Rahmen dieser Arbeit behandelte Problemstellung. Abschnitt 1.2 fasst den Stand der Technik zusammen und bettet die Problemstellung in den wissenschaftlichen Kontext ein. In Abschnitt 1.3 werden die Ziele der Arbeit formuliert.

1.1 Problemstellung

Der Mensch entwickelt seit jeher Werkzeuge und technische Systeme, um mit seiner Umwelt zu interagieren. Für die Entwicklung jener Systeme war die Simulation schon sehr früh ein wichtiges Hilfsmittel. Die ersten Modelle, die für die Simulation eingesetzt wurden, waren Gedankenmodelle [Bossel, 1994]. Später kamen physikalische Modelle hinzu. Oftmals wurden skalierte Modelle gebaut, bevor das technische System in Originalgröße realisiert wurde, um die Funktionalität des Systems vorab zu testen. Auch heute noch werden beide Arten von Modellen verwendet. Heutzutage stellt jedoch auch die Simulation auf Basis von mathematischen Modellen einen wichtigen Baustein im Entwicklungsprozess dar. Mit Hilfe des V-Modells können die Vorteile der Simulation unter Verwendung von mathematischen Modellen leicht erläutert werden (siehe Abbildung 1.1). Das V-Modell ist eine Formalisierung des mechatronischen Entwicklungsprozesses, welcher vielen mechatronischen Produktentwicklungen zugrunde liegt.

Nach der Definition der Anforderungen an das technische System wird der Entwicklungszyklus in der Regel mehrmals durchlaufen. Dieser Zyklus beginnt mit einem ersten Entwurf des technischen Systems. Dieser setzt sich zusammen aus dem Systementwurf sowie den domänenspezifischen Entwürfen der Subsysteme der vertretenden

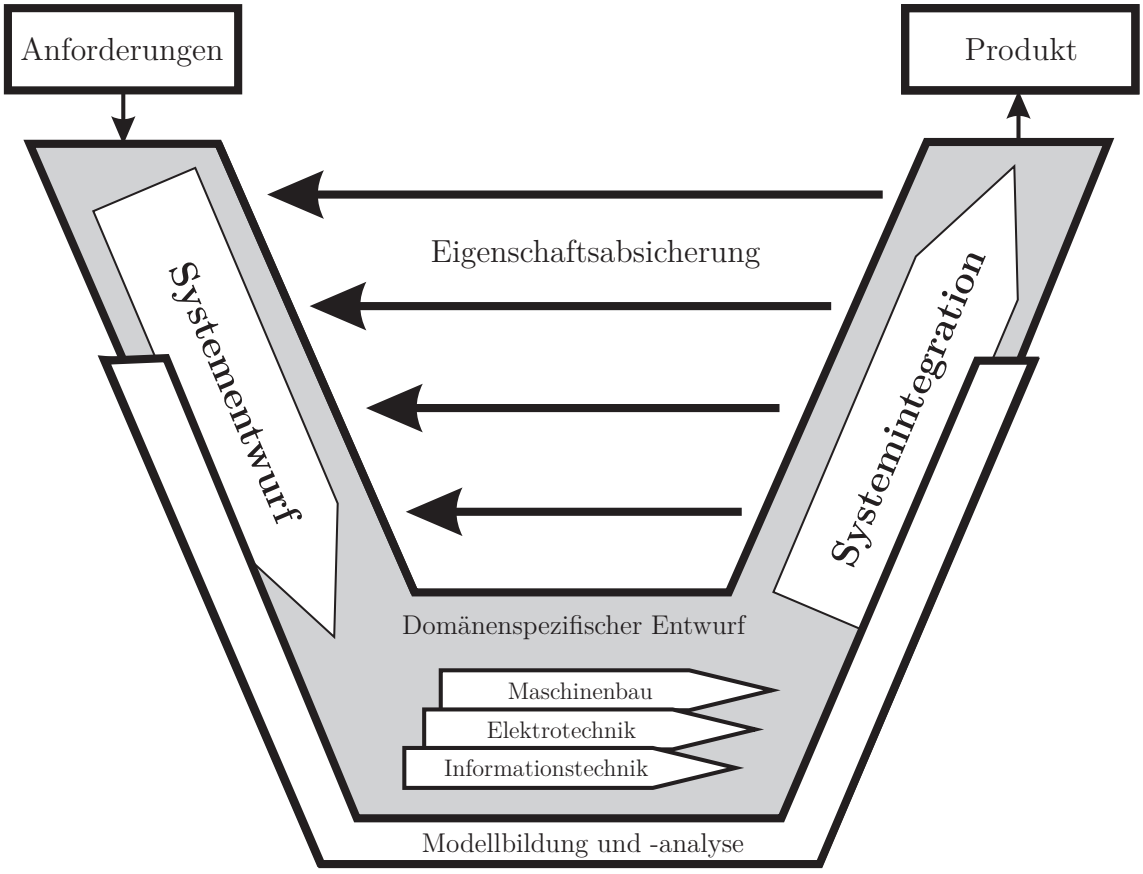


Abbildung 1.1: V-Modell der mechatronischen Produktentwicklung [VDI2206, 2004]

physikalischen Domänen. Der rechte Zweig des V-Modells beschreibt die Integration der domänenspezifischen Subsysteme. Während dieser Realisierungsphase wird überprüft, ob das Produkt bereits alle zuvor geforderten Eigenschaften aufweist. Ist dies nicht der Fall, wird der Systementwurf optimiert und der Zyklus beginnt von Neuem. Es ist wünschenswert, den gesamten Entwicklungsprozeß in möglichst wenig Iterationen zu durchlaufen, da insbesondere die Systemintegration einschließlich des Prototypenbaus sehr kostenintensiv ist. Wird im Rahmen der Eigenschaftsabsicherung die Nichterfüllung einer geforderten Eigenschaft schon frühzeitig mit Hilfe der Simulation erkannt, so werden Kosten und Zeit gespart.

Da Gedankenmodelle in der Regel einen niedrigen Detaillierungsgrad haben, können mit ihrer Hilfe nur wenige Eigenschaften abgesichert werden. Allerdings können von vielen technischen Systemen mathematische Modelle mit sehr hohem Detaillierungsgrad generiert werden. Folglich können viele Eigenschaften bereits in der Simulation abgesichert werden. Der Einsatz mathematischer Modelle bietet neben der früheren Verfügbarkeit noch weitere Vorteile gegenüber dem Einsatz von realen Modellen. So ist es schwierig, ein Experiment mit einem realen Modell unter identischen Umständen zu wiederholen. Zusätzlich ist es riskant, gefährliche Experimente durchzuführen, da eine Beschädigung des realen Modells kostspielig sein kann und eine Gefährdung von Personen von vornherein ausgeschlossen werden muss. Eine Simulation kann beliebig oft mit einstellbaren äußeren Einflüssen ohne jegliche Ge-

fahr für Mensch oder Maschine wiederholt werden. Dies macht die Simulation auf Basis mathematischer Modelle zu einem wichtigen Werkzeug im oben beschriebenen mechatronischen Entwicklungsprozess.

Seit der Entwicklung der ersten technischen Systeme und Werkzeuge steigen die Anforderungen an Funktion und Qualität stetig an. Aus dieser Entwicklung resultieren viele der modernen hochkomplexen Produkte, die den heutigen Alltag erleichtern. Um den wirtschaftlichen Erfolg dieser Produkte zu gewährleisten, muss die steigende Komplexität einhergehen mit niedrigen Entwicklungskosten und kurzen Entwicklungszeiten. Aus diesem Grund ist es notwendig, auch die Simulationsumgebungen ständig zu verbessern. Insbesondere der Trend zu mechatronischen Lösungen verlangt nach domänenübergreifenden Simulationsumgebungen.

Hier hat die Verwendung von objektorientierten Lösungsansätzen einen wichtigen Beitrag geleistet. Objektorientierte Modellbildung erleichtert nicht nur die Modellbildung domänenübergreifender Systeme, sie führt auch zu Modellen, die wiederverwendbar und leicht zu warten sind. Objektorientierte Simulationsumgebungen verfügen in der Regel über grafische Benutzeroberflächen und vorgefertigte Bibliotheken mit Standardkomponenten. So können heutzutage die Modelle komplexer Systeme unter Verwendung dieser Bibliotheken per *„drag&drop“* generiert werden.

Für die Entwicklung technischer Systeme ist jedoch ein isoliertes Modell des betrachteten Systems in der Regel nicht ausreichend. Meist werden mehrere Modelle verschiedener Detaillierungsgrade benötigt. Hinzu kommen Modelle, welche lediglich bestimmte physikalische Effekte abbilden sollen. Bislang sind objektorientierte Simulationsumgebungen nicht in der Lage, den Detaillierungsgrad eines Modells zu variieren. Folglich muss nach wie vor ein Experte auf Basis seiner Erfahrung und seines Systemwissens die Teile des Modells identifizieren, welche für den betrachteten Detaillierungsgrad wenig relevant sind. Diese werden dann entfernt oder durch einfachere Modelle ersetzt, um den Detaillierungsgrad zu senken.

Ein Anwendungsfall für Modelle mit häufig niedrigem Detaillierungsgrad, der in den letzten Jahren besonders an Relevanz gewonnen hat, ist die Echtzeitsimulation. Hierfür gibt es mehrere Gründe. Zum einen stellen auch Mikroprozessoren mittlerweile genug Rechenleistung zur Verfügung, um zumindest Modelle von relativ geringer Komplexität in Echtzeit zu simulieren. Zum anderen erleichtern objektorientierte Simulationsumgebungen die Erzeugung von Code, der auf Echtzeitbetriebssystemen lauffähig ist. Die meisten objektorientierten Simulationsumgebungen erzeugen mathematische Modelle, deren Gleichungen symbolisch vorliegen. Dies hat die Möglichkeit eröffnet, aus den symbolischen Gleichungen C-Code zu erzeugen, diesen mit einem numerischen Integrationsverfahren zu koppeln und für ein Echtzeitbetriebssystem zu kompilieren.

Die Modellierung eines Systems für die Echtzeitsimulation stellt jedoch meist eine große Herausforderung dar. In diesem Fall wird ein Modell benötigt, das auf der einen Seite die relevanten physikalischen Effekte in der gewünschten Genauigkeit abbildet und auf der anderen Seite einfach genug ist, um auf der gewünschten Zielhardware im gewünschten Echtzeit-Takt simuliert werden zu können. Die Generierung eines solchen Modells ist in der Regel ein zeitintensiver, iterativer Prozess, in dem der

Detaillierungsgrad eines komplexen Modells des betrachteten Systems immer weiter gesenkt und auf der Zielhardware getestet wird. Kann das Modell schließlich auf der Zielhardware in Echtzeit simuliert werden, so bricht der Reduktionsprozess ab.

In Bereichen, in denen die Zielhardware eine hohe Rechenleistung aufweist, ist dieser Prozess meist relativ kurz. Ein Beispiel hierfür ist der Simulatorenbau, in dem heutzutage meist leistungsfähige PC-Hardware oder spezielle Echtzeit-Hardware zum Einsatz kommt. Gerade bei Produkten, welche in großer Stückzahl produziert werden, ist jedoch schon wegen des Kostendrucks deutlich weniger Rechenleistung verfügbar. Hier ist die Generierung eines echtzeitfähigen Modells eine komplexe Aufgabe, die nur mit Hilfe von Erfahrung und Systemwissen gelöst werden kann.

Um Zeit und Kosten zu sparen, ist folglich ein Werkzeug wünschenswert, das ausgehend von einem einzelnen Modell eines Systems die weiteren benötigten Modelle automatisiert erzeugen kann. Hierbei kommt der Erzeugung von echtzeitfähigen Modellen eine besondere Bedeutung zu.

1.2 Literaturübersicht

Ausgehend von der zuvor beschriebenen Problemstellung stellt sich die Frage, ob der Detaillierungsgrad des Modells, aus dem weitere benötigte Modelle generiert werden sollen, besonders hoch, besonders niedrig oder dazwischen liegen sollte. Da die detaillierte Modellierung von komplexen Systemen in objektorientierten Simulationsumgebungen heutzutage relativ einfach geschehen kann, bietet sich das detaillierteste Modell als Ausgangsmodell an. In der Folge können die Modelle mit geringerem Detaillierungsgrad durch Modellreduktion gewonnen werden. Dieses Vorgehen erfordert ein Modellreduktionsverfahren, das sich auf einfache Art und Weise mit einer objektorientierten Simulationsumgebung koppeln lässt.

Die einfachsten Modellreduktionsverfahren basieren auf der Idee, die schnellen Lösungskomponenten zu vernachlässigen und lediglich die dominierenden Komponenten zu berücksichtigen [Davison, 2002]. Dieser Ansatz ist jedoch beschränkt auf Systeme mit linearer Dynamik, also Systemen, die geschrieben werden können als

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{1.1}$$

mit konstanten Matrizen $\mathbf{A}$ und $\mathbf{B}$. Die Zuordnung in schnelle und langsame (dominierende) Lösungskomponenten geschieht dabei mit Hilfe von Systemwissen. Die Beschränkung auf lineare Systeme macht diese Methode für den Einsatz in objektorientierten Simulationsumgebungen jedoch unbrauchbar, da die erzeugten mathematischen Modelle dort in der Regel nichtlinear sind. An diesem Kriterium scheitert beispielsweise auch die *optimale Hankel-Norm Approximation* [Zhou *et al.*, 1996; Glover, 1984] oder die *Pade Approximation* [Alexandro, 1984].

Auch die *Modal-Analyse* basiert auf der Idee, unwichtige Lösungskomponenten zu vernachlässigen, kann dabei aber auf nichtlineare Systeme erweitert werden [Margolis

& Young, 1977; Fertis, 1995]. In der einfachsten Form wird ein Differentialgleichungssystem zweiter Ordnung der Form

$$\mathbf{M}\ddot{\mathbf{x}} + \mathbf{K}\mathbf{x} = \mathbf{0}, \tag{1.2}$$

auf ein System der Form aus Gleichung (1.1) mit einer Diagonalmatrix $\mathbf{A}$ projiziert. Hierzu wird das Differentialgleichungssystem zunächst auf ein System erster Ordnung transformiert und anschließend ein neuer Zustandsvektor $\boldsymbol{\xi}$ eingeführt, so dass $\mathbf{A}$ eine Diagonalmatrix ist. Diese Projektion gelingt mit Hilfe der so genannten *Normalmoden* $\mathbf{r}$, die aus dem verallgemeinerten Eigenwertproblem

$$\mathbf{K}\mathbf{r} = \omega^2\mathbf{M}\mathbf{r} \tag{1.3}$$

berechnet werden, wobei ω die Eigenfrequenzen bezeichnet. Aufgrund der Diagonalform von $\mathbf{A}$ können so sehr einfach Moden eliminiert und damit die Dimension des Modells reduziert werden. Diese Methode ist zwar auf nichtlineare Systeme verallgemeinerbar [Gorban *et al.*, 2006], jedoch weist sie einen anderen Nachteil auf: Nach der Projektion ist das Modell in anderen Koordinaten beschrieben als vorher. In der Folge sind physikalische Interpretationen anhand des reduzierten Modells nur noch schwer möglich, da die physikalische Bedeutung der Zustandsvariablen nicht mehr ohne Weiteres ersichtlich ist. Darüber hinaus existiert eine eindeutige Rücktransformation in der Regel nicht.

Diesen Nachteil weisen leider sämtliche weiteren Projektionsmethoden ebenfalls auf. Die meisten der Projektionsmethoden basieren auf der *Karhunen-Loeve Transformation* [Hotelling, 1933; Sirovich, 1987], welche eine Transformation in einen geeigneten Unterraum des Zustandsraums darstellt. Dieser Unterraum ist auf der einen Seite von geringerer Dimension als der Zustandsraum, reicht auf der anderen Seite aber aus, um die maßgebliche Dynamik zu beschreiben. Identifiziert wird der Unterraum mit Hilfe von gemessenen oder durch Simulation erzeugten Daten. Dabei wird eine Basis dieses Unterraums generiert, die nicht Teilmenge der verwendeten Basis des Zustandsraums ist. Die Verwendung dieser neuen Basis ermöglicht die Reduktion, führt jedoch auf den Nachteil der ungewohnten Koordinaten.

Eine Methode, die keine Koordinatentransformation benötigt und zur Reduktion von nichtlinearen Modellen geeignet ist, basiert auf der Betrachtung des mathematischen Modells als singulär gestörtes System der Form [Kokotović *et al.*, 1999]

$$\dot{\mathbf{x}}_1 = \mathbf{f}_1(\mathbf{x}_1, \mathbf{x}_2), \tag{1.4}$$

$$\epsilon\dot{\mathbf{x}}_2 = \mathbf{f}_2(\mathbf{x}_1, \mathbf{x}_2). \tag{1.5}$$

Dabei ist ϵ ein kleiner Parameter, in dem alle Parameter des Modells, die als klein angenommen werden können, zusammengefasst sind. Dann wird derjenige Teil des mathematischen Modells, der eine langsame Dynamik aufweist (Gleichung (1.4)), von demjenigen Teil, der eine schnelle Dynamik hat (Gleichung (1.5)), entkoppelt. Hierzu wird des schnelle Submodell im stationären Zustand

$$0 = \mathbf{f}_2(\mathbf{x}_1, \mathbf{x}_2), \tag{1.6}$$

gelöst. Mit der Annahme, dass diese Gleichung geschrieben werden kann als

$$\mathbf{x}_2 = \mathbf{\Phi}(\mathbf{x}_1), \tag{1.7}$$

folgt durch Einsetzen aus Gleichung (1.4)

$$\dot{\mathbf{x}}_1 = \mathbf{f}_1(\mathbf{x}_1, \mathbf{\Phi}(\mathbf{x}_1)). \tag{1.8}$$

Zum einen wird auf diese Art und Weise die Dimension des Modells gesenkt und zum anderen ist die Steifigkeit des Modells geringer, da die schnelle und die langsame Dynamik nicht mehr in einem Modell vereint sind. In der Folge kann ein numerisches Integrationsverfahren größere Schrittweiten verwenden. Ein ähnlicher Ansatz wird in [Burgermeister *et al.*, 2009] gewählt. Auch hier werden die schnellen Zustände durch stationäres Verhalten approximiert. Der Nachteil dieser Verfahren besteht darin, dass für die Partitionierung des Modells in schnelle und langsame Anteile *a priori* bekannt sein muss, welcher Zustand zu einem schnellen oder langsamen Submodell gehört. Dieses Systemwissen ist in einer Simulationsumgebung im Allgemeinen nicht verfügbar, weshalb dieses Verfahren im Rahmen dieser Arbeit nicht zum Einsatz kommen kann.

In [Rosenberg & Zhou, 1988] wird ein Modellreduktionsverfahren vorgestellt, das auf nichtlineare Systeme angewendet werden kann, auf Koordinatentransformationen verzichtet und kein Systemwissen verwendet. Im Gegensatz zu den bisherigen Reduktionstechniken arbeitet dieses Verfahren nicht auf der Gleichungsebene, sondern auf Komponentenebene. Während bei den bisherigen Ansätzen Zustände identifiziert wurden, die wenig Einfluss auf die Gesamtlösung des Modells haben, werden hier Komponenten gesucht, die wenig Einfluss auf die Gesamtlösung des Systems haben. Um den Einfluss auf die Lösung zu bestimmen, wird zunächst der quadratische Mittelwert des Energieflusses durch jede Komponente des Modells bestimmt. Basierend auf der Annahme, dass Komponenten, durch die wenig Energie fließt, wenig Einfluss auf die Lösung haben, werden dann entsprechende Komponenten vernachlässigt.

In [Louca *et al.*, 1997] wird *MORA*, eine Erweiterung des Verfahrens von Rosenberg und Zhou, vorgestellt. Zum einen wird dort nicht mehr der quadratische Mittelwert des Energieflusses, sondern die *Aktivität* verwendet und zum anderen wird diese Aktivität für alle Elemente, die in der Bondgraphen-Repräsentation des Systems auftauchen, bestimmt. Somit ist das Modellreduktionsverfahren nicht auf ganze Komponenten des Modells beschränkt. Die Aktivität A eines Elements entpricht hierbei dem Integral über den Betrag der Leistung über die gesamte Simulationszeit. Die Leistung kann dabei immer als Produkt eines Variablenpaars, bestehend aus $a(t)$ und $f(t)$ geschrieben werden (siehe Abschnitt 2.1). Damit ist die Aktivität eines Elements definiert durch

$$A = \int_0^T |f(t)a(t)|\, dt. \tag{1.9}$$

Ein weiterer Vorteil dieses Ansatzes ist, dass es mit Hilfe der Aktivität möglich ist, Modelle in für Kosimulationen geeignete Submodelle zu zerlegen [Rideout *et al.*, 2004].

Ersal kombiert MORA und die Karhunen-Loeve Transformation in [Ersal *et al.*, 2009]. So werden dort nicht mehr ausschließlich die Höhe der Aktivitäten, sondern auch deren Korrelation in Betracht gezogen. Zwar beinhaltet die Karhunen-Loeve Transformation einen Koordinatenwechsel, doch wird das Ergebnis wieder auf das Modell zurück projiziert, so dass das reduzierte Modell dieselben Zustände besitzt, wie das ursprüngliche Modell. Der Nachteil dieser Methode besteht darin, dass das Modell als Bond-Graph vorliegen muss.

Für die Simulation und Auslegung von analogen Schaltkreisen werden häufig *Verhaltensmodelle* verwendet. Verhaltensmodelle stellen dabei nichts anderes dar als reduzierte Modelle. So wird in [Borchers, 1998] ein Verfahren entwickelt, mit dessen Hilfe die Modelle von nichtlinearen analogen Schaltkreisen reduziert werden können. Im Gegensatz zu den bisherigen Verfahren werden bei diesem Verfahren weder Zustände vernachlässigt, noch eine Graphenrepräsentation manipuliert. Statt dessen wird die symbolische Repräsentation der Gleichungen, die das mathematische Modell darstellen, manipuliert. Da es speziell für analoge Schaltkreise entwickelt wurde, wird für die Reduktion jedoch lediglich eine DC- oder eine AC-Analyse des Modells verwendet[1]. Basierend auf dieser Analyse wird dann geschätzt, wie groß der Einfluss aller in den Gleichungen enthaltener Terme auf die Lösung ist. Anschließend werden die *unwichtigen* Terme nach und nach vernachlässigt, bis eine vom Benutzer vorgegebene Fehlergrenze erreicht ist. In [Popp *et al.*, 2002; Nathke *et al.*, 2004] werden alternative Reduktionskriterien vorgestellt. Wichmann erweitert dieses Reduktionsverfahren, so dass die Methode auf beliebige nichtlineare Modelle angewendet werden kann in [Wichmann, 2003]. In [Wichmann, 2004] wird zusätzlich zur AC- und DC-Analyse die transiente Analyse (Simulation) als Basis für die Schätzung des Einflusses der Terme des Modells ermöglicht. Somit war der Weg zur Reduktion von mechatronischen Modellen geebnet. Obwohl in [Sommer *et al.*, 2008] gezeigt wird, dass die Reduktion mechatronischer Modelle möglich ist, wird dieses Verfahren bisher, nach Wissen des Autors, nur in der Elektrotechnik eingesetzt.

Diese gleichungsbasierte Modellreduktion basiert auf einer symbolischen Repräsentation des Modells, die in (fast) allen objektorientierten Simulationstools verfügbar ist. Ebenso wie für die Berechnung der Aktivitäten bei Louca ist bei gleichungsbasierter Modellreduktion eine Referenzsimulation notwendig. Die dazu benötigten Größen, wie zum Beispiel Anfangswerte und Parameter, müssen in objektorientierten Simulationstools jedoch ohnehin angegeben werden. Darüber hinaus wird keine Koordinatentransformation benötigt. Folglich erscheint die gleichungsbasierte Modellreduktion geeignet als Basis für die Generierung von Modellen verschiedener Detaillierungsstufen.

Weitere Übersichtsliteratur zu Modellreduktionsverfahren findet man beispielsweise in [Gugercin & Antoulas, 2002; Ersal *et al.*, 2008] und [Lamba & Mahmoud, 1982].

[1] Die DC- und die AC-Analyse sind numerische Verfahren zu Analyse von analogen Schaltkreisen. Bei der DC-Analyse wird das Verhalten des Modells im Gleichgewichtszustand betrachtet. Die dynamischen Anteile des Modells werden also vernachlässigt. In der Elektrotechnik entspricht dies, Kapazitäten durch Leerläufe und Induktivitäten durch Kurzschlüsse zu ersetzen. Bei der AC-Analyse wird die Lösung des linearisierten Modells bei einer sinusförmigen Anregung mit konstanter Frequenz betrachtet. Die AC-Analyse wird üblicherweise im Frequenzraum durchgeführt.

Literatur zu Modellreduktionsverfahren, die die Vorgabe einer Echtzeit-Hardware, beschrieben durch Anzahl möglicher Rechenoperationen pro Zeiteinheit, erlauben, existiert nach Wissen des Autors nicht.

1.3 Inhalt und Ziel dieser Arbeit

Die automatisierte Erzeugung von Modellen mit verschiedenen Detaillierungstufen, stellt eine enorme Vereinfachung des Modellierungsprozesses dar. In der Folge können das V-Modell schneller durchlaufen und somit Zeit und Kosten gespart werden. Ziel dieser Arbeit ist die Weiterentwicklung eines Modellreduktionsverfahrens, das ausgehend von einem komplexen Modell, automatisiert Modelle mit einem vorgegebenen Detaillierungsgrad erzeugen kann. Dieses Modellreduktionsverfahren soll anhand komplexer mechatronischer Systeme validiert und in eine objektorientierte Simulationsumgebung integriert werden. Daher muss es möglich sein, das Modellreduktionsverfahren in die Struktur eines objektorientierten Simulationstools einzugliedern. Diese Rahmenbedingung erfordert ein Verfahren, das auf beliebige Modelle anwendbar ist, auf Koordinatentransformationen verzichtet und auf der symbolischen Repräsentation der Gleichungen operiert. Da heutige Entwicklungsprozesse oftmals auf Hardware-in-the-Loop, oder Human-in-the-Loop-Konzepte zur Eigenschaftsabsicherung zurückgreifen, stellt die Erzeugung von echtzeitfähigen Modellen ein wesentliches Ziel dar.

In Abbildung 1.2 sind mögliche Verwendungszwecke eines mathematischen Modells in einem Simulationstool anhand des Beispiels einer Baumaschine gezeigt. Neben der Betrachtung des Verlaufs aller Zustände (Mitte) können die Simulationsergebnisse visualisiert werden (links) oder das Modell als echtzeitfähiger Code exportiert werden (rechts). Die dritte Möglichkeit weist jedoch den Nachteil auf, dass der Code üblicherweise zwar generiert werden kann, jedoch nicht geprüft wird, ob der Code in Echtzeit auf der gewünschten Zielhardware lauffähig ist. Daher soll in dem Modellreduktionsverfahren neben der Vorgabe einer Fehlertoleranz auch die Vorgabe einer gewünschten Zielhardware möglich sein. Dies ermöglicht die Erzeugung von Code für Echtzeitsimulationen, die garantiert in einem gewünschten Takt auf der vorgegenenen Zielhardware ausgeführt werden können.

Die Kopplung einer objektorientierten Simulationsumgebung mit einem Modellreduktionsverfahren ist zusätzlich ein Schritt in Richtung generischer Modellierung. Das entwickelte Verfahren macht es möglich, den Detaillierungsgrad von Modellen zu variieren. Somit sind die Modelle generisch bezüglich ihres Detaillierungsgrades. An dieser Stelle ist es möglich, noch einen Schritt weiter zu gehen. In der Idealvorstellung sind Modelle nicht nur generisch bezüglich ihres Detaillierungsgrades, sondern auch generisch bezüglich ihrer Dimension. Hierbei bezeichnet Dimension nicht die Dimension des mathematischen Modells, sondern die Dimension der Zustände und Parameter. So ist es wünschenswert, dass ein Modell über so wenig Parameter wie möglich verfügt und auf einfache Art und Weise skaliert werden kann. Darüber hinaus ist insbesondere für Regelungsaufgaben die einfache Erstellung eines Nominalmodells wünschenswert. In dieser Arbeit wird die gleichungsbasierte Modellreduktion mit der Dimensionsanalyse gekoppelt, so dass in einer Fallstudie alle

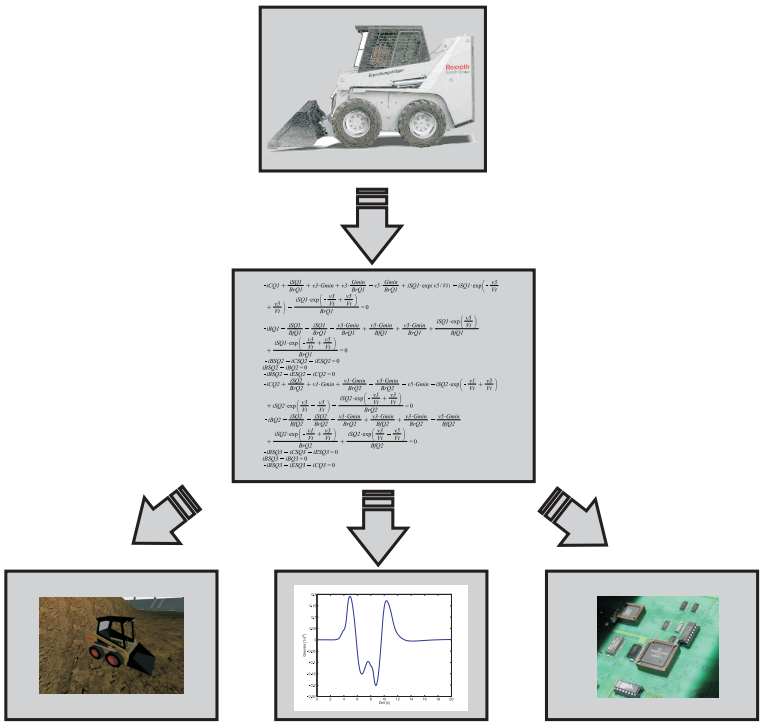


Abbildung 1.2: Verwendungsmöglichkeiten von Modellen in objektorientierten Simulationsumgebungen

oben gelisteten Eigenschaften für ein Modell realisiert werden können. Somit wird der Grundstein für die Erzeugung von Modellen gelegt, die sowohl generisch in ihrem Detaillierungsgrad als auch bezüglich der Dimension ihrer Zustände und Parameter sind.

Im nächsten Abschnitt werden zunächst die notwendigen Grundlagen erläutert. Hier wird auf die Paradigmen der objektorientierten Modellbildung sowie die daraus resultierenden Vorteile eingegangen. Anschließend werden mathematische Modelle, die aus einer objektorientierten Modellbildung hervorgehen, betrachtet. Hierzu werden numerische Lösungsverfahren vorgestellt und bewertet.

In Kapitel 3 wird dann die gleichungsbasierte Modellreduktion eingeführt. Hier werden sowohl das bekannte Verfahren aus [Wichmann, 2004] sowie die im Rahmen dieser Arbeit entwickelten Erweiterungen vorgestellt. Hierzu zählen zum einen zwei Erweiterungen, um den Gültigkeitsbereich der reduzierten Modelle zu vergrößern und zum anderen die Reduktion für Echtzeitsimulationen. Die Vorgehensweise der entsprechenden Algorithmen wird an einfachen Beispielen illustriert.

Die Validierung des Verfahrens ist in Kapitel 4 beschrieben. Hierzu wird zunächst die Anwendung der gleichungsbasierten Modellreduktion auf Modelle aus der Fahrzeugdynamik beschrieben, da die reduzierten Modelle dort leicht zu interpretieren sind. Danach wird die gleichungsbasierte Modellreduktion auf das Modell einer Baumaschine angewendet. Dieses Modell vereint hydraulische, mechanische und regelungstechnische Komponenten und stellt somit ein typisches mechatronisches Modell dar. Das Kapitel schließt mit der Vorstellung des Ansatzes zur Erstellung von generischen

Modellen.

Die Arbeit schließt in Kapitel 5 mit einer Zusammenfassung und einem Ausblick auf zukünftige Arbeiten.

KAPITEL 2

Grundlagen

Gleichungsbasierte Modellreduktion eignet sich insbesondere für die Reduktion von Modellen, die mit Hilfe von objektorientierten Simulationsumgebungen erstellt wurden. Die entsprechenden mathematischen Modelle sind im Allgemeinen differential-algebraische Gleichungen (Differential Algebraic Equations). Im ersten Teil dieses Kapitels wird gezeigt, wie die Paradigmen der objektorientierten Modellbildung auf komplexe DAEs führen. Darüber hinaus werden Methoden vorgestellt, um diese DAEs zu vereinfachen. Außerdem wird die populärste objektorientierte Modellierungssprache Modelica kurz vorgestellt, da die im Rahmen dieser Arbeit entwickelten Algorithmen unter anderem in einem Compiler für Modelica umgesetzt wurden. Im zweiten Teil des Kapitels werden Methoden zur Klassifizierung und Berechnung der Lösung von DAEs vorgestellt.

2.1 Objekorientierte Modellbildung

Der Trend zu mechatronischen Ansätzen und Lösungen zwingt Ingenieure dazu, domänenübergreifend zu denken. In der Folge werden Simulationsumgebungen benötigt, die eine domänenübergreifende Modellbildung erlauben, um das Systemverhalten in der Simulation zu bewerten. Früher erfolgte die Modellbildung der Subsysteme in einer domänenspezifischen Simulationsumgebung von Experten für die jeweilige Domäne. In der Folge wurde für den Entwurf eines Subsystems lediglich dessen Modell betrachtet. Im mechatronischen Entwicklungsprozess erfolgt die Modellbildung des Gesamtsystems in einer einzelnen Simulationsumgebung. Dabei wird die Modellierungsaufgabe häufig von einem interdisziplinären Team erledigt, so dass für

den Systementwurf der Subsysteme alle Subsysteme berücksichtigt werden. Die Mitglieder eines solchen Teams sind jedoch in der Regel Experten für jeweils lediglich einen Teil des Gesamtsystems. Die Forderung nach Modellen, welche für verschiedene Zwecke und von verschiedenen Personen in einfacher Art und Weise weiterverwendet werden können, ist die logische Konsequenz. Insbesondere sollten Modelle nicht nur von Experten, sondern auch von Laien benutzt werden können. Mit anderen Worten: Die Modelle sollten nutzbar sein, ohne die Details des betrachteten Systems zu kennen.

Ein weiteres Argument für die Wiederverwendbarkeit von Modellen ist, dass für die Modellierung der meisten Systeme innerhalb einer Domäne einige Standardkomponenten ausreichend sind. Die Verwendung von Bibliotheken von Standardkomponenten spart folglich Entwicklungszeit und minimiert das Risiko von Fehlern.

Die Modellbildung geschieht heute meist mit Hilfe von grafischen Benutzeroberflächen. Hier ist es neben der Wiederverwendbarkeit von Modellen wünschenswert, dass sich die Topologie des Systems in der (grafischen) Repräsentation des Systems widerspiegelt. So lässt sich die Struktur des betrachteten Systems leichter erkennen und Subsysteme können ohne großen Aufwand isoliert analysiert werden. Diese beiden Eigenschaften werden in der objektorientierten Modellbildung vereint.

2.1.1 Grundkonzepte

Der Begriff *objektorientierte Modellierung* wurde erstmals in [Nilsson, 1989] verwendet. Die Idee kam jedoch schon früher auf. So wurden in [Elmqvist, 1978] und [Runge, 1977] bereits Ende der siebziger Jahre unabhängig voneinander strukturierte Sprachen zur Modellierung komplexer Systeme entwickelt, welche heute als erste objektorientierte Modellierungssprachen angesehen werden. Die entwickelten Sprachen konnten ihre Leistungsfähigkeit jedoch erst später zeigen, denn die Computer waren Ende der siebziger Jahre nicht leistungsfähig genug, um die generierten Modelle symbolisch zu vereinfachen oder in angemessener Zeit zu lösen. Obwohl die Konzepte objektorientierter Modellierung also lange bekannt sind, ist es nach wie vor schwierig, den Begriff objektorientierte Modellbildung exakt zu definieren. Daher wird hierzu meist, wie auch im Folgenden, eine Liste von Eigenschaften herangezogen, welche von einer Modellierungssprache erfüllt werden muss, um als objektorientiert zu gelten [Andersson, 1994].

Verwendung deklarativer Modelle Die in einem Modell enthaltenen Gleichungen sind deklarativ. Dies bedeutet, dass das Gleichheitszeichen keine Zuweisung, sondern eine Äquivalenzrelation ist. Die Modelle

$$F = c \cdot \Delta x \tag{2.1}$$

und

$$\Delta x = \frac{F}{c} \tag{2.2}$$

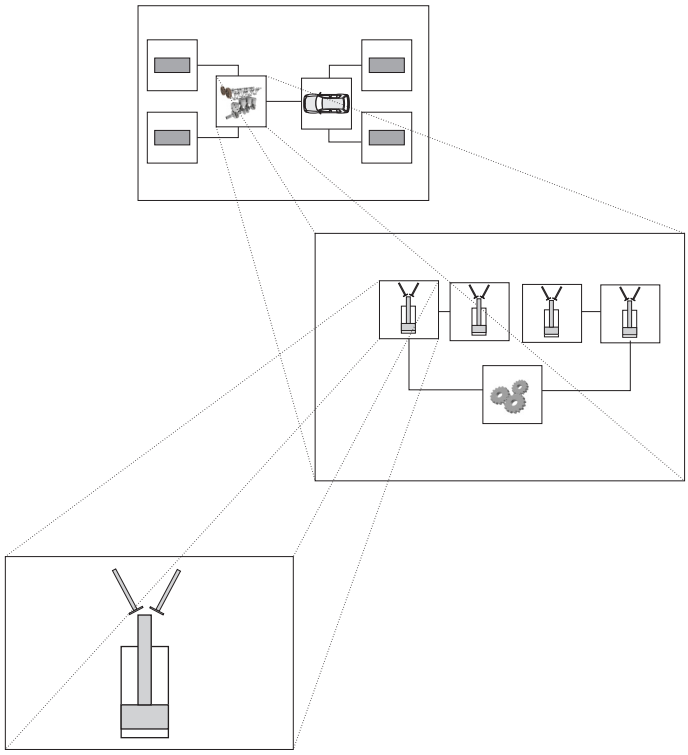


Abbildung 2.1: Hierarchiestufen in einem Modell

sind also äquivalent. Deklarative Modelle werden in den meisten Fällen durch akausale Modelle realisiert. Auf akausale Modelle und ihre Folgen für die Numerik wird in Abschnitt 2.1.3 näher eingegangen.

Modularität Modularität ist der Schlüssel zu strukturierten, verständlichen Modellen. Jedes Modell kann aus beliebig vielen Submodellen bestehen, welche ihrerseits wieder aus Submodellen bestehen können. Beispielsweise enthält das Modell eines Kraftfahrzeugs unter anderem das Modell eines Motors, welches wiederum das Modell eines Zylinders enthält. Diese Art der Modellierung wird auch *hierarchische Modellierung* genannt. Nur die Modelle der untersten Hierarchiestufe beinhalten Gleichungen. Dieses Prinzip ist in Abb. 2.1 veranschaulicht. In dem obigen Beispiel entspricht das oberste Bild dem Modell des Kraftfahrzeugs, das mittlere Bild dem Modell des Motors und das untere Bild dem Modell eines Zylinders. Von den hervorgehobenen Modellen sind dabei lediglich im Modell des Zylinders Gleichungen hinterlegt. Ein Modell entspricht hierbei einem Objekt in der objektorientierten Programmierung insoweit, als dass die Zustände gekapselt und von außen nicht einsehbar sind.

Abstraktion Abstrakte Modelle bestehen aus einer Schnittstelle sowie der Modellbeschreibung. Auf die Modellbeschreibung kann von außen nur durch die Schnittstelle zugegriffen werden. An dieser Stelle wird die Verbindung zur *Modularität* deutlich: Die Schnittstelle kapselt die Modellbeschreibung von der Umwelt ab. Hierfür bestehen die Schnittstellen aus *cuts*, die die Variablen, welche von außen zugänglich sein

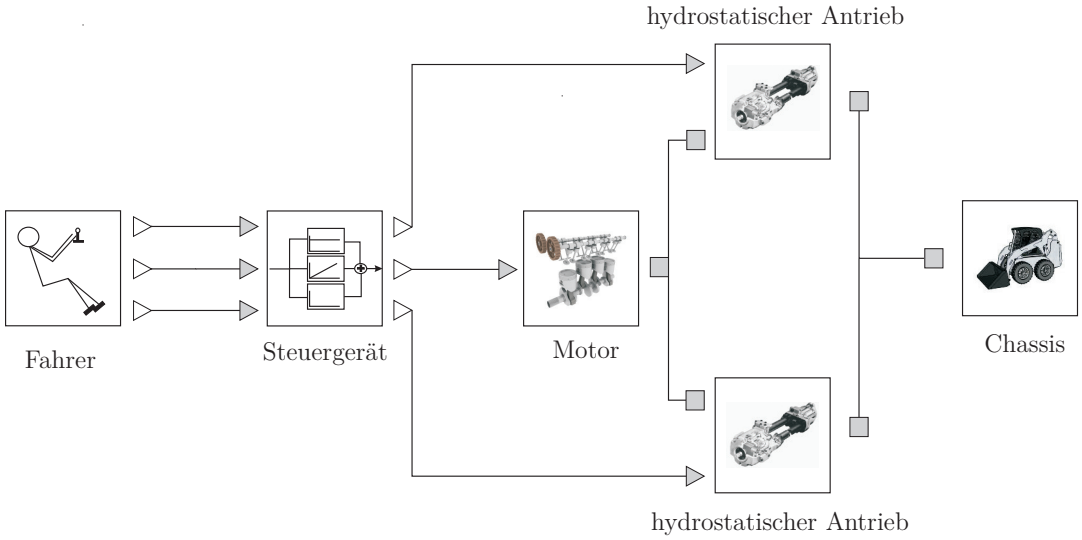


Abbildung 2.2: Objektdiagramm des Skid-Steer-Loaders

sollen, beinhalten. Bei der Auswahl dieser Variablen muss beachtet werden, dass die Energiebilanz an der Schnittstelle für die Verknüpfung mit einem beliebigen anderen Modell stimmt. Meist wird dafür auf das Konzept von *across* und *through* Variablen zurück gegriffen. Der Energiefluss kann stets durch ein Variablenpaar beschrieben werden, deren Produkt die momentane Leistung beschreibt [Wellstead, 1979]. Dieses Variablenpaar besteht aus einer *across* Variable und einer *through* Variable. Die Zuweisung der beiden Klassen erfolgt anhand des Prinzips, mit dem die entsprechende Größe gemessen werden kann. Wird eine Größe zwischen zwei Stellen gemessen, so gehört sie zu den *across*-Variablen, während *through*-Variablen an einer Stelle gemessen werden können. In der Elektrotechnik entspricht die Spannung der *across*-Variablen und der Strom der *through*-Variablen. Die Zuordnung für einige Domänen ist in Tabelle 2.1 angegeben. Werden nun n *cuts* mit *across*-Variablen a_i und *through*-Variablen f_i miteinander verknüpft, so gelten die beiden Verknüfungsregeln

$$a_1 = a_2 = \ldots = a_n \tag{2.3}$$

und

$$\sum_{i=1}^{n} f_i = 0. \tag{2.4}$$

Abstrakte Schnittstellen implizieren, dass die Verbindungen zwischen zwei Modellen den realen Verbindungen im System entsprechen. Somit reflektiert die (grafische) Repräsentation (im Folgenden als Objektdiagramm bezeichnet) die Topologie des Systems. In Abbildung 2.2 ist dies anhand des Objektdiagramms des Skid-Steer-Loaders aus Abbildung 2.3 gezeigt. Der Fahrer steuert die Baumaschine mit einem 2-achsigen Steuerknüppel und einem Pedal. Ein Steuergerät verarbeitet die Eingaben des Bedieners. Der Ausgang des Steuergeräts ist mit dem Verbrennungsmotor sowie den hydrostatischen Antrieben verbunden. Der Verbrennungsmotor wiederum versorgt die beiden hydrostatischen Antriebe mit Leistung. Diese Antriebe befinden sich auf dem Chassis und sind direkt mit den Rädern verbunden. Man beachte dabei die

Abbildung 2.3: Skid-Steer-Loader der Firma Bosch Rexroth AG

verschiedenen Schnittstellen. Die Ausgangssignale des Fahrers und des Steuergeräts haben eine feste Richtung. Die Wirkungsrichtung ist in diesen Fällen also festgelegt. Für diese gerichteten Signale sind die Schnittstellen durch Dreiecke gekennzeichnet. Im Gegensatz dazu kennzeichnen kleine Quadrate physikalische Verbindungen. Die Verbindungen in dem Objektdiagramm entsprechen mit diesen Bezeichnungen genau den echten Verbindungen im realen System.

Klassen In der objektorientierten Modellbildung wird zwischen Klassen und Instanzen unterschieden. Eine Klasse ist eine generische Beschreibung eines Systems, während eine daraus generierte Instanz ein bestimmtes System beschreibt. Bei der Erstellung der Instanz werden den in der Klasse festgelegten Attributen also konkrete Werte zugewiesen. Die Klasse *Fahrzeug* könnte zum Beispiel Attribute wie Fahrzeugmasse, Schwerpunktlage oder Fahrzeugmaße enthalten. Eine Instanz dieser Klasse beschreibt dann ein spezifisches Fahrzeug, wie beispielsweise einen VW Golf. Klassen ermöglichen es, einmal modellierte Systeme mehrfach zu instanziieren.

Vererbung Vererbung bezeichnet den Vorgang von einer Klasse eine Unterklasse abzuleiten. Die Unterklasse erbt dabei alle Attribute der Klasse, von der sie abge-

Domäne	Across	Through
Mechanik	Geschwindigkeit	Kraft
Elektrotechnik	Spannung	Strom
Hydraulik	Druck	Volumenstrom
Pneumatik	Druck	Massenstrom
Thermodynamik	Temperatur	Entropiestrom
Magnetismus	magnetomotorische Kraft	Flussänderung

Tabelle 2.1: Die *across*- und *through*-Variablen für verschiedene Domänen

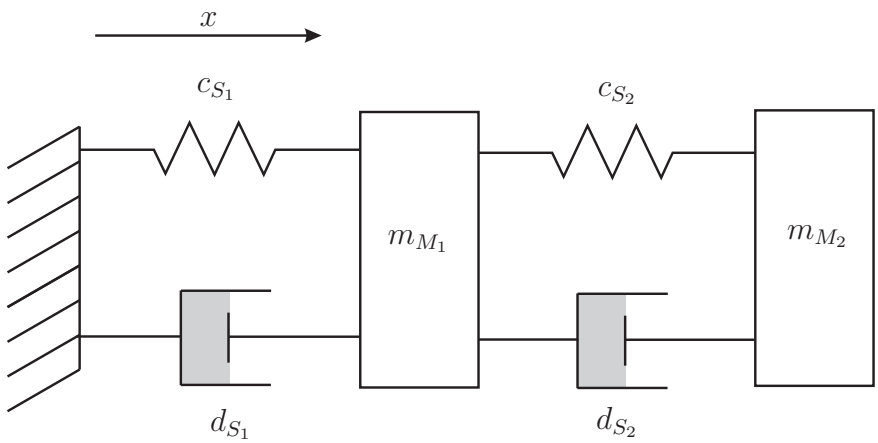
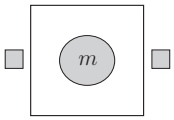


Abbildung 2.4: Zweimassenschwinger

leitet ist. Von der Klasse *Fahrzeug* kann beispielsweise die Klasse *Elektrofahrzeug* abgeleitet werden, welche zusätzliche Attribute wie zum Beispiel Batterie-Kapazität enthält. Auf diese Art und Weise können bestehende Modelle weiterverwendet und spezialisiert werden.

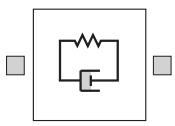
In der Literatur weichen die Eigenschaften, die zur Beschreibung der objektorientierten Modellierung verwendet werden, voneinander ab. Beispielsweise können die Eigenschaften der *Modularität* und der *abstrakten Schnittstellen* auch durch *Verkapselung* ersetzt werden [Cellier, 1996]. Im nun folgenden Beispiel ist das Prinzip der objektorientierten Modellbildung anhand des in Abbildung 2.4 gezeigten Zweimassenschwingers erläutert.

Beispiel 2.1.1. Der Zweimassenschwinger besteht aus zwei Feder-Dämpfer-Elementen, zwei Massen und einem Festlager. In Abbildung 2.5 ist das Objektdiagramm des Zweimassenschwingers (mit den verwendeten Submodellen) gezeigt. Wie zu erkennen ist, werden drei Klassen zur Modellierung benötigt. Jede Klasse besteht hierbei, wie oben beschrieben, aus einer Schnittstelle und einem Satz Gleichungen beziehungsweise Submodellen.



Das Interface der Klasse `Masse` besteht aus zwei Ports, welche jeweils die Variablen für die Position s_i und die Kraft F_i enthalten ($i \in \{a, b\}$). Die Beschleunigung der Masse wird aus dem Impulssatz bestimmt. Der Gleichungssatz lautet dementsprechend

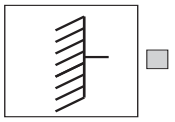
$$m \cdot a = F_a + F_b, \tag{2.5}$$
$$\ddot{x}_a = \ddot{x}_b = a. \tag{2.6}$$



Auch die Klasse `Feder-Dämpfer` besitzt zwei Ports. Im Gegensatz zur Klasse `Masse` wird das Feder-Dämpfer-System jedoch nicht explizit durch Gleichungen beschrieben, sondern besteht aus Submodellen: Dem Modell einer Feder und dem Modell eines Dämpfers. Hieraus ergibt sich der entsprechende Gleichungssatz für das gesamte Feder-Dämpfer-

System zu

$$x_{rel} = x_b - x_a, \tag{2.7}$$
$$F_b = F, \tag{2.8}$$
$$F_a = -F, \tag{2.9}$$
$$v_{\text{rel}} = \dot{x}_{\text{rel}}, \tag{2.10}$$
$$F = c \cdot x_{\text{rel}} + d \cdot v_{\text{rel}}, \tag{2.11}$$



Im Gegensatz zu den beiden vorherigen Klassen besitzt die Klasse `Lager` nur einen Port, der die Variable für die Position s und die Variable für die Kraft F enthält. Zur Beschreibung des Lagers ist lediglich die Gleichung

$$x = x_0 \tag{2.12}$$

notwendig, wobei x_0 die Position des Lagers angibt. Werden nun zwei Massen M_1 und M_2, zwei Feder-Dämpfer-Systeme S_1 und S_2 sowie ein festes Lager L instantiiert, so erhält man die gesammelten Objektgleichungen

$$m_{M_1} \cdot a_{M_1} = F_{M_1,a} + F_{M_1,b}, \tag{2.13}$$
$$\ddot{x}_{M_1,a} = \ddot{x}_{M_1,b} = a_{M_1}, \tag{2.14}$$
$$m_{M_2} \cdot a_{M_2} = F_{M_2,a} + F_{M_2,b}, \tag{2.15}$$
$$\ddot{x}_{M_2,a} = \ddot{x}_{M_2,b} = a_{M_2}, \tag{2.16}$$
$$x_{S_1,\text{rel}} = x_{S_1,b} - x_{S_1,a}, \tag{2.17}$$
$$F_{S_1,b} = F_{S_1}, \tag{2.18}$$
$$F_{S_1,a} = -F_{S_1}, \tag{2.19}$$
$$v_{S_1,\text{rel}} = \dot{x}_{S_1,\text{rel}}, \tag{2.20}$$
$$F_{S_1} = c_{S_1} \cdot x_{S_1,\text{rel}} + d_{S_1} \cdot v_{S_1,\text{rel}}, \tag{2.21}$$
$$x_{S_2,\text{rel}} = x_{S_2,b} - x_{S_2,a}, \tag{2.22}$$
$$F_{S_2,b} = F_{S_2}, \tag{2.23}$$
$$F_{S_2,a} = -F_{S_2}, \tag{2.24}$$
$$v_{S_2,\text{rel}} = \dot{x}_{S_2,\text{rel}}, \tag{2.25}$$
$$F_{S_2} = c_{S_2} \cdot x_{S_2,\text{rel}} + d_{S_2} \cdot v_{S_2,\text{rel}}, \tag{2.26}$$
$$x_L = x_{L,0}. \tag{2.27}$$

Das Verbinden der Objekte in einem Objektdiagramm zu dem Zweimassenschwinger,

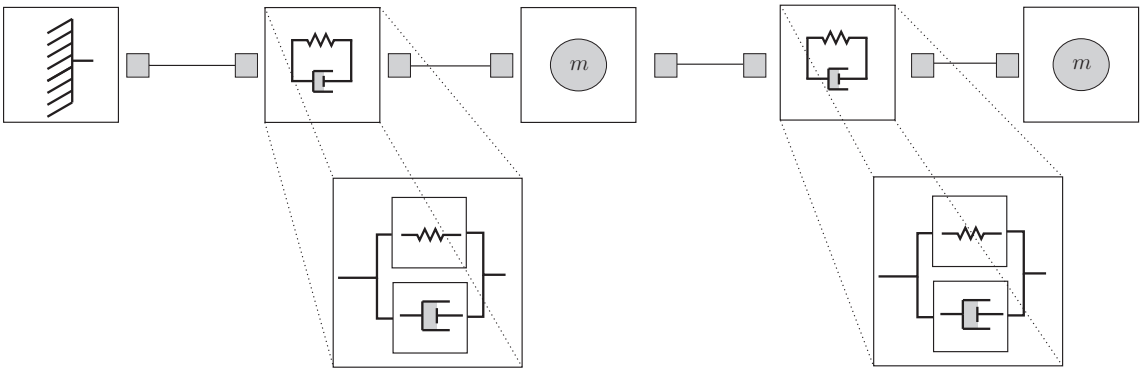


Abbildung 2.5: Objektdiagramm des Zweimassenschwingers mit Submodellen

liefert gemäß Gleichung (2.3) die Verbindungsgleichungen

$$F_L + F_{S_1,a} = 0, \tag{2.28}$$
$$x_{S_1,a} = x_L, \tag{2.29}$$
$$F_{M_1,a} + F_{S_1,b} = 0, \tag{2.30}$$
$$x_{S_1,b} = x_{M_1,a}, \tag{2.31}$$
$$F_{M_1,b} + F_{S_2,a} = 0, \tag{2.32}$$
$$x_{S_2,a} = x_{M_1,b}, \tag{2.33}$$
$$F_{M_2,a} + F_{S_2,b} = 0, \tag{2.34}$$
$$x_{S_2,b} = x_{M_2,a}, \tag{2.35}$$
$$F_{M_2,b} = 0. \tag{2.36}$$

Die letzte Gleichung zeigt hier an, dass das Objekt M_2 mit keinem weiteren Objekt verbunden ist. Die Gleichungen (2.13)-(2.36) bilden das mathematische Modell des Zweimassenschwingers. Schon in diesem einfachen Beispiel ist zu sehen, dass objektorientierte Modellbildung zunächst auf sehr umfangreiche Modelle führt, deren Lösung ohne vorherige Zusammenfassung sehr ineffizient ist. Dieses unvereinfachte Modell wird auch *flaches Modell* genannt. Auf die Vereinfachung flacher Modelle wird im nächsten Abschnitt eingegangen. Das flache Modell ist im Allgemeinen eine DAE. Dabei ist eine DAE formal folgendermaßen definiert:

Es sei $\Omega \subseteq \mathbb{R}^N \times \mathbb{R}^N$ *eine offene Menge und* $\mathcal{I} \subset \mathbb{R}$ *ein Intervall. Für* $\mathbf{F} : \Omega \times \mathcal{I} \mapsto \mathbb{R}^N$ *heißt*

$$\mathbf{F}(\dot{\mathbf{x}}, \mathbf{x}, t) = \mathbf{0} \tag{2.37}$$

dann DAE, falls $\frac{\partial \mathbf{F}}{\partial \dot{\mathbf{x}}}$ *singulär ist.*

Hierbei sei $\mathbf{F}$ hinreichend oft differenzierbar. Im Fall

$$det(\frac{\partial \mathbf{F}}{\partial \dot{\mathbf{x}}}) \neq 0 \tag{2.38}$$

könnte Gleichung (2.37) zumindest theoretisch nach $\dot{\mathbf{x}}$ aufgelöst werden. Das Ergebnis wäre wieder eine explizite ODE. Es ist zu beachten, dass eine DAE auch in der Form

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t), \tag{2.39}$$
$$\mathbf{0} = \mathbf{g}(\mathbf{x}, t) \tag{2.40}$$

gegeben sein kann. In diesem Fall spricht man von einer *semi-expliziten DAE*, während die Gleichung (2.37) als *implizite DAE* bezeichnet wird. In Abschnitt 2.2 wird auf die Lösung und Klassifikation von DAEs eingegangen.

2.1.2 Symbolische Manipulation objektorientierter Modelle

Im vorherigen Abschnitt wurde dargelegt, wie unter Verwendung von objektorientierten Prinzipien ein mathematisches Modell eines physikalischen Systems generiert werden kann. Diese Modelle sind in der Regel komplexe DAEs [Cellier, 1991; Otter, 1994]. Dabei bedeutet komplex, dass sowohl die Dimension als auch die Anzahl der auftretenden Terme sehr groß ist. Daher ist es nicht empfehlenswert, diese Modelle ohne vorherige Vereinfachung zu lösen. Es existieren verschiedene Algorithmen, um komplexe DAEs automatisiert zu vereinfachen [Carpanzano & Maffezzoni, 1998; Otter, 1994]. Das grundsätzliche Vorgehen ist jedoch in der Regel dasselbe.

Zunächst werden die Gleichungen der DAE umsortiert, um eine einfachere Struktur zu erhalten. Hierbei können auch Teile der DAE voneinander entkoppelt werden. Anschließend wird im so genannten *Tearing* versucht, die Anzahl der Iterationsvariablen im Newton-Verfahren zu senken. Diese beiden Algorithmen werden im Folgenden kurz umrissen.

Zusätzlich zu diesen Basis-Algorithmen existieren noch weitere Verfahren, die ein noch effizienteres Lösen von komplexen DAEs erlauben. Diese Verfahren versuchen in der Regel das Systemwissen des Ingenieurs zu nutzen. Ein Beispiel hierfür ist der *Relaxations*-Algorithmus. Dieser führt die Gauss-Elimination zur Lösung der linearen Gleichungssysteme im Newton-Verfahren symbolisch durch, ohne zu pivotisieren. Anstatt der Pivotisierung kann die Reihenfolge der Elimination schon während der Modellbildung beeinflusst werden [Otter *et al.*, 1996]. Auf diese Art und Weise können in vielen Fällen lineare Gleichungssysteme automatisiert symbolisch gelöst werden.

Sortieren der Gleichungen

Ein wichtiger Schritt, um die Lösung der komplexen DAE effizienter zu gestalten, ist die Aufteilung in kleinere, von einander unabhängige Subsysteme. Hierzu wird

zunächst die so genannte *Inzidenzmatrix* eingeführt, die anzeigt, welche Variablen in welcher Gleichung auftauchen. Es wird dabei davon ausgegangen, dass die DAE wieder in derselben Form wie in Gleichung (2.37) vorliegt. Die Zeilen der Inzidenzmatrix $\mathbf{I} \in \mathbb{R}^{N \times N}$ repräsentieren dann die Gleichungen, während die Spalten die Variablen repräsentieren. Taucht nun x_j oder $\dot{x}_j$ in der i-ten Gleichung auf, so ist $\mathbf{I}_{i,j}$ ungleich Null, andernfalls Null. Ziel ist es, die Inzidenzmatrix unter Verwendung von Permutationsmatrizen $\mathbf{P}$ und $\mathbf{Q}$ auf die Gestalt

$$\mathbf{P} \cdot \mathbf{I} \cdot \mathbf{Q} = \begin{bmatrix} \tilde{\mathbf{I}}_{1,1} & & & & \\ \tilde{\mathbf{I}}_{2,1} & \tilde{\mathbf{I}}_{2,2} & & & \\ \tilde{\mathbf{I}}_{3,1} & \tilde{\mathbf{I}}_{3,2} & \tilde{\mathbf{I}}_{3,3} & & \\ \vdots & \vdots & \vdots & \ddots & \\ \tilde{\mathbf{I}}_{m,1} & \tilde{\mathbf{I}}_{m,2} & \dots & \dots & \tilde{\mathbf{I}}_{m,m} \end{bmatrix} \tag{2.41}$$

mit Diagonalblöcken von $\tilde{\mathbf{I}}_{i,i}$ minimaler Dimension zu transformieren. Hierbei ist $\mathbf{P}$ für die Vertauschung der Gleichungen und $\mathbf{Q}$ für die Vertauschung der Variablen zuständig. Die Zeilen in Gleichung (2.41) sind nur von den darüber stehenden Zeilen abhängig. Somit können die entsprechenden Subsysteme der DAE in einer Vorwärtsrekursion sukzessive nacheinander gelöst werden. Dieses Vorgehen ist natürlich viel effizienter als die ursprüngliche DAE der Dimension N zu lösen. Darüber hinaus kann das im nächsten Abschnitt erläuterte Tearing auf jeden einzelnen Block angewendet werden.

Zum Berechnen der Blockgestalt existieren verschiedene Algorithmen. In der Regel wird die Berechnung in zwei Schritten vollzogen. Zunächst werden die Zeilen der Inzidenzmatrix so permutiert, dass die Einträge auf der Diagonalen ungleich Null sind. Der Aufwand hierfür beträgt $\mathcal{O}(n\tau)$, wobei τ die Anzahl der von Null verschiedenen Elemente von $\mathbf{I}$ ist [Duff *et al.*, 1986]. Erst im zweiten Schritt wird die eigentliche Blockgestalt berechnet. Meist wird hierfür auf Tarjan's Algorithmus aus der Graphentheorie zurückgegriffen. Dieser benötigt $\mathcal{O}(n) + \mathcal{O}(\tau)$ Operationen. Insgesamt beträgt der Aufwand im schlimmsten Fall also $\mathcal{O}(n^2)$ Operationen [Tarjan, 1972]. In der Praxis ist der Aufwand jedoch meist deutlich kleiner, da τ in der Regel sehr viel kleiner als n ist.

Tearing

Die Idee des Tearings ist es, eine komplexe DAE in einen explizit lösbaren und einen impliziten Teil aufzuteilen. Diejenigen Variablen, die im Newton-Verfahren iteriert werden, sind dann nur noch diejenigen, welche nicht im explizit lösbaren Teil auftauchen. Um das Problem des Tearings zu formalisieren, wird Gleichung (2.37) geschrieben als

$$\tilde{\mathbf{F}}(\dot{\mathbf{y}}, \mathbf{y}, \mathbf{z}, t) = \mathbf{0}. \tag{2.42}$$

Hierbei bezeichnet $\mathbf{y} \in \mathbb{R}^{N_y}$ den Vektor der Variablen, die sowohl algebraisch als auch differentiell in $\mathbf{F}$ enthalten sind und $\mathbf{z} \in \mathbb{R}^{N_z}$ den Vektor der Variablen, die nur

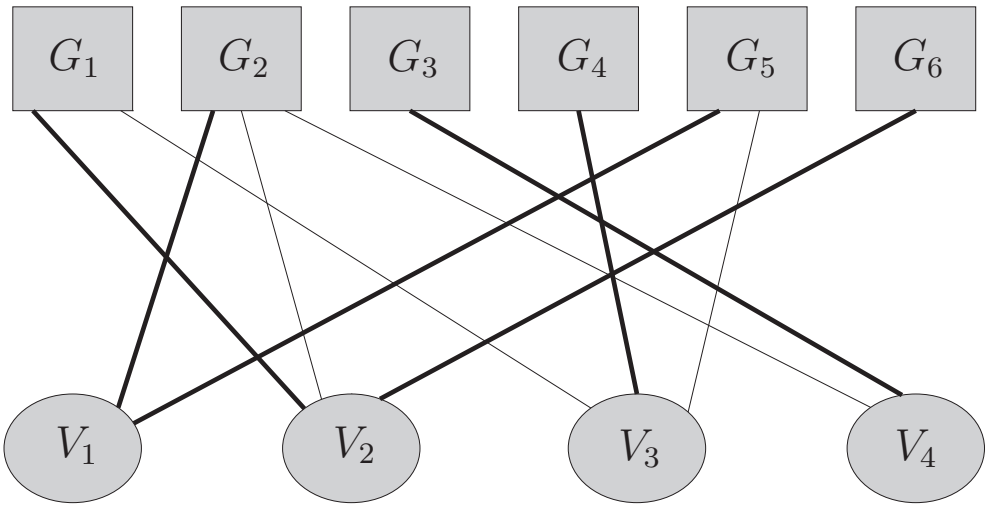


Abbildung 2.6: Strukturgraph des Beispiels

algebraisch auftreten. Die Aufgabe eines Tearing-Algorithmus ist es, einen Untervektor $\tilde{\mathbf{z}}$ minimaler Dimension $N_{\tilde{z}}$ zu finden, so dass die Gleichung (2.42) geschrieben werden kann als

$$z_1 = g_1(\dot{\mathbf{y}}, \mathbf{y}, \tilde{\mathbf{z}}, t) \tag{2.43}$$
$$z_2 = g_2(\dot{\mathbf{y}}, \mathbf{y}, \tilde{\mathbf{z}}, z_1, t) \tag{2.44}$$
$$\vdots \tag{2.45}$$
$$z_k = g_k(\dot{\mathbf{y}}, \mathbf{y}, \tilde{\mathbf{z}}, z_1, \ldots, z_{k-1}, t) \tag{2.46}$$
$$\tilde{\mathbf{F}}(\dot{\mathbf{y}}, \mathbf{y}, \tilde{\mathbf{z}}, z_1, \ldots, z_k, t) = \mathbf{0}. \tag{2.47}$$

Hierbei sind $g_1, \ldots, g_k$ geeignete skalare Funktionen und $\tilde{\mathbf{F}} \in \mathbb{R}^{N-k}$ die verbleibende vektorwertige Funktion. Die Funktionen g_j erhält man, indem in einer Gleichung von (2.42) z_j abhängig von $\dot{\mathbf{y}}, \mathbf{y}, \tilde{\mathbf{z}}, z_1, \ldots, z_{j-1}$ und t isoliert wird. Würden $z_1, \ldots, z_k$ nun in $\tilde{\mathbf{F}}$ eingesetzt, so entstünde eine äquivalente DAE mit der reduzierten Dimension $N - k$. In der Praxis setzt man die isolierten Variablen jedoch nicht ein, da diese mehrfach in $\tilde{\mathbf{F}}$ auftauchen können und so die Komplexität der DAE unnötig steigen würde. Anstatt dessen werden zunächst die skalaren Funktionen $g_1, \ldots, g_k$ und anschließend erst $\tilde{\mathbf{F}}$ ausgewertet.

In [Carpanzano, 2000] wird gezeigt, dass das Problem des Tearings NP-vollständig ist. Es existiert also kein deterministischer Algorithmus polynomialer Laufzeit für diese Aufgabe. Folglich muss man sich mit Algorithmen begnügen, die einem echten Tearing-Algorithmus möglichst nahe kommen [Elmqvist & Otter, 1994]. Hierzu wird die reduzierte Inzidenzmatrix $\mathbf{R} \in \mathbb{R}^{n \times n_z}$ eingeführt, welche im Gegensatz zur Inzidenzmatrix nur die n_z algebraischen Variablen berücksichtigt. Ein weiterer Unterschied zur Inzidenzmatrix besteht darin, dass die reduzierte Inzidenzmatrix anzeigt, welche Gleichung nach welcher Variable aufgelöst wereden kann. Hierzu sind die entsprechenden Einträge fett gedruckt. Lässt sich die i-te Gleichung nach der Variablen z_j auflösen, so ist der entsprechende Eintrag fett. Es ist leicht zu entscheiden, ob eine Gleichung nach einer Variablen aufgelöst werden kann. Es ist jedoch schwierig zu

entscheiden, ob dies im Allgemeinen auch getan werden sollte, um eine vereinfachte DAE zu erhalten, die effizient gelöst werden kann. Daher werden häufig Heuristiken verwendet, um diese Frage zu beantworten. Diese Heuristiken sind maßgeblich für die Effizienz eines Tearing-Algorithmus.

Sei nun eine DAE mit sechs Gleichungen gegeben, in denen vier Variablen entsprechend dem Graphen in Abbildung 2.6 auftauchen, wobei $G_1, \ldots, G_6$ die sechs Gleichungen und $V_1, \ldots V_4$ die vier Variablen bezeichnen. Eine Verbindungslinie zwischen G_i und V_j bedeutet dabei, dass V_j in G_i auftaucht. Eine fette Verbindungslinie bedeutet, dass die Gleichung nach der entsprechenden Variablen aufgelöst werden kann. Die Inzidenzmatrix zu dem Graphen ist dann gegeben durch

$$\mathbf{R} = \begin{bmatrix} 0 & \mathbf{X} & \mathrm{X} & 0 \\ \mathbf{X} & \mathrm{X} & 0 & \mathrm{X} \\ 0 & 0 & 0 & \mathbf{X} \\ 0 & 0 & \mathbf{X} & 0 \\ \mathbf{X} & 0 & \mathrm{X} & 0 \\ 0 & \mathbf{X} & 0 & 0 \end{bmatrix}. \tag{2.48}$$

Mit diesen Begrifflichkeiten kann das Problem des Tearing auch folgendermaßen formuliert werden:

Sei eine Inzidenzmatrix $\mathbf{R}$ gegeben. Finde das größtmögliche k, so dass durch Umordnung der Zeilen und Spalten die Submatrix $\tilde{\mathbf{R}}_{1\ldots k,1\ldots k}$ der neuen Matrix $\tilde{\mathbf{R}}$ eine untere Dreiecksmatrix ist. Dabei sollen die Diagonalelemente von $\tilde{\mathbf{R}}_{1\ldots k,1\ldots k}$ fett gedruckt sein.

Tearing kann noch effizienter gestaltet werden, indem zugelassen wird, dass die Gleichungen (2.43) bis (2.46) nicht nur Zuweisungen enthalten. So ist es beispielsweise möglich, hier lineare Gleichungssysteme mit einzubeziehen. Diese können dann (zumindest bei geringer Dimension) symbolisch gelöst werden.

2.1.3 Kausalität

Der Begriff Kausalität bezeichnet die Beziehung zwischen Ursache und Wirkung. In einem kausalen Modell wird jede Wirkung durch eine Ursache ausgelöst, wobei die Ursache zeitlich immer vor der Wirkung einsetzt.

Wie schon zu Beginn des Abschnitts erwähnt, galt es bis zum Aufkommen der objektorientierten Modellierung als natürlich, dass physikalische Systeme als ODE in der Zustandsraumdarstellung

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t) \tag{2.49}$$

modelliert wurden. Dieser Modellierungsansatz beschreibt stets kausale Zusammenhänge, wobei die rechte Seite als Ursache und die linke Seite als Wirkung angesehen werden kann. Dies wird nochmals in der Implementierung der Lösungsverfahren deutlich. Hier steht in jeder Gleichung genau eine Variable auf der linken Seite und die Auswertung der rechten Seite liefert den Wert, der dieser Variablen zugewiesen

wird. Dementsprechend bestanden die mathematischen Modelle in den frühen Modellierungsprachen nicht wirklich aus Gleichungen, sondern aus Zuweisungen und besaßen somit eine vorgefertigte Kausalität.

In der Folge musste zwangsläufig während der Modellbildung eine Kausalität festgelegt werden. Im Falle einer Feder musste also entschieden werden, ob die Auslenkung der Feder die Federkraft erzeugt (Gleichung (2.1)) oder die Federkraft die Feder zusammendrückt (Gleichung (2.2)). Beide Möglichkeiten beschreiben dabei natürlich denselben physikalischen Effekt. Das Modell der Feder konnte jedoch lediglich in dem Fall wiederverwendet werden, dass wieder die gleiche Kausalität angenommen wurde. In der Folge war die Wiederverwertbarkeit der Modelle offensichtlich stark eingeschränkt.

Die objektorientierte Modellbildung generiert mathematische Modelle, die aus Gleichungen und nicht aus Zuweisungen bestehen und somit akausal sind. Akausale Modelle sind intuitiver und einfacher wiederzuverwenden als kausale, haben jedoch auch einen großen Nachteil: Die numerische Integration akausaler Modelle ist in der Regel sehr ineffektiv. Daher wird bei akausalen Modellen zum Kompilierungszeitpunkt (soweit möglich) eine Kausalität festgelegt. Diese Festlegung der Berechnungsreihenfolge geschieht durch die Transformation der Inzidenzmatrix auf Blockgestalt sowie im Tearing. Ist die Matrix $\tilde{\mathbf{R}}$ aus dem Tearing eine untere Dreiecksmatrix (also $n = k$), so ist der Störungsindex $\nu_p = 0$ und das mathematische Modell ist eine ODE. In diesem Fall ist die Kausalität, wie zu erwarten war, eindeutig festgelegt.

2.1.4 Die Modelica-Sprache

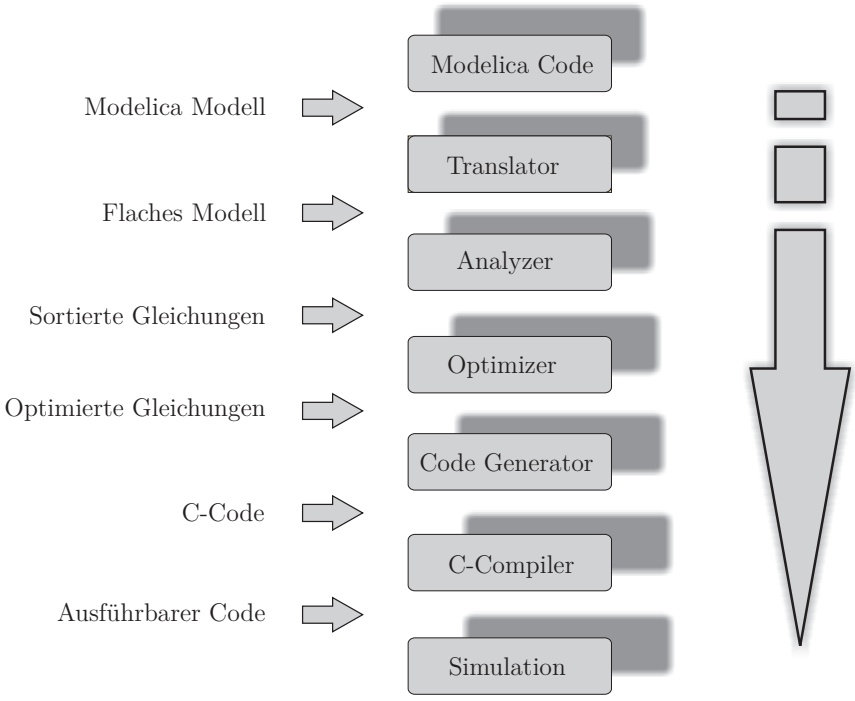


Abbildung 2.7: Struktur des OpenModelica Compilers

Die Konzepte der objektorientierten Modellierung wurden, wie in Abschnitt 2.1 beschrieben, schon Ende der siebziger Jahre entwickelt. Populär wurde die objektorientierte Modellierung jedoch erst Anfang der neunziger Jahre als auch Standard-Computer genug Rechenleistung aufbrachten, um komplexe Modelle zu verarbeiten. Einer der großen Vorteile kam jedoch noch nicht zur Entfaltung: Da eine Vielzahl an verschiedenen Modellierungssprachen wie zum Beispiel ASCEND, Dymola, gPROMS, NMF, ObjectMath, Omola, SIDOPS+, Smile, ALLAN oder VHDL-AMS verwendet wurde, konnte die Wiederverwendbarkeit nicht zum Tragen kommen.

Im Rahmen des europäischen ESPRIT-Programmes kam im September 1996 die *Simulation in Europe Basic Research Workung Group* (SiE-WG) zusammen. Das ursprüngliche Ziel, einen Beitrag zur möglichen Vereinigung existierender objektorientierter Modellierungssprachen zu veröffentlichen, wurde schnell um das Ziel erweitert, eine neue einheitliche Modellierungssprache zu entwickeln.

Diese neue Sprache wurde *Modelica* genannt [Fritzson & Engelson, 1998]. Ein Jahr später, im September 1997, wurde die erste Version der Sprachenbeschreibung veröffentlicht. Im Februar 2000 wurde die Modelica Association gegründet. Diese Nonprofit-Organisation treibt seitdem die Weiterentwicklung der Sprache sowie die Entwicklung von Standardbibliotheken voran. Heute ist eine große Modelica Standardbibliothek verfügbar und es existieren verschiedene Modelica Compiler. Diese Compiler sind zum Teil mit einer grafischen Benutzeroberfläche verbunden und werden kommerziell vertrieben und gewartet. Beispiele hierfür sind Dymola [Brück *et al.*, 2002], MapleSim [Zheng-Da *et al.*, 2009], SimulationX oder MathModelica [Fritzson *et al.*, 2002b]. Diese kommerziellen Simulationsumgebungen sind in der Industrie inzwischen sehr weit verbreitet.

Der Grund dafür, dass Modelica in relativ kurzer Zeit so populär geworden ist, liegt daran, dass die objektorientierten Prinzipien in Modelica konsequent umgesetzt sind. Folglich lassen sich auf einfache Art und Weise komplexe mechatronische Systeme modellieren. Dabei sind die Modelle einfach zu warten und können durch die grafische Oberfläche auch von Laien wiederverwendet werden. Darüber hinaus besitzt Modelica einen sehr einfachen Syntax. Dies erlaubt es den Benutzern, die Modelica Standardbibliotheken auf einfache Art und Weise durch eigene Modelle zu erweitern.

Neben den kommerziellen Simulationsumgebungen, die auf Modelica basieren, existiert auch ein Open Source Modelica Compiler [Fritzson *et al.*, 2002a]. Die Weiterentwicklung des OpenModelica Compilers wird durch das OpenModelica Konsortium überwacht und vorangetrieben.

Die Struktur des OpenModelica Compilers ist in Abbildung 2.7 dargestellt [Fritzson *et al.*, 2006]. Der *Translator* erzeugt aus dem Modelica Modell das flache Modell. Der *Analyzer* und der *Optimizer* wenden die in Abschnitt 2.1.2 beschriebenen Techniken an, um das flache Modell zu vereinfachen. Anschließend erzeugt der *Code Generator* aus den vereinfachten Gleichungen einen C-Code und bindet ein numerisches Lösungsverfahren ein. Für die Simulation wird der C-Code dann kompiliert und ausgeführt.

Diese Arbeit ist darüber hinaus Grundlage für ein Plug-in für den OpenModelica Compiler, welches auf den vereinfachten Gleichungen eine gleichungsbasierte Mo-

dellreduktion durchführen kann. Eine Beschreibung dieses Plug-ins sowie eine Untersuchung der Leistungsfähigkeit ist in Kapitel 4 gegeben.

2.2 Differential-algebraische Gleichungen

In den sechziger Jahren kamen die ersten Modellierungssprachen für dynamische Systeme auf. In dieser Zeit bestanden die mathematischen Modelle dynamischer Systeme in der Regel aus Systemen von gewöhnlichen Differentialgleichungen (ODEs), da ODEs die übliche Beschreibungsform für dynamische Systeme darstellte [Strauss *et al.*, 1967]. Erst ab Mitte der achtziger Jahre wurde erkannt, dass die Modellierung akausaler Effekte in diesen Modellierungssprachen zu Problemen führt. In [Cellier *et al.*, 1995] werden die Vorteile der akausalen Modellbildung, welche auf DAEs führt, dargelegt. Bekannt sind DAEs jedoch schon seit Anfang der sechziger Jahre [Dolezal, 1960; Gantmacher, 1959]. Beispielsweise sind die von Dolezal zur Untersuchung von Einschwingvorgängen bei linearen Systemen untersuchten *Integrodifferentialgleichungen* nichts anderes als lineare DAEs. Zunächst wurden DAEs jedoch als eine besondere Klasse von ODEs betrachtet. Folglich wurden numerische Integrationsverfahren für ODEs übernommen und für DAEs verwendet. In den frühen achtziger Jahren ließen sich jedoch zunehmend Ergebnisse nicht mit der Theorie von ODEs vereinbaren. In [Petzold, 1982] wird gezeigt, dass sich DAEs grundlegend von ODEs unterscheiden. So können beispielsweise numerische Integrationsverfahren für ODEs nicht immer auch zur Lösung von DAEs eingesetzt werden, auch wenn diese Herangehensweise auf den ersten Blick erfolgversprechend scheint. Darüber hinaus müssen zur Lösung von DAEs zunächst konsistente Anfangswerte ermittelt werden, was im Allgemeinen kein einfaches Problem ist [Vieira & J., 2000].

In den zwanzig folgenden Jahren waren DAEs ein beliebtes Forschungsgebiet, was sich in den zahlreichen Veröffentlichungen aus dieser Zeit widerspiegelt. Insbesondere mit Blick auf semi-explizite DAEs erscheint es zunächst erfolgversprechend, implizite Verfahren zur Lösung von ODEs auch zur Lösung von DAEs zu nutzen. Das folgende Beispiel zeigt, dass diese Vorgehensweise jedoch nicht immer funktioniert [Ascher & Petzold, 1998]. Danach wird der Indexbegriff zur Klassifikation von DAEs eingeführt.

Beispiel 2.2.1. Es sei die DAE

$$\dot{x}_1 = x_2(t), \tag{2.50}$$
$$0 = x_1(t) - g(t) \tag{2.51}$$

mit einer bekannten Funktion $g(t)$ in semi-expliziter Darstellung gegeben. Wendet man nun das implizite Euler-Verfahren an, so lauten die diskretisierten Gleichungen

$$x_{2,n+1} = \frac{x_{1,n+1} - x_{1,n}}{h_n} = \frac{g(t_{n+1}) - g(t_n)}{h_n}, \tag{2.52}$$
$$x_{1,n+1} = g(t_{n+1}), \tag{2.53}$$

wobei $x_{i,n}$ den Wert von x_i und h_n jeweils die Schrittweite zum Zeitpunkt t_n bezeichnet ($1 \leq i \leq 2$). Wird weiterhin eine Schrittweitensteuerung verwendet, die

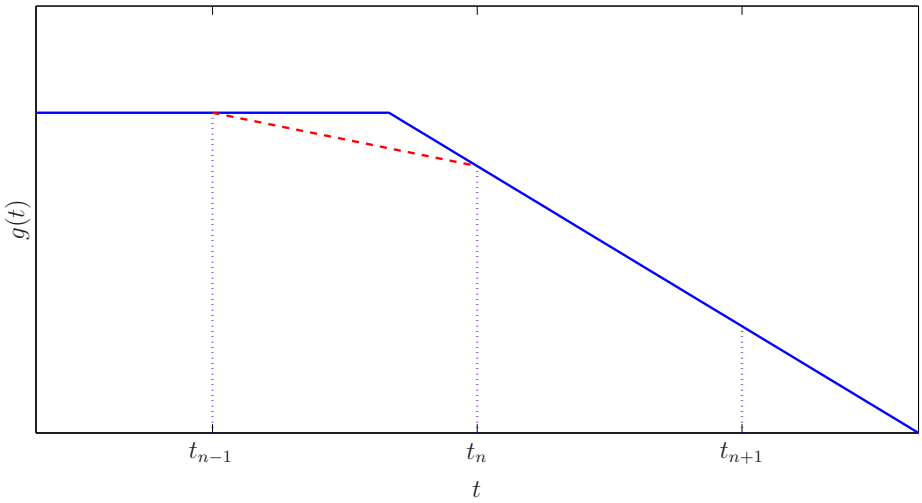


Abbildung 2.8: Die Steigung der gestrichelte Linie entspricht dem Differenzenquotienten in Gleichung (2.59), während $\dot{g}(t_n)$ offensichtlich der Steigung von g zwischen t_n und t_{n+1} entspricht

dafür sorgt, dass lokal eine vorgegebene Fehlerschranke eingehalten wird, so wird nach jedem Zeitschritt der entstandene Fehler geschätzt. Hierzu verwendet man in der Regel eine Näherung für die Differenz zwischen exakter und berechneter Lösung. Im Falle des impliziten Euler-Verfahrens kann die Schätzung $\mathbf{d}_{n+1}$ dieses Fehlers mit Hilfe der dividierten Differenzen zum Zeitpunkt t_{n+1} angegeben werden [Bauer, 1999]. Betrachtet man nur die zweite Komponente von $\mathbf{d}_{n+1}$, so ergibt sich:

$$d_{2,n+1} = h_{n+1}(h_{n+1} + h_n)\nabla^2 x_{2,n+1} \tag{2.54}$$

$$= h_{n+1}(h_{n+1} + h_n)\left(\frac{\frac{x_{2,n+1}-x_{2,n}}{h_{n+1}} - \frac{x_{2,n}-x_{2,n-1}}{h_n}}{h_{n+1}+h_n}\right) \tag{2.55}$$

$$= \frac{g(t_{n+1}) - g(t_n)}{h_{n+1}} - \frac{g(t_n) - g(t_{n-1})}{h_n} - h_{n+1} \cdot r \tag{2.56}$$

mit

$$r = \frac{\frac{g(t_n)-g(t_{n-1})}{h_n} - \frac{g(t_{n-1})-g(t_{n-2})}{h_{n-1}}}{h_n}. \tag{2.57}$$

Hierbei ist $\nabla^2 x_{n+1}$ die zweite dividierende Differenz von x_{n+1}. Betrachtet man nun diese Fehlerschätzung für $h_{n+1} \to 0$, so erhält man

$$\lim_{h_{n+1}\to 0} d_{2,n+1} = \lim_{h_{n+1}\to 0} \frac{g(t_{n+1}) - g(t_n)}{h_{n+1}} - \frac{g(t_n) - g(t_{n-1})}{h_n} \tag{2.58}$$

$$= \dot{g}(t_n) - \frac{g(t_n) - g(t_{n-1})}{h_n}. \tag{2.59}$$

Sei g nun eine Funktion, deren Sekante zwischen den Zeitpunkten t_{n-1} und t_n eine andere Steigung aufweist als die Funktion im Punkt t_n (siehe Abb. 2.8), dann kann

der Wert Fehlerschätzung $\mathbf{d}_{n+1}$ durch Verkleinern der Schrittweite nicht beliebig verringert werden. In der Folge bricht die Integration ab, da es so scheint, als ob die vorgegebene Fehlerschranke nicht eingehalten werden kann. Eine Möglichkeit, diesem Problem zu begegnen, ist der Ausschluss der Variabeln aus der Fehlerschätzung, die nur algebraisch auftreten. Alternativ kann der Fehler der Variablen, die differentiell und denjenigen, welche nur algebraisch auftreten, auch unterschiedlich gewichtet werden. Diese Technik ist beispielsweise in der Implementierung RADAU5 des numerischen Integrationsverfahren Radau-IIa umgesetzt [Hairer & Wanner, 1991].

Im nächsten Abschnitt wird auf die Klassifikation von DAEs eingegangen und erläutert, welche Systeme besonders große Schwierigkeiten bei der Lösung bereiten.

2.2.1 Indexbegriffe für DAEs

In der Regel werden DAEs anhand ihres Indexes klassifiziert. In der Literatur existieren viele verschiedene Indexbegriffe, die in der Regel nicht äquivalent zueinander sind. Alle Indexbegriffe haben jedoch den Anspruch ein Maß dafür darzustellen, wie sehr sich die vorliegende DAE von einer ODE unterscheidet. Da mit zunehmender *Entfernung* einer DAE von einer ODE auch die Schwierigkeit steigt, die DAE numerisch zu lösen, zeigt der Index auch an, ob und mit wie großen numerischen Schwierigkeiten zu rechnen ist. Einen guten Überblick über die hier verwendeten, verwandte Indexbegriffe bieten beispielsweise [Griepentrog *et al.*, 1992; Le Vey, 1998]. In diesem Abschnitt werden die beiden wichtigsten Indexbegriffe für DAEs eingeführt. Dies sind zum einen der differentielle Index und zum anderen der Störungsindex.

Differentieller Index

Der *differentielle Index* ist der wohl verbreitetste und somit auch bekannteste Index. Ist in der Literatur ohne nähere Erläuterungen nur vom Index die Rede, so ist in den meisten Fällen der differentielle Index gemeint. Die Idee hinter diesem Index ist es, die Anzahl von Differentiationen zu zählen, die notwendig sind, um die vorliegende DAE in eine ODE zu transformieren, also um $\dot{\mathbf{x}}$ eindeutig aus $\mathbf{x}$ und t zu bestimmen. Gewöhnliche Differentialgleichungen sind folglich Index-0-Probleme. Formal lautet die Definition dann:

> *Der differentielle Index ν_d der DAE* (2.37) *ist die kleinste natürliche Zahl p, so dass $\dot{\mathbf{x}}$ eindeutig als eine stetige Funktion in Abhängigkeit von $\mathbf{x}$ und t aus dem überbestimmten System*
>
> $$\mathbf{F}(\mathbf{x}, \dot{\mathbf{x}}, t) = \mathbf{0}, \tag{2.60}$$
> $$\frac{d\mathbf{F}}{dt}(\mathbf{x}, \dot{\mathbf{x}}, \ddot{\mathbf{x}}, t) = \mathbf{0}, \tag{2.61}$$
> $$\vdots \tag{2.62}$$
> $$\frac{d^p\mathbf{F}}{dt^p}(\mathbf{x}, \dot{\mathbf{x}}, \ldots, \mathbf{x}^{(p+1)}, t) = \mathbf{0} \tag{2.63}$$
>
> *bestimmt werden kann.*

Hierbei werden die Gleichungen (2.60)-(2.63) auch als *erweitertes System* bezeichnet. In der Praxis ist die Berechnung des differentiellen Index über die symbolische Aufstellung des erweiterten Systems sehr aufwändig und wird daher nur sehr selten durchgeführt[1]. In den folgenden Beispielen wird zunächst der Index einer DAE anhand des erweiterten Systems bestimmt (siehe auch [Hairer *et al.*, 1989]). Danach wird der Index von Modellen von einer ganzen Klasse von mechanischen Systemen betrachtet. Hierzu wird jedoch noch folgende Definition benötigt.

Eine DAE liegt in Hessenberg-Form vom Index r, $r \geq 2$ vor, wenn es die Gestalt

$$\dot{\mathbf{x}}_1 = \mathbf{f}_1(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_r, t), \tag{2.64}$$
$$\dot{\mathbf{x}}_i = \mathbf{f}_i(\mathbf{x}_{i-1}, \mathbf{x}_i, \ldots, \mathbf{x}_{r-1}, t), \quad 2 \leq i \leq r-1, \tag{2.65}$$
$$\mathbf{0} = \mathbf{f}_r(\mathbf{x}_{r-1}) \tag{2.66}$$

besitzt und die Jacobi-Matrizen

$$\frac{\partial \mathbf{f}_r}{\partial \mathbf{x}_{r-1}}, \frac{\partial \mathbf{f}_{r-1}}{\partial \mathbf{x}_{r-2}}, \ldots, \frac{\partial \mathbf{f}_2}{\partial \mathbf{x}_1}, \frac{\partial \mathbf{f}_1}{\partial \mathbf{x}_r} \tag{2.67}$$

regulär sind.

Die Bezeichnung Hessenberg-Form ist der Tatsache geschuldet, dass die transponierte Jacobi-Matrix der rechten Seite eine Matrix in Hessenberg-Form ist. Um das System (2.64)-(2.66) in eine ODE zu transformieren, muss eine Gleichung zur Bestimmung von $\dot{\mathbf{x}}_r$ gefunden werden. Dies gelingt durch r-maliges differenzieren von Gleichung (2.66). Hieraus folgt der differentielle Index $\nu_d = r$. In vielen Anwendungen sind die am häufigsten verwendeten mathematischen Modelle DAEs in Hessenberg-Form.

Beispiel 2.2.2. Es sei die DAE

$$\dot{x}_1 - x_3\dot{x}_2 + x_2\dot{x}_3 = 0, \tag{2.68}$$
$$x_2 = 0, \tag{2.69}$$
$$x_3 = 0 \tag{2.70}$$

mit der Anfangsbedingung

$$x_1(0) = 0 \tag{2.71}$$

gegeben [Hairer *et al.*, 1989]. Wird diese DAE einmal differenziert, so erhält man

$$\ddot{x}_1 - \dot{x}_3\dot{x}_2 - x_3\ddot{x}_2 + \dot{x}_2\dot{x}_3 + x_2\ddot{x}_3 = 0, \tag{2.72}$$
$$\dot{x}_2 = 0, \tag{2.73}$$
$$\dot{x}_3 = 0. \tag{2.74}$$

[1] Obwohl das erweiterte System in der Praxis nur selten berechnet wird, ist es sehr hilfreich für die Herleitung von theoretischen Aussagen.

Die Gleichungen (2.68),(2.73) und (2.74) können dann nach $\dot{x}_1$, $\dot{x}_2$ und $\dot{x}_3$ aufgelöst werden und bilden die ODE

$$\dot{x}_1 = 0, \tag{2.75}$$
$$\dot{x}_2 = 0, \tag{2.76}$$
$$\dot{x}_3 = 0. \tag{2.77}$$

Folglich hat die DAE den differentiellen Index $\nu_d = 1$.

Beispiel 2.2.3. In diesem Beispiel wird der differentielle Index für Modelle von einer Klasse von mechanischen Systemen bestimmt. Wie bereits oben erwähnt, lassen sich die mathematischen Modelle vieler Systeme von praktischer Relevanz als DAE in Hessenberg-Form schreiben. Die Bewegungsgleichungen für ein Mehrkörpersystem mit geometrischen Bindungen lauten [Arnold *et al.*, 1989]:

$$\dot{\mathbf{p}} = \mathbf{v}, \tag{2.78}$$
$$\mathbf{M}(\mathbf{p})\dot{\mathbf{v}} = \mathbf{f}(\mathbf{p}, \mathbf{v}) - \mathbf{G}(\mathbf{p})^T\boldsymbol{\lambda}, \tag{2.79}$$
$$\mathbf{0} = \mathbf{g}(\mathbf{p}). \tag{2.80}$$

Hierbei sind

- $\mathbf{p}$ die verallgemeinerten Koordinaten,
- $\mathbf{v}$ die verallgemeinerten Geschwindigkeiten,
- $\boldsymbol{\lambda}$ die Lagrange-Multipikatoren,
- $\mathbf{M}(\mathbf{p})$ die positiv definite Massenmatrix,
- $\mathbf{f}(\mathbf{p}, \mathbf{v})$ die externen Kräfte,
- $\mathbf{g}(\mathbf{p})$ die Bindungsgleichungen,
- $\mathbf{G} := \frac{\partial}{\partial \mathbf{p}}\mathbf{g}(\mathbf{p})$ die Jacobi-Matrix der Bindungen bezüglich der verallgemeinerten Koordinaten.

Schreibt man diese Gleichungen als

$$\dot{\mathbf{v}} = \mathbf{M}^{-1}(\mathbf{p})\mathbf{f}(\mathbf{p}, \mathbf{v}) - \mathbf{M}^{-1}(\mathbf{p})\mathbf{G}(\mathbf{p})^T\boldsymbol{\lambda}, \tag{2.81}$$
$$\dot{\mathbf{p}} = \mathbf{v}, \tag{2.82}$$
$$\mathbf{0} = \mathbf{g}(\mathbf{p}), \tag{2.83}$$

so wird offensichtlich, dass es sich um eine DAE in Hessenberg-Form mit differentiellem Index $\nu_d = 3$ handelt. An dieser Stelle wird deutlich, dass DAEs mit hohem Index häufig in der Praxis auftreten. Ein weiteres Beispiel dafür entstammt der Strömungsmechanik. Modelliert man den Fluss eines inkompressiblen Fluids mit Hilfe der Navier-Stokes-Gleichungen, so führt dies auf eine DAE in Hessenberg-Form mit Index $\nu_d = 2$.

Blickt man noch einmal zurück auf das Beispiel 2.2.1, so stellt man fest, dass die dort betrachtete DAE offensichtlich in Hessenberg-Form vorliegt. Dabei ist gemäß der Definition der Hessenberg-Form $r = 2$, womit für den differentiellen Index $\nu_d = 2$

gilt. Dies erklärt, wieso die DAE nicht ohne Weiteres mit Hilfe des impliziten Euler-Verfahrens gelöst werden konnte.

Der differentielle Index ν_d kann als Abstand zwischen der betrachteten DAE und einer zugehörigen ODE interpretiert werden. So ist es möglich, eine DAE mit Index v_d mit Hilfe von ν_d Differentiationen in eine ODE zu transformieren. Im folgenden Abschnitt wird der Störungsindex eingeführt, welchem eine grundsätzlich andere Idee zugrunde liegt. Gleichwohl sind der differentielle Index und der Störungsindex für einige DAEs identisch.

Störungsindex

Der Störungsindex ν_p ist ein Maß für den Einfluss von Störungen auf die Lösung einer DAE und wurde erstmalig in [Hairer *et al.*, 1989] verwendet.

Die DAE (2.37) *mit der Lösung* $\mathbf{x}$ *hat den Störungsindex* ν_p*, falls* ν_p *die kleinste natürliche Zahl ist, so dass für die Lösung* $\hat{\mathbf{x}}$ *der gestörten DAE*

$$\mathbf{F}(\hat{\mathbf{x}}, \dot{\hat{\mathbf{x}}}, t) = \boldsymbol{\delta}(t) \tag{2.84}$$

mit einer ausreichend glatten Funktion $\boldsymbol{\delta}(t)$ *eine Abschätzung*

$$\begin{aligned}\|\hat{\mathbf{x}} - \mathbf{x}\| \leq C(\|\hat{\mathbf{x}}(0) - \mathbf{x}(0)\| + \max_{0\leq\xi\leq x} \| \boldsymbol{\delta}(\xi)\| + \ldots \\ + \max_{0\leq\xi\leq x} \left\| \boldsymbol{\delta}^{(\nu_p-1)}(\xi)\right\|)\end{aligned} \tag{2.85}$$

für hinreichend kleines $\|\boldsymbol{\delta}(t)\|$ *gefunden werden kann.*

Ein Zusammenhang zwischen dem Störungsindex und den numerischen Schwierigkeiten ist somit direkt ersichtlich, denn die Störung kann als Rundungsfehler interpretiert werden. Der Störungsindex zeigt dann an, bis zu welchem Ableitungsgrad die Störung Einfluss auf die Lösung hat. In der Folge können auch kleine Rundungsfehler durch entsprechend große Ableitungen einen drastischen Einfluss auf die Lösung haben. Trotz der völlig verschiedenen Definitionen von differentiellem Index und Störungsindex stimmen beide für einige DAEs überein. So wird in [Gear, 1990] gezeigt, dass die beiden Indexbegriffe identisch sind, falls die DAE in der Form

$$\frac{\partial \mathbf{a}}{\partial \mathbf{x}}(\mathbf{x}, t)\dot{\mathbf{x}} + \mathbf{b}(\mathbf{x}, t) = \mathbf{0} \tag{2.86}$$

geschrieben werden kann. Hierbei sind $\mathbf{a}$ und $\mathbf{b}$ zwei hinreichend glatte Funktionen. Man beachte, dass insbesondere der differentielle und der Störungsindex für semi-explizite DAE (und somit auch für DAEs in Hessenberg-Form) übereinstimmen. Darüber hinaus wurde vermutet, dass für beliebige DAEs

$$\nu_d \leq \nu_p \leq \nu_d + 1 \tag{2.87}$$

gilt. In [Campbell & Gear, 1995] wird diese Vermutung jedoch widerlegt und gezeigt, dass ν_d und ν_p beliebig weit auseinander liegen können. Im folgenden Beispiel wird der Störungsindex für die DAE (2.68)-(2.70) bestimmt.

Beispiel 2.2.4. In diesem Beispiel wird die gestörte DAE (siehe Beispiel 2.2.2)

$$\begin{bmatrix} \dot{\hat{x}}_1 - \hat{x}_3\dot{\hat{x}}_2 + x_2\dot{\hat{x}}_3 \\ \hat{x}_2 \\ \hat{x}_3 \end{bmatrix} = \boldsymbol{\delta}(t) \tag{2.88}$$

mit der Anfangsbedingung

$$\hat{x}_1(0) = 0 \tag{2.89}$$

und der Störung

$$\boldsymbol{\delta}(t) = \begin{bmatrix} 0 \\ \epsilon\sin(\omega t) \\ \epsilon\cos(\omega t) \end{bmatrix} \tag{2.90}$$

betrachtet. Die erste Komponente von Gleichung (2.88) liefert

$$\dot{\hat{x}}_1 - \epsilon^2\omega\cos^2(\omega t) - \epsilon^2\omega\sin^2(\omega t) = 0. \tag{2.91}$$

Dies kann vereinfacht werden zu

$$\dot{\hat{x}}_1 = \epsilon^2\omega \tag{2.92}$$

und somit

$$\hat{x}_1 = \epsilon^2\omega t. \tag{2.93}$$

Mit der Lösung $x_1 = 0$ des ungestörten Systems folgt

$$\|x_1 - \hat{x}_1\| = \left\|-\epsilon^2\omega t\right\|. \tag{2.94}$$

Betrachtet man nun $\omega \to \infty$, so kann der Term $\left\| \dot{\boldsymbol{\delta}}(\xi)\right\|$ in Gleichung (2.85) nicht vernachlässigt werden. Ansonsten würde die linke Seite von Gleichung (2.85) beliebig groß werden, während die rechte Seite beschränkt bliebe, da ω dort nur im Argument der trigonometrischen Funktionen auftauchen würde. Folglich hat die DAE den Störungsindex $\nu_p = 2$ [Hairer *et al.*, 1989].

2.2.2 Indexreduktion

Mit Hilfe der vorgestellten Indexbegriffe lassen sich DAEs klassifizieren und eventuelle Probleme bei der numerischen Lösung voraussagen. So wurde in Beispiel 2.2.1 gezeigt, dass das Schätzen des lokalen Fehlers bei DAEs mit differentiellem Index $\nu_d \geq 2$ zu Problemen führen kann, die ein numerisches Integrationsverfahren scheitern lassen. Ferner wurde im letzten Abschnitt deutlich, dass bei DAEs mit einem hohen Störungsindex nicht nur Rundungsfehler, sondern auch deren Ableitungen Einfluss auf die numerische Lösung haben. Auf der anderen Seite wurde in Beispiel

2.2.3 gezeigt, dass DAEs mit hohem Störungsindex keine Seltenheit sind. In vielen Anwendungen führt eine Modellierung, welche die Systemstruktur und nicht die Gleichungsstruktur in den Vordergrund stellt, auf eine DAE mit hohem Störungsindex. Während DAEs zur Beschreibung mechanischer Systeme oft den Index $v_d = 3$ aufweisen, können die Modelle von gekoppelten Systemen verschiedener Domänen auch einen differentiellen Index $\nu_d > 3$ haben [Campbell, 1995].

Da die direkte Lösung von DAEs mit hohem Störungsindex sehr aufwändig ist, ist es naheliegend zu versuchen, sie in eine ODE oder eine DAE mit Störungsindex $\nu_p = 1$ zu transformieren. Dies geschieht in der Regel durch Differentiation geeigneter Gleichungen. Es existieren verschiedene Techniken, um aus der ursprünglichen DAE und den differenzierten Gleichungen die DAE mit reduziertem Index zu generieren. Im Wesentlichen kann man hierbei zwischen drei Methoden unterscheiden. Diese werden in den folgenden Abschnitten diskutiert.

Die zugrundeliegende ODE

Generiert man durch Differentiation aus der DAE eine ODE, so heißt diese *zugrundeliegende* ODE. Die Herleitung der zugrundeliegenden ODE, sowie die in diesem Zusammenhang zu lösenden Probleme werden im Folgenden anhand der DAE aus den Gleichungen (2.81)-(2.83) beschrieben, welche die Dynamik eines Mehrkörpersystems mit geometrischen Bindungen beschreibt.

Zweimaliges Differenzieren von Gleichung (2.83) liefert

$$\mathbf{0} = \mathbf{G}(\mathbf{p})\dot{\mathbf{v}} + \frac{\partial(\mathbf{G}(\mathbf{p})\mathbf{v})}{\partial \mathbf{p}}\mathbf{v}. \tag{2.95}$$

Durch Einsetzen von Gleichung (2.81) und Umformung erhält man

$$\boldsymbol{\lambda} = \mathbf{H}^{-1}(\mathbf{p})(\mathbf{G}(\mathbf{p})\mathbf{M}^{-1}(\mathbf{p})\mathbf{f}(\mathbf{p},\mathbf{v}) + \frac{\partial(\mathbf{G}(\mathbf{p})\mathbf{v})}{\partial \mathbf{p}}\mathbf{v}) \tag{2.96}$$

mit

$$\mathbf{H}(\mathbf{p}) := \mathbf{G}(\mathbf{p})\mathbf{M}^{-1}(\mathbf{p})\mathbf{G}^T(\mathbf{p}). \tag{2.97}$$

Damit lautet die zugrundeliegende Differentialgleichung

$$\dot{\mathbf{v}} = \mathbf{M}^{-1}(\mathbf{p})\widetilde{\mathbf{f}}(\mathbf{p},\mathbf{v}), \tag{2.98}$$

$$\dot{\mathbf{p}} = \mathbf{v}, \tag{2.99}$$

wobei

$$\widetilde{\mathbf{f}}(\mathbf{p},\mathbf{v}) = \mathbf{f}(\mathbf{p},\mathbf{v}) - \mathbf{G}^T(\mathbf{p})\mathbf{H}^{-1}(\mathbf{p})(\mathbf{G}(\mathbf{p})\mathbf{M}^{-1}(\mathbf{p})\mathbf{f}(\mathbf{p},\mathbf{v}) + \frac{\partial(\mathbf{G}(\mathbf{p})\mathbf{v})}{\partial \mathbf{p}}\mathbf{v}) \tag{2.100}$$

ist.

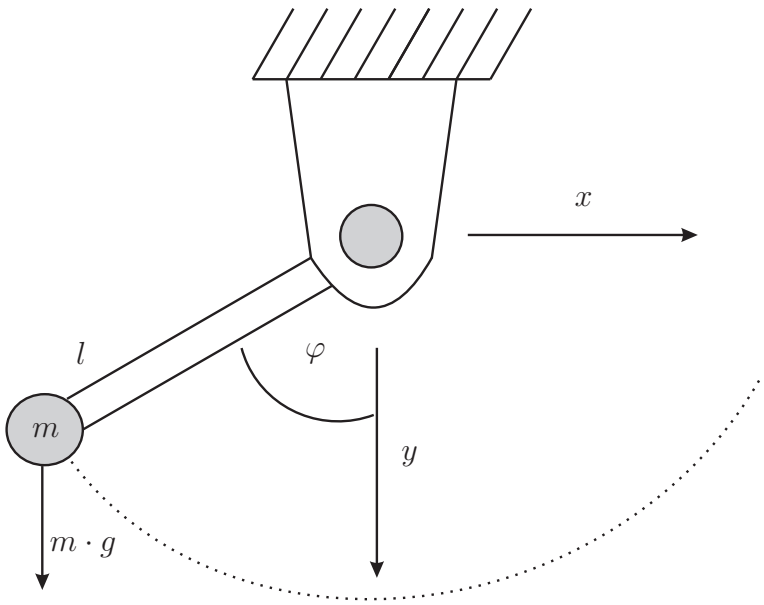


Abbildung 2.9: Mathematisches Pendel

Die DAE mit differentiellem Index $\nu_d = 3$ wurde offensichtlich mit nur zwei Differentiationen auf eine ODE transformiert. Der Grund hierfür ist, dass in der zugrundeliegenden ODE offensichtlich keine Gleichung zur Bestimmung der Lagrange-Multiplikatoren $\boldsymbol{\lambda}$ vorhanden ist. Für eine Gleichung zur Bestimmung von $\dot{\boldsymbol{\lambda}}$ wäre eine weitere Differentiation notwendig. Auf den ersten Blick scheint man sich den mit dem hohen differentiellem Index verbundenen Problemen auf einfache Art und Weise, durch mehrfache Differentiation entledigt zu haben. Auf den zweiten Blick offenbaren sich jedoch die folgenden Herausforderungen [Campbell & Leimkuhler, 1991; Führer & Leimkuhler, 1991].

Drift-off-Effekt Die numerischen Lösungen von DAEs und ODEs sind natürlich nicht exakt, sondern beinhalten einen gewissen numerischen Fehler. In Abschnitt 2.2.3 wird hierauf näher eingegangen. Die Bindungen aus Gleichung (2.83) sind Bindungen auf Lageebene. Folglich werden diese Bindungen während der numerischen Lösung bis auf den oben erwähnten Fehler nicht verletzt. In der zugrundeliegenden ODE existieren diese Bindungen nicht mehr, sondern wurden durch die Gleichung (2.95) ersetzt. Diese neuen Bindungen sind allerdings Bindungen auf Beschleunigungsebene. Wird die zugrundeliegende ODE numerisch gelöst, werden folglich nur die Bindungen auf Beschleunigungsebene bis auf den numerischen Fehler eingehalten. Auf der Lageebene akkumuliert sich dieser Fehler und wächst an. Dieses Verhalten wird *Drift-off*-Effekt genannt und lässt sich am Beispiel eines ebenen Pendels sehr leicht veranschaulichen. Das mathematische Modell des in Abb. 2.9 dargestellten ebenen Pendels in kartesischen Koordinaten ist eine DAE in Hessenberg-Form mit differentiellem Index $\nu_d = 3$ und beinhaltet die Bindung

$$x^2 + y^2 - l^2 = 0. \tag{2.101}$$

Hierbei sind x und y die kartesischen Koordinaten und l die Länge des Pendels. Die zugrundeliegende ODE erfüllt diese Bindung lediglich in zweimal differenzierter

Form:

$$\dot{x}^2 + x\ddot{x} + \dot{y}^2 + y\ddot{y} = 0. \tag{2.102}$$

Aufgrund des Drift-off-Effektes ist dann in der numerischen Lösung die Länge des Pendels nicht mehr l, sondern ändert sich aufgrund der Akkumulierung des Fehlers auf Lageebene.

Versteckte Nebenbedingungen Die zugrundeliegende ODE besitzt nicht nur die Lösungen der zugehörigen DAE, sondern weitaus mehr. Der Grund hierfür liegt in den Anfangswerten. Wie schon zu Beginn dieses Abschnitts erwähnt, müssen zur Lösung einer DAE konsistente Anfangswerte bekannt sein oder bestimmt werden. Für die numerische Lösung der zugrundeliegenden ODE hingegen sind lediglich Anfangswerte für die verallgemeinerten Koordinaten $\mathbf{p}$ und Geschwindigkeiten $\mathbf{v}$ notwendig. Damit die Lösung jedoch mit der Lösung der zugehörigen DAE übereinstimmt, müssen die Anfangswerte nicht nur den ursprünglichen Bindungen

$$\mathbf{g}(\mathbf{p}) = \mathbf{0}, \tag{2.103}$$

sondern auch den Ableitungen der Bindungsgleichungen, welche bei der Indexreduktion entstehen,

$$\mathbf{G}(\mathbf{p})\mathbf{v} = \mathbf{0} \tag{2.104}$$

genügen. Die Gleichungen (2.103) und (2.104) heißen *versteckte Nebenbedingungen.*

Die versteckten Nebenbedingungen bedeuten keinen wirklichen Nachteil der zugrundeliegenden ODE gegenüber der zugehörigen DAE. Denn auch bei der Bestimmung von konsistenten Anfangswerten für die DAE müssen die versteckten Nebenbedingungen beachtet werden. Der Drift-off-Effekt hingegen stellt ein großes Problem dar. Besonders im Fall von DAEs mit hohem differentiellem Index kann der entstehende Drift beachtlich sein [Campbell & Leimkuhler, 1991]. Daher wurden Stabilisierungsverfahren entwickelt, die dafür Sorge tragen, dass die Lösung der zugrundeliegenden ODE die Bindungen auf Lageebene in hinreichender Genauigkeit erfüllt [Alishenas, 1992; Baumgarte, 1972; Gear *et al.*, 1985]. In dieser Arbeit wird auf diese Techniken jedoch nicht näher eingegangen.

Überbestimmte DAE

Anstatt die zugrunde liegende ODE zu verwenden, kann auch eine überbestimmte DAE verwendet werden. Dies ist eine Möglichkeit dem Drift-off-Effekt zu begegnen. Die Idee besteht darin, diejenigen Gleichungen, welche differenziert werden, nicht aus der DAE zu streichen. Statt dessen wird die DAE um die sukzessive entstehenden differenzierten Gleichungen erweitert. Das Ergebnis dieses Vorgehens ist eine überbestimmte DAE mit Störungsindex $\nu_p = 1$. Betrachtet man wieder die Anwendung

dieser Methode auf die mechanischen Bewegungsgleichungen, so ergibt sich:

$$\dot{\mathbf{v}} = \mathbf{M}^{-1}(\mathbf{p})\mathbf{f}(\mathbf{p},\mathbf{v}) - \mathbf{M}^{-1}(\mathbf{p})\mathbf{G}^T(\mathbf{p})\boldsymbol{\lambda}, \tag{2.105}$$

$$\dot{\mathbf{p}} = \mathbf{v}, \tag{2.106}$$

$$\mathbf{0} = \mathbf{g}(\mathbf{p}), \tag{2.107}$$

$$\mathbf{0} = \mathbf{G}(\mathbf{p})\mathbf{v}, \tag{2.108}$$

$$\mathbf{0} = \mathbf{G}(\mathbf{p})\mathbf{M}^{-1}(\mathbf{p})\mathbf{f}(\mathbf{p},\mathbf{v}) - \mathbf{H}(\mathbf{p})\boldsymbol{\lambda} + \frac{\partial(\mathbf{G}\mathbf{v})}{\partial\mathbf{p}}\mathbf{v}. \tag{2.109}$$

Der Vorteil dieser Methode ist, dass die analytische Lösung der überbestimmten DAE mit der Lösung der zugehörigen DAE übereinstimmt. Dieses Vorgehen birgt jedoch auch zwei Nachteile. Zum einen sind zur Lösung von überbestimmten DAEs spezielle Integrationsverfahren notwendig. Zum anderen zerstört die zur Lösung notwendige Diskretisierung im Allgemeinen die numerische Stabilität, was wiederum spezielle Stabilisierungsverfahren notwendig macht [Eich *et al.*, 1990].

Dummy Derivatives

Im Jahr 1988 wurde in [Pantelides, 1988] ein Algorithmus zur Berechnung von konsistenten Anfangswerten vorgestellt. Dieser Algorithmus eignet sich jedoch nicht nur zur Initialisierung von DAEs. Mattsson und Söderlind entwickelten basierend auf *Pantelide's Algorihmus* ein Indexreduktionsverfahren, welches mit so genannten *dummy derivatives* arbeitet. Das grundsätzliche Vorgehen ist hierbei zunächst dasselbe wie im vorherigen Abschnitt. Nach jeder Differentiation werden nun allerdings die dummy derivatives als zusätzliche Variablen eingeführt, so dass die reduzierte DAE nicht mehr überbestimmt ist [Mattsson & Söderlind, 1993]. Dieses Verfahren wird zunächst an dem Beispiel eines ebenen Pendels wie auch in [Cellier & Kofman, 2006; Mattsson & Söderlind, 1993] vorgeführt.

Beispiel 2.2.5. Das mathematische Modell des in Abb. 2.9 gezeigten Pendels mit Masse m und Länge l lautet in kartesischen Koordinaten (vergleiche Gleichung (2.81)-(2.83)):

$$\dot{x} = v_x, \tag{2.110}$$

$$\dot{y} = v_y, \tag{2.111}$$

$$\dot{v}_x = -\frac{x \cdot \lambda}{m \cdot l}, \tag{2.112}$$

$$\dot{v}_y = g - \frac{y \cdot \lambda}{m \cdot l}, \tag{2.113}$$

$$0 = x^2 + y^2 - l^2. \tag{2.114}$$

In dieser Formulierung hat die DAE den differentiellen Index und Störungsindex $\nu_d = \nu_p = 3$. Wird die Bindungsgleichung (2.114) einmal differenziert und der DAE

hinzugefügt, so ergibt sich die überbestimmte DAE

$$\dot{x} = v_x, \tag{2.115}$$
$$\dot{y} = v_y, \tag{2.116}$$
$$\dot{v}_x = -\frac{x \cdot \lambda}{m \cdot l}, \tag{2.117}$$
$$\dot{v}_y = g - \frac{y \cdot \lambda}{m \cdot l}, \tag{2.118}$$
$$0 = x^2 + y^2 - l^2, \tag{2.119}$$
$$0 = x \cdot \dot{x} + y \cdot \dot{y} \tag{2.120}$$

mit 5 Unbekannten und 6 Gleichungen. Damit die DAE wieder bestimmt ist, wird nun eine neue Unbekannte (dummy derivative) eingeführt. Ersetzen von $\dot{x}$ durch die neue Unbekannte $d_{1,x}$ liefert das gewünschte Ergebnis. Dabei ist zu beachten, dass $d_{1,x}$ eine neue algebraische Variable und keine differentielle Variable mehr ist. Die neue DAE

$$d_{1,x} = v_x, \tag{2.121}$$
$$\dot{y} = v_y, \tag{2.122}$$
$$\dot{v}_x = -\frac{x \cdot \lambda}{m \cdot l}, \tag{2.123}$$
$$\dot{v}_y = g - \frac{y \cdot \lambda}{m \cdot l}, \tag{2.124}$$
$$0 = x^2 + y^2 - l^2, \tag{2.125}$$
$$0 = x \cdot d_{1,x} + y \cdot \dot{y} \tag{2.126}$$

ist dann bestimmt und hat den differentiellen Index $\nu_d = 2$. Dies lässt sich wie folgt einsehen. Gleichung (2.125) kann nach x (nun eine algebraische Variable) und Gleichung (2.126) kann nach $d1, x$ aufgelöst werden. Es verbleibt also Gleichung (2.121) als einzige algebraische Gleichung und die DAE liegt in Hessenberg-Form vom Index 2 vor. Um den differentiellen Index ν_d weiter zu senken, muss also die neue Bindungsgleichung einmal differenziert werden. Die DAE wird also um die Gleichung

$$\dot{d}_{1,x} = \dot{v}_x \tag{2.127}$$

erweitert und die dummy derivative $d_{2,x}$ wird für $\dot{dx}$ eingeführt. Die Gleichung (2.126) wurde vorher nur noch benötigt, um $d_{1,x}$ zu bestimmen und muss daher nun auch differenziert werden. Das liefert die zusätzliche Gleichung

$$0 = d_{1,x}^2 + x \cdot d_{2,x} + \dot{y}^2 + y \cdot \ddot{y}. \tag{2.128}$$

Durch die zusätzliche Gleichung ist die DAE wieder überbestimmt. Die Einführung der neuen dummy derivative $d_{2,y}$ für $\ddot{y}$ löst dieses Problem zunächst. Ähnlich wie nach der vorherigen Differentiation muss nun jedoch Gleichung (2.122) differenziert werden, da diese Gleichung $\dot{y}$ bestimmte. Die zusätzliche Gleichung lautet

$$d_{2,y} = \dot{v}_y. \tag{2.129}$$

Wählt man nun d_{1,v_x} als dummy derivative, so erhält man eine DAE mit neun Gleichungen

$$d_{1,x} = v_x, \tag{2.130}$$
$$\dot{y} = v_y, \tag{2.131}$$
$$\dot{v}_x = -\frac{x \cdot \lambda}{m \cdot l}, \tag{2.132}$$
$$\dot{v}_y = g - \frac{y \cdot \lambda}{m \cdot l}, \tag{2.133}$$
$$d_{2,x} = d_{1,v_x}, \tag{2.134}$$
$$d_{2,y} = \dot{v}_y, \tag{2.135}$$
$$0 = x^2 + y^2 - l^2, \tag{2.136}$$
$$0 = x \cdot dx + y \cdot \dot{y}, \tag{2.137}$$
$$0 = d_{1,x}^2 + y \cdot \dot{y} \tag{2.138}$$

und den neun Unbekannten $x, v_x, d_{1,x}, d_{2,x}, d_{1,v_x}, y, v_y, d_{2,y}$ und λ. Die DAE hat wieder Hessenberg-Form, so dass sich der Index $\nu_d = \nu_p = 1$ leicht ablesen lässt.

In dem obigen Beispiel wirkt die Auswahl der Gleichungen, welche differenziert werden sowie die Zuweisung der dummy derivatives willkürlich. Im allgemeinen Fall erfolgt die Auswahl der Gleichungen, die differenziert werden, und die Zuweisung der dummy derivatives durch *Pantelide's Algorithmus*.

In [Mattsson & Söderlind, 1993] wird gezeigt, dass die Indexreduktion mit dummy derivatives die Reduktion auf eine DAE mit Störungsindex $\nu_p \leq 1$ garantiert. Darüber hinaus tritt der Drift-off-Effekt nicht auf. Daher ist dieses Verfahren sehr verbreitet und wird auch in kommerziellen Simulationsumgebungen wie zum Beispiel Dymola verwendet [Elmqvist *et al.*, 1995a].

Das Verfahren bringt jedoch auch zwei Nachteile mit sich. Zum einen kann die DAE mit reduziertem Index mehr Lösungen als die ursprüngliche DAE besitzten [Cellier & Kofman, 2006]. In der Folge muss dann automatisiert zwischen der *richtigen* und der *falschen* Lösung unterschieden werden. In [Mattsson *et al.*, 2000] wird gezeigt, wie automatisch zwischen verschiedenen Modellvarianten umgeschaltet werden kann, um stest die *richtige* Lösung zu verfolgen. Zum anderen kann der Aufwand des Verfahrens (zumindest theoretisch) beliebig groß sein[2] [Reissig *et al.*, 2000].

In [Kunkel & Mehrmann, 2004] wird eine Weiterentwicklung der Indexreduktion mit dummy derivatives präsentiert. Ist die Struktur der DAE bekannt, so kann in vielen Fällen der numerische Aufwand der Indexreduktion gesenkt werden. Die von Kunkel beschriebenen Techniken funktionieren beispielsweise für DAEs in Hessenberg-Form und sind somit in vielen Anwendungen einsetzbar.

[2] Reißig zeigt, dass man DAEs konstruieren kann, die den Störungsindex $\nu_p = 1$ besitzen und gleichwohl beliebig viele Iterationen in Pantelide's Algorithmus benötigen.

2.2.3 Numerische Integrationsverfahren für DAEs

Im vorherigen Abschnitt wurden Methoden vorgestellt, um den Index von DAEs zu senken. Dabei wurde festgestellt, dass bei der Reduktion von DAEs auf ODEs der Drift-off-Effekt auftritt oder alternativ komplizierte Projektionsmethoden verwendet werden müssen. Mit Hilfe von dummy derivatives kann der Drift-off-Effekt vermieden werden, jedoch ist das Resultat im Allgemeinen eine DAE mit Störungsindex $\nu_p = 1$. Im Folgenden werden zwei verschiedene Methoden zur Lösung von DAEs mit Störungsindex $\nu_p = 1$ vorgestellt. Wie schon zu Beginn dieses Abschnitts erwähnt scheinen implizite Runge-Kutta-Verfahren zur Lösung von ODEs auch zur Lösung von DAEs geeignet. Hier haben sich insbesondere die Radau-IIa-Verfahren etabliert.

Auf der anderen Seite wurde bereits in [Gear, 1971] die Verwendung von BDF-Verfahren (*backward differential formulas*) zur Lösung von DAEs vorgeschlagen. Hierauf basiert auch das wohl verbreitetste und bekannteste Verfahren DASSL (Differential Algebraic System Solver) [Petzold, 1983].

Zunächst wird jedoch noch ein kurzer Überblick über einige Begrifflichkeiten gegeben, die schon aus der Theorie der ODEs bekannt sind, um die Verfahren besser miteinander vergleichen zu können. Ein wichtiges Merkmal eines numerischen Integrationsverfahrens ist selbstverständlich dessen Genauigkeit. Die Genauigkeit eines Verfahrens wird gewöhnlich anhand des *lokalen Diskretisierungsfehlers* gemessen. Der lokale Diskretisierungsfehler ist der Fehler, welcher durch das numerische Integrationsverfahren in einem Zeitschritt unter der Annahme, dass die exakte Lösung zu Beginn dieses Schrittes bekannt ist, verursacht wird.

> *Ist* $\mathbf{x}(t_{n+1})$ *die exakte Lösung und* $\hat{\mathbf{x}}_{n+1}$ *die durch ein numerisches Integrationsverfahren berechnete Lösung einer DAE an einer Stelle* t_{n+1}*, dann heißt*
>
> $$\boldsymbol{\rho}_{n+1} = \mathbf{x}(t_{n+1}) - \hat{\mathbf{x}}_{n+1} \tag{2.139}$$
>
> *lokaler Diskretisierungsfehler.*

Darüber hinaus werden verschiedene numerische Integrationsverfahren anhand ihrer Konsistenzordnung verglichen. Die Konsistenzordnung gibt dabei an, ab welchem Glied sich die Taylorreihe der exakten Lösung von der Taylorreihe der numerischen Lösung unterscheidet.

> *Ein numerisches Integrationsverfahren ist von der Konsistenzordnung* p*, falls für den lokalen Diskretisierungsfehler*
>
> $$\boldsymbol{\rho}_n = \mathcal{O}(h^{p+1}) \tag{2.140}$$
>
> *gilt.*

Neben der Genauigkeit eines Verfahrens ist auch die Stabilität ein wichtiges Merkmal von numerischen Integrationsverfahren. Wie im Falle des Index für DAEs existieren auch hier verschiedene Stabilitätsbegriffe [Hairer & Wanner, 1991]. In dieser Arbeit wird die A-Stabilität sowie die L-Stabilität verwendet.

Auf den ersten Blick scheint es natürlich zu fordern, dass für alle Probleme die numerisch berechnete Lösung die wesentlichen Eigenschaften der exakten Lösung widerspiegelt. Da dies leider nicht möglich ist, beschränkt man sich an dieser Stelle. Das Konzept der A-Stabilität geht zurück auf Dahlquist und fordert, dass die numerisch berechnete Lösung einer linearen Testgleichung für beliebige Anfangswerte und Schrittweiten die wesentlichen Eigenschaften der exakten Lösung widerspiegelt [Dahlquist, 1963].

Ein numerisches Integrationsverfahren heißt A-stabil, falls für die numerisch berechnete Lösung $(x)_n$ der Testgleichung

$$\dot{x} = \lambda x, \quad x(t_0) = x_0 \tag{2.141}$$

für alle $\lambda \in \mathbb{C}$ mit $\mathrm{Re}(\lambda) \leq 0$ gilt

$$|x_n| \geq |x_{n+1}|\,. \tag{2.142}$$

Die wesentlichen Eigenschaften der exakten Lösung $e^{\lambda h}x_0$ bestehen hier also darin, dass sie monoton fällt und gegen Null konvergiert. Beispiele für A-stabile Verfahren sind das implizite Euler-Verfahren, die Trapezregel oder die Radau-IIa-Verfahren.

Diese Forderung ist aber für viele numerischen Integrationsverfahren, welche in der Praxis gut funktionieren, zu hoch. Darüber hinaus existiert nur die Unterscheidung zwischen A-stabil und nicht A-stabil. Widlund führte 1967 die etwas schwächere A(α)-Stabilität ein. Nennt man denjenigen Bereich der komplexen Zahlenebene, in welchem die numerische Lösung der Testgleichung betragsmäßig monoton fällt, so fordert die A(α)-Stabilität lediglich, dass der Stabilitätsbereich bestimmte Sektoren der linken komplexen Halbebene beinhaltet [Widlund, 1967].

Ein numerisches Integrationsverfahren heißt A(α)-stabil, falls für die numerisch berechnete Lösung $(x)_n$ der Testgleichung

$$\dot{x} = \lambda x, \quad x(t_0) = x_0 \tag{2.143}$$

für alle $\lambda \in \mathbb{C}$ mit $|arg(-\lambda)| \leq \alpha$ gilt

$$|x_n| \geq |x_{n+1}|\,. \tag{2.144}$$

In Abbildung 2.10 ist die A(α)-Stabilität veranschaulicht. Offensichtlich gilt, dass A($\frac{\pi}{2}$)-Stabilität identisch mit A-Stabilität ist. Folglich ist die A(α)-Stabilität ein Maß, um die Stabilität verschiedener numerischer Lösungsverfahren miteinander zu vergleichen. Leider ist jedoch auch die A(α)-Stabilität trügerisch. Auch ihre Aussagekraft bezüglich der Eignung eines Verfahrens zur Lösung einer ODE ist begrenzt. Die Anwendung der A-stabilen Trapezregel auf die lineare Testgleichung liefert beispielsweise

$$x_{n+1} = x_n + \frac{\lambda h}{2}(x_{n+1} + x_n) \tag{2.145}$$

$$\Leftrightarrow \frac{x_{n+1}}{x_n} = \frac{2 + \lambda h}{2 - \lambda h}. \tag{2.146}$$

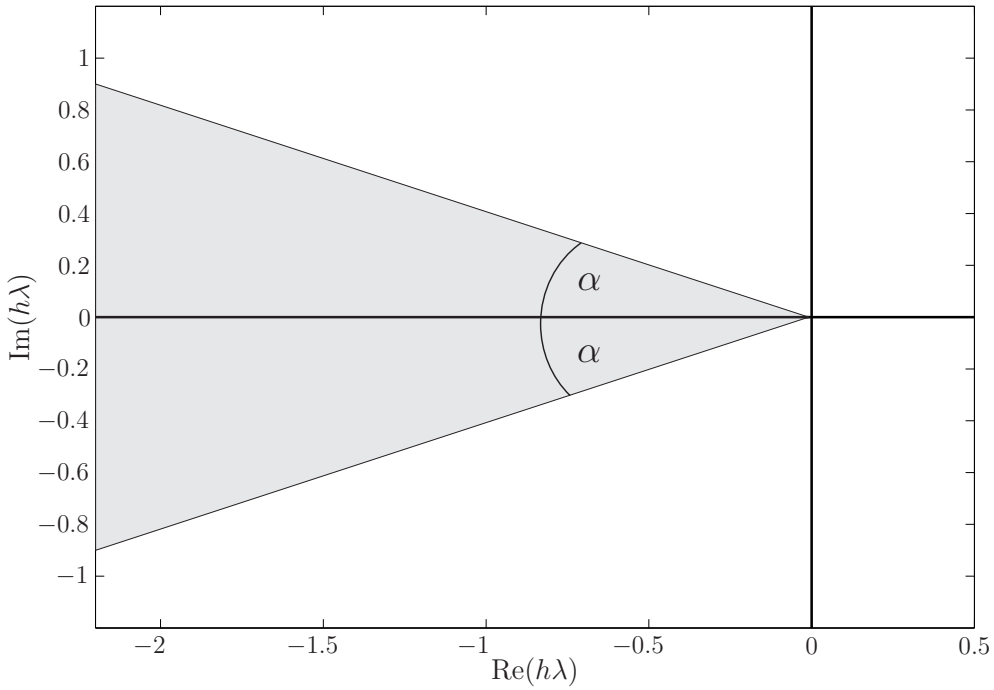


Abbildung 2.10: A(α)-Stabilität

Es gilt aber

$$\frac{2+\lambda h}{2-\lambda h} \to -1, \text{ für } h\mathrm{Re}\,\lambda \to -\infty. \tag{2.147}$$

Folglich ist für λh mit $\mathrm{Re}\,\lambda << -1$

$$\left|\frac{x_{n+1}}{x_n}\right| \approx 1. \tag{2.148}$$

Der Realteil eines Eigenwerts beschreibt seine Dämpfung. Eigenwerte mit großem negativen Realteil besitzen eine sehr hohe Dämpfung. Gleichung (2.148) besagt allerdings, dass gerade die Lösungskomponenten, die zu diesen Eigenwerten gehören, von der Trapezregel nur sehr wenig gedämpft werden. Diese Komponenten bleiben also sehr lange in der Lösung erhalten, obwohl sie in der Realität sehr schnell abklingen. Somit ist die Trapezregel nicht geeignet ODEs zu lösen, die Eigenwerte mit großen negativen Realteilen aufweisen. Dies ist insbesondere bei steifen ODEs beziehungsweise DAEs der Fall.

Für die Simulation von steifen ODEs oder DAEs sind daher andere Verfahren notwendig. Im besten Fall erhöht sich die Dämpfung bei der Vergrößerung der Schrittweite. Ein solches Verfahren ist zum Beispiel das implizite Euler-Verfahren. Die Anwendung auf die Testgleichung liefert

$$\frac{x_{n+1}}{x_n} = \frac{1}{1-\lambda h}. \tag{2.149}$$

Für das implizite Euler-Verfahren gilt also offensichtlich

$$\left|\frac{x_{n+1}}{x_n}\right| \to 0, \text{ für } h\,\mathrm{Re}(\lambda) \to -\infty. \tag{2.150}$$

Somit werden Lösungskomponenten, die zu Eigenwerten mit großem negativen Realteil gehören, stark numerisch gedämpft. Der Unterschied zwischen der Trapezregel und dem impliziten Euler-Verfahren kann durch die *L-Stabilität* charakterisiert werden [Ehle, 1969].

Ein numerisches Integrationsverfahren heißt L-stabil, falls für die numerisch berechnete Lösung $(x)_n$ der Testgleichung

$$\dot{x} = \lambda x, \quad x(t_0) = x_0 \tag{2.151}$$

gilt

$$|x_n| \to 0, \textit{ für } h\,\mathrm{Re}(\lambda) \to -\infty. \tag{2.152}$$

Man beachte, dass alle drei Stabilitätsbegriffe auf einer linearen Testgleichung beruhen und man somit auch streng genommen nur Rückschlüsse auf das Verhalten bei der numerischen Lösung von linearen ODEs ziehen kann. Dennoch sind diese drei Stabilitätsbegriffe gute Indikatoren für das Verhalten bei nichtlinearen ODEs (oder DAEs). Im Folgenden werden nun die beiden wohl verbreitetsten numerischen Integrationsverfahren für DAEs (und auch ODEs) vorgestellt und verglichen.

Radau-IIa-Verfahren

Die Radau-IIa-Verfahren gehören zur Klasse der impliziten Runge-Kutta-Verfahren. Die allgemeine Verfahrensvorschrift für ein s-stufiges implizites Runge-Kutta-Verfahren mit Koeffizienten a_{ij}, c_j und b_j lautet [Hairer *et al.*, 1987]

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\sum_{j=1}^{s} b_j \mathbf{k}_j \tag{2.153}$$

mit

$$\mathbf{k}_i = \mathbf{f}(\mathbf{x}_n + \sum_{j=1}^{s} a_{ij}\mathbf{k}_j, t_n + c_j h), \quad 1 \le i \le s. \tag{2.154}$$

Die Radau-Verfahren zeichnen sich durch eine spezielle Wahl der Stellen $c_j h$ aus, an denen (im Falle von ODEs) die rechte Seite ausgewertet wird. Beispielsweise kann auch das implizite Euler-Verfahren formal zu den Radau-IIa-Verfahren gezählt werden. Von praktischer Relevanz sind jedoch lediglich das zweistufige Radau-IIa-Verfahren von der Konsistenzordnung drei sowie das dreistufige Verfahren, welches von der Konsistenzordnung fünf ist. Wie für alle Runge-Kutta-Verfahren können die Koeffizienten für die Verfahren in einem Butcherschema dargestellt werden. In Tabelle 2.2 und 2.3 sind die entsprechenden Tableaus abgebildet. Um die Notation

möglichst einfach zu halten, wird im Folgenden nur auf das Radau-IIa-Verfahren dritter Ordnung eingegangen, obwohl das Verfahren fünfter Ordnung verbreiteter ist. Für die Berechnung der Lösung $\mathbf{x}_{n+1} \in \mathbb{R}^N$ an der Stelle $t_{n+1} \in \mathbb{R}$ einer ODE in Zustandsform

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t) \tag{2.155}$$

folgt aus dem Butcher-Tableau die Berechnungsvorschrift

$$\mathbf{k}_1 = \mathbf{f}(\mathbf{x}_n + \frac{5h}{12}\mathbf{k}_1 - \frac{h}{12}\mathbf{k}_2, t_n + \frac{h}{3}), \tag{2.156}$$

$$\mathbf{k}_2 = \mathbf{f}(\mathbf{x}_n + \frac{3h}{4}\mathbf{k}_1 + \frac{h}{4}\mathbf{k}_2, t_n + h), \tag{2.157}$$

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \frac{h}{4}(3\mathbf{k}_1 + \mathbf{k}_2). \tag{2.158}$$

Der Übergang auf die Lösung von DAEs der Form ((2.37)) kann also sehr leicht durch die neue Berechnungsvorschrift

$$\mathbf{F}(\mathbf{x}_n + \frac{5h}{12}\mathbf{k}_1 - \frac{h}{12}\mathbf{k}_2, \mathbf{k}_1, t_n + \frac{h}{3}) = \mathbf{0}, \tag{2.159}$$

$$\mathbf{F}(\mathbf{x}_n + \frac{3h}{4}\mathbf{k}_1 + \frac{h}{4}\mathbf{k}_2, \mathbf{k}_2, t_n + h) = \mathbf{0}, \tag{2.160}$$

$$\tag{2.161}$$

geschehen. In jedem Zeitschritt ist also die Lösung eines nichtlinearen Gleichungssystems der Dimension $2N$ notwendig. In der Regel wird hierzu das Newton-Verfahren verwendet. Man beachte, dass nun die Vorgabe eines Startwerts $\mathbf{x}(t_0)$ nicht mehr ausreicht, sondern zusätzlich ein Startwert für $\mathbf{k}_1 = \mathbf{k}_2 = \dot{\mathbf{x}}(t_0)$ bereit gestellt werden muss.

Anwendung des Verfahrens auf die lineare Testgleichung liefert

$$k_1 = (1 - \frac{2\lambda h}{3} + \frac{(\lambda h)^2}{6})^{-1} \cdot (1 - \frac{\lambda h}{3}) \cdot \lambda \cdot x_n, \tag{2.162}$$

$$k_2 = (1 - \frac{2\lambda h}{3} + \frac{(\lambda h)^2}{6})^{-1} \cdot (1 + \frac{\lambda h}{3}) \cdot \lambda \cdot x_n, \tag{2.163}$$

$$x_{n+1} = x_n + \frac{h}{4}(3k_1 + k_2). \tag{2.164}$$

Es ergibt sich

$$x_{n+1} = \left[1 + (1 - \frac{2\lambda h}{3} + \frac{(\lambda h)^2}{6})^{-1} \cdot (1 - \frac{\lambda h}{6}) \cdot \lambda h\right] \cdot x_n. \tag{2.165}$$

1/3	5/12	-1/12
1	3/4	1/4
	3/4	1/4

Tabelle 2.2: Butcher-Tableau des zweistufigen Radau-IIa-Verfahrens

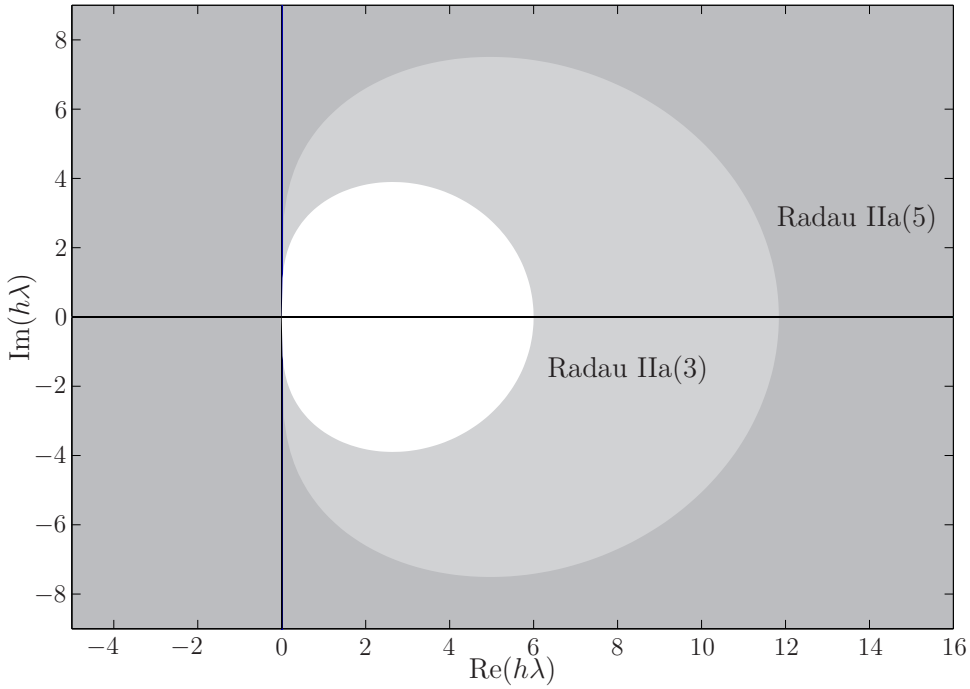


Abbildung 2.11: Stabilitätsbereiche (dunkle Bereiche) der Radau-IIa-Verfahren

Der Vergleich der Taylorreihenentwicklung des Koeffizienten vor x_n um $h = 0$

$$1 + \lambda h + \frac{(\lambda h)^2}{2} + \frac{(\lambda h)^3}{6} + \frac{(\lambda h)^4}{36} + \dots \tag{2.166}$$

mit der Entwicklung des Koeffizienten in der exakten Lösung

$$e^{\lambda h} = 1 + \lambda h + \frac{(\lambda h)^2}{2} + \frac{(\lambda h)^3}{6} + \frac{(\lambda h)^4}{24} + \dots \tag{2.167}$$

belegt die oben behauptete Konsistenzordung (zumindest für lineare ODEs). Die Stabilitätsbereiche der beiden Radau-IIa-Verfahren sind in Abbildung 2.11 dargestellt. Wie zu sehen ist, sind beide Verfahren A-stabil. Darüber hinaus ist die L-Stabilität direkt aus Gleichung (2.165) ersichtlich. Weiterhin kann man zeigen, dass die Radau-IIa-Verfahren *B-stabil* sind [Frank *et al.*, 1985]. Für B-Stabilität wird nicht mehr auf die lineare Testgleichung zurückgegriffen, sondern das Verhalten des Verfahrens für nichtlineare Probleme bewertet. Dabei ist B-Stabilität stärker als A-Stabilität und impliziert diese somit [Hairer & Wanner, 1991].

$\frac{4-\sqrt{6}}{10}$	$\frac{88-7\sqrt{6}}{360}$	$\frac{296-169\sqrt{6}}{1800}$	$\frac{-2+3\sqrt{6}}{225}$
$\frac{4+\sqrt{6}}{10}$	$\frac{296+169\sqrt{6}}{1800}$	$\frac{88+7\sqrt{6}}{360}$	$\frac{-2-3\sqrt{6}}{225}$
1	$\frac{16-\sqrt{6}}{36}$	$\frac{16+\sqrt{6}}{36}$	$\frac{1}{9}$
	$\frac{16-\sqrt{6}}{36}$	$\frac{16+\sqrt{6}}{36}$	$\frac{1}{9}$

Tabelle 2.3: Butcher-Tableau des dreistufigen Radau-IIa-Verfahrens

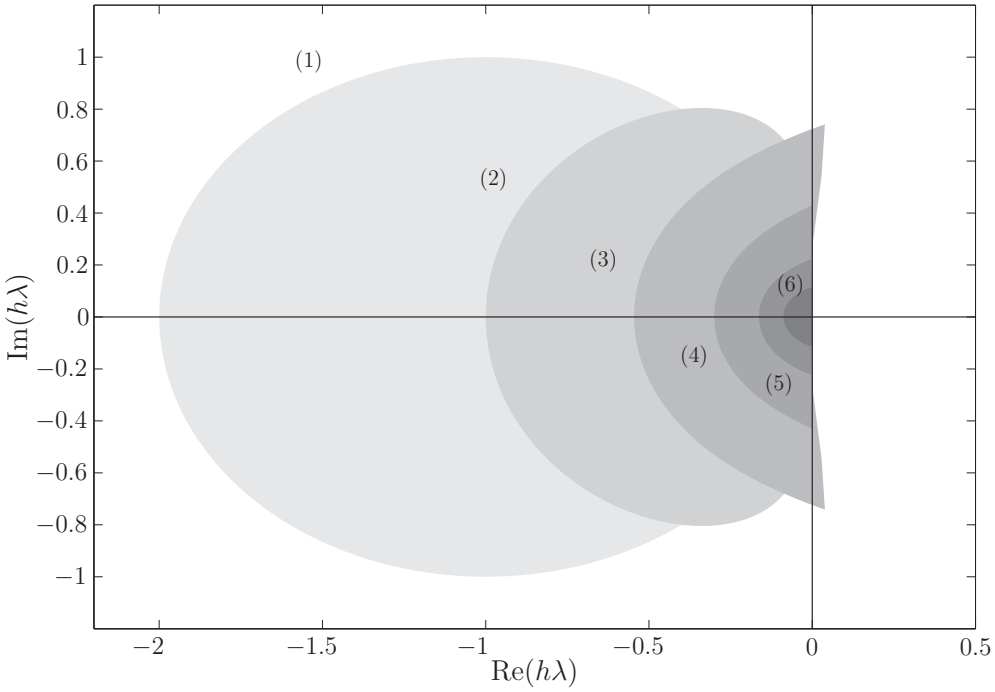


Abbildung 2.12: Stabilitätsbereiche (dunkle Bereiche) der Adams-Bashforth-Verfahren

Eine äußerst effiziente FORTRAN- sowie C/C++-Implementierung des Radau-IIa-Verfahrens fünfter Ordnung mit Schrittweitensteuerung mittels eines eingebetteten Verfahrens ist online erhältlich [Hairer & Wanner, 1996].

DASSL

Der wohl populärste Code zur numerischen Integration ist DASSL. Geschrieben wurde DASSL 1982 von Linda Petzold und wird heutzutage (nach einigen Weiterentwicklungen) von vielen kommerziellen Simulationsumgebungen verwendet. DASSL basiert auf den BDF-Verfahren, welche zur Klasse der linearen Mehrschrittverfahren gehören. Die ersten linearen Mehrschrittverfahren wurden 1882 von J.C Adams und F. Bashforth eingeführt. Mit Hilfe der so genannten Adams-Bashforth-Verfahren untersuchten sie den Kapillareffekt [Bashforth & Adams, 1882].

Die Adams-Bashforth-Verfahren basieren auf der formalen Integration von Gleichung (2.155). Dies liefert

$$\mathbf{x}(t_{n+1}) = \mathbf{x}(t_n) + \int_{t_n}^{t_{n+1}} \mathbf{f}(x(t), t)dt. \tag{2.168}$$

Die Idee der Adams-Bashforth-Verfahren ist es, den Integranden in Gleichung (2.168) mit einem Polynom p der Ordnung $k-1$ zu interpolieren. Die Integration von p

liefert dann den gesuchten Wert x_{n+1}. Für die Interpolation werden die in den k vorherigen Schritten berechneten Werten $\mathbf{f}(\mathbf{x}_n, t_n)$, $\mathbf{f}(\mathbf{x}_{n-1}, t_{n-2})$, ..., $\mathbf{f}(\mathbf{x}_{n-k+2}, t_{n-k+2})$ und $\mathbf{f}(\mathbf{x}_{n-k+1}, t_{n-k+1})$ verwendet. Die auf diese Art und Weise konstruierten Verfahren haben die Konsistenzordnung k. Es lassen sich folglich recht einfach Verfahren sehr hoher Ordnung konstruieren, die lediglich eine Funktionsauswertung pro Zeitschritt benötigen. Dies ist ein großer Vorteil gegenüber beispielsweise Runge-Kutta-Verfahren. So benötigt ein explizites s-stufiges Runge-Kutta-Verfahren s Funktionsauswertungen und erreicht maximal die Ordnung s. Leider weisen die Adams-Bashforth-Verfahren jedoch sehr schlechte Stabilitätseigenschaften auf. In Abbildung 2.12 sind die Stabilitätsgebiete einiger Verfahren gezeigt. Es ist zu sehen, dass die Stabilitätsbereiche dieser Verfahren sehr klein sind. Ab dem Verfahren siebter Ordnung sind die Verfahren sogar in keinem Punkt mehr stabil.

Die Stabilitätseigenschaften können verbessert werden, indem das Interpolationspolynom p der Ordnung $k-1$ so konstruiert wird, dass auch der Wert $\mathbf{f}(\mathbf{x}_{n+1}, t_{n+1})$ interpoliert wird. In diesem Fall werden also die $k-1$ Punkte $\mathbf{f}(\mathbf{x}_{n+1}, t_{n+1})$, $\mathbf{f}(\mathbf{x}_{n-1}, t_{n-2})$, ..., $\mathbf{f}(\mathbf{x}_{n-k+3}, t_{n-k+3})$ und $\mathbf{f}(\mathbf{x}_{n-k+2}, t_{n-k+2})$ verwendet. In der Folge sind die so konstruierten Mehrschrittverfahren implizit und werden auch Adams-Moulton-Verfahren genannt[3]. Die Konsistenzordnung der Adams-Moulton-Verfahren ist jeweils um eins höher als bei den Adams-Bashforth-Verfahren. Die Verwendung von $k-1$ früheren Schritten führt also auf die Konsistenzordnung k. Ebenso sind die Stabilitätseigenschaften der Adams-Moulton-Verfahren besser als die der Adams-Bashforth-Verfahren. Nichts desto trotz sind die Stabilitätsgebiete immer noch relativ klein und schrumpfen mit steigender Ordnung (siehe Abbildung 2.13). Dieser Effekt lässt sich jedoch recht leicht einsehen. Die Interpolation (oder Extrapolation) von Datenpunkten führt nicht zwangsläufig zu einer guten Approximation der Funktion, auf der die Datenpunkte liegen. Das gilt insbesondere, wenn ein Polynom von hohem Grad gewählt wird[4].

Dennoch existieren lineare Mehrschrittverfahren mit guten Stabilitätseigenschaften. Curtiss und Hirschfelder erfanden 1952 die BDF-Verfahren zur Integration von steifen Differentialgleichungen. Im Gegensatz zu den bisherigen Mehrschrittverfahren werden bei den BDF-Verfahren nicht frühere Funktionswerte interpoliert, sondern frühere Lösungen [Curtiss & Hirschfelder, 1952]. Das Interpolationspolynom wird dann differenziert und anschließend an der Stelle t_{n+1} ausgewertet, um die neue Lösung $\mathbf{x}_{n+1}$ zu erhalten. Diese Vorgehensweise wird nun zunächst für den Fall von $k = 3$ verwendeten früheren Werten demonstriert, um anschließend die allgemeinen Formeln anzugeben.

Wie schon oben erwähnt, wird zunächst ein Polynom konstruiert, welches die Punkte $\mathbf{x}_n, \mathbf{x}_{n-1}$ und $\mathbf{x}_{n-2}$ interpoliert. Hierzu kann das Newton-Gregory-Verfahren mit der Hilfsvariable $s = \frac{t-t_{n+1}}{h}$ verwendet werden, wobei h den festen Abstand der Zeitpunkte t_n, t_{n-1} und t_{n-2} bezeichnet [Gerald & Wheatley, 1999]. Dann lässt sich das

[3] Es ist an dieser Stelle ganz interessant, dass die Entwicklung dieser Verfahren lediglich auf J.C. Adams zurückgehen

[4] Ein bekanntes Beispiel, das dieses Phänomen veranschaulicht, ist die Runge-Funktion [Epperson, 1987].

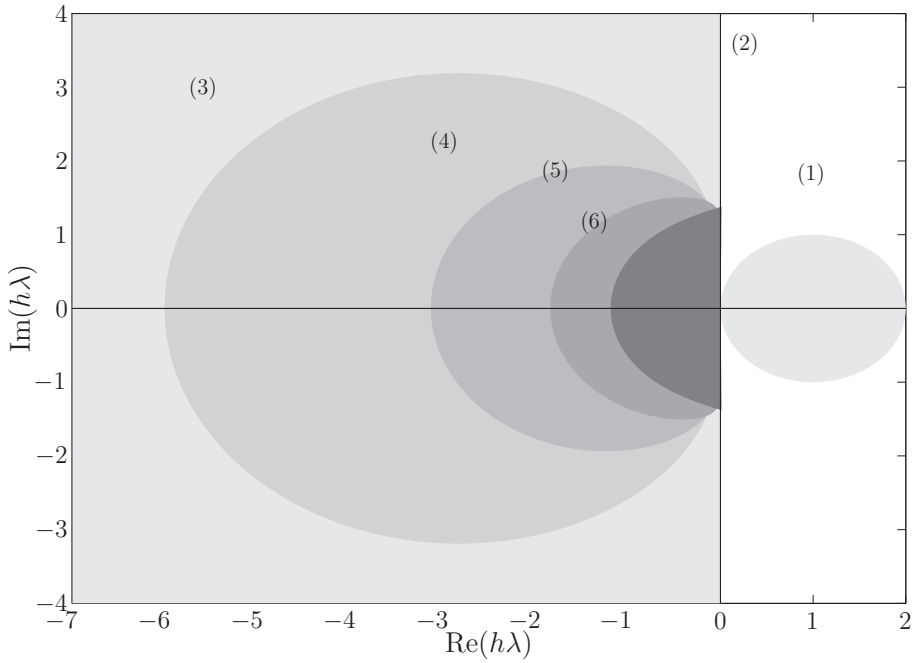


Abbildung 2.13: Stabilitätsbereiche (dunkle Bereiche, ausser bei (1)) der Adams-Moulton-Verfahren

Interpolationspolynom $\mathbf{q}(s)$ schreiben als

$$\mathbf{q}(s) = \mathbf{x}_{n+1} + \binom{s}{1}\nabla\mathbf{x}_{n+1} + \binom{s+1}{2}\nabla^2\mathbf{x}_{n+1} + \binom{s+2}{3}\nabla^3\mathbf{x}_{n+1}. \tag{2.169}$$

Dabei bezeichnet ∇ den *Rückwärtsdifferenzenoperator* und ist definiert durch

$$\nabla\mathbf{x}_i = \mathbf{x}_i - \mathbf{x}_{i-1} \tag{2.170}$$

beziehungsweise

$$\nabla^2\mathbf{x}_i = \nabla(\nabla\mathbf{x}_i) = \nabla(\mathbf{x}_i - \mathbf{x}_{i-1}) = \nabla\mathbf{x}_i - \nabla\mathbf{x}_{i-1} = \mathbf{x}_i - 2\mathbf{x}_{i-1} + \mathbf{x}_{i-2} \tag{2.171}$$

$$\nabla^3\mathbf{x}_i = \nabla(\nabla^2\mathbf{x}_i) = \mathbf{x}_i - 3\mathbf{x}_{i-1} + 3\mathbf{x}_{i-2} - \mathbf{x}_{i-3} \tag{2.172}$$

$$\vdots$$

$$\nabla^n\mathbf{x}_i = \sum_{k=0}^{n}(-1)^k\binom{n}{k}\mathbf{x}_{i-k}. \tag{2.173}$$

Ausmultiplizieren des Polynoms liefert

$$\mathbf{q}(s) = \mathbf{x}_{n+1} + s\nabla\mathbf{x}_{n+1} + (\frac{s^2}{2} + \frac{s}{2})\nabla^2\mathbf{x}_{n+1} + (\frac{s^3}{6} + \frac{s^2}{2} + \frac{s}{3})\nabla^3\mathbf{x}_{n+1}. \tag{2.174}$$

Differenziert man $\mathbf{q}(s)$ nun und fordert, dass

$$\dot{\mathbf{q}}(0) = \mathbf{f}(\mathbf{x}_{n+1}, t_{n+1}) \tag{2.175}$$

β	α_0	α_1	α_2	α_3	α_4	α_5	α_6
1	1	-1					
$\frac{2}{3}$	1	$-\frac{4}{3}$					
$\frac{6}{11}$	1	$-\frac{18}{11}$	$\frac{9}{11}$	$-\frac{2}{11}$			
$\frac{12}{25}$	1	$-\frac{48}{25}$	$\frac{36}{25}$	$-\frac{16}{25}$	$\frac{3}{25}$		
$\frac{60}{137}$	1	$-\frac{300}{137}$	$\frac{300}{137}$	$-\frac{200}{137}$	$\frac{75}{137}$	$-\frac{12}{137}$	
$\frac{60}{147}$	1	$-\frac{360}{147}$	$\frac{450}{147}$	$-\frac{400}{147}$	$\frac{225}{147}$	$-\frac{72}{147}$	$\frac{10}{147}$

Tabelle 2.4: Koeffizienten der BDF-Verfahren bis zur Ordnung 6

gilt, so erhält man

$$\mathbf{f}_{k+1} = \frac{1}{h}\left[\frac{11}{6}\mathbf{x}_{n+1} - 3\mathbf{x}_n + \frac{3}{2}x_{n-1} - \frac{1}{3}\mathbf{x}_{n-2}\right]. \tag{2.176}$$

Umformen dieser Gleichung liefert die Verfahrensvorschrift für das BDF3-Verfahren

$$\mathbf{x}_{n+1} = \frac{18}{11}\mathbf{x}_n - \frac{9}{11}\mathbf{x}_{n-1} + \frac{2}{11}\mathbf{x}_{n-2} + \frac{6h}{11}\mathbf{f}_{n+1}. \tag{2.177}$$

Im allgemeinen Fall hat das Interpolationspolynom $\mathbf{q}(s)$ die Form

$$\mathbf{q}(s) = \sum_{k=0}^{i} \binom{s+k-1}{k} \nabla^k \mathbf{x}_{n+1}. \tag{2.178}$$

Das gleiche Vorgehen wie oben führt auf die allgemeine Verfahrensvorschrift

$$\sum_{k=0}^{i} \alpha_k \mathbf{x}_{n+1-k} = h\beta\mathbf{f}_{n+1} \tag{2.179}$$

für das BDF-Verfahren, welches die i vorherigen Schritte verwendet (im Folgenden als BDFi-Verfahren bezeichnet). In Tabelle 2.4 sind die Koeffizienten für die ersten sechs BDF-Verfahren aufgelistet.

Ab dem Verfahren BDF7 sind die BDF-Verfahren instabil und somit wertlos. Abbildung 2.14 zeigt die Stabilitätsgebiete für die ersten sechs Verfahren. Auch hier ist das für Mehrschrittverfahren typische Verhalten zu beobachten, dass die Stabilitätsgebiete mit steigender Anzahl von verwendeten früheren Schritten stark schrumpfen. Nichts desto trotz sind die Stabilitätsgebiete erheblich größer als bei den Adams-Bashforth und Adams-Moulton-Verfahren. Insbesondere sind die BDFi-Verfahren A(α) stabil mit den folgenden Winkeln α [Ascher & Petzold, 1998]:

i	1	2	3	4	5	6
α	90°	90°	86°	73.3°	51.8°	17.8°

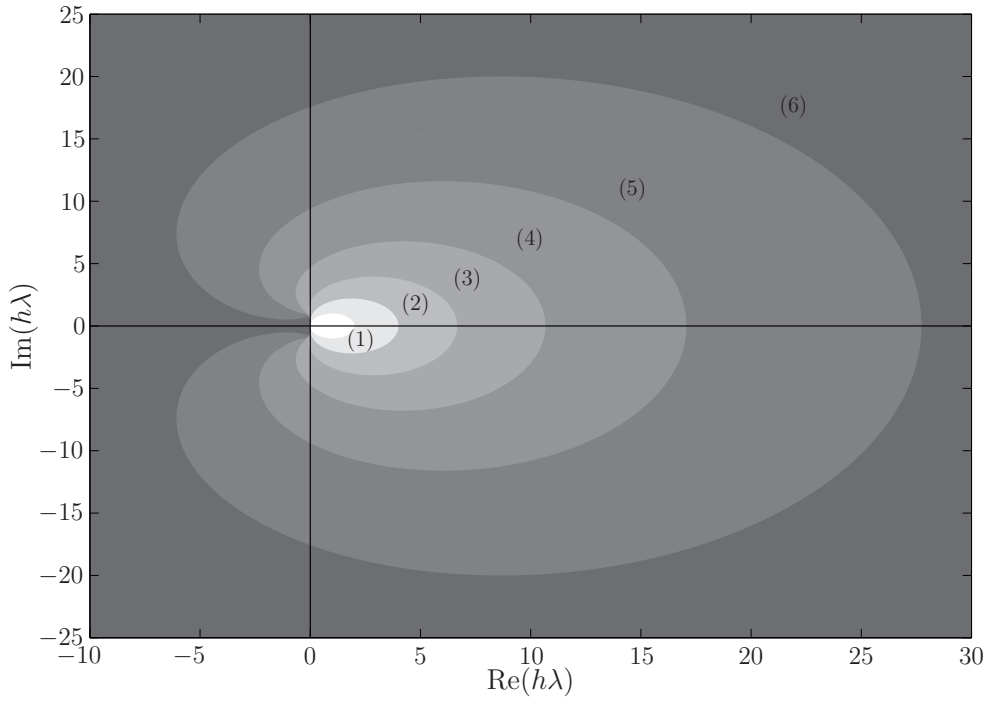


Abbildung 2.14: Stabilitätsbereiche (dunkle Bereiche) der BDF-Verfahren

Das BDF1 und das BDF2-Verfahren sind demnach A-stabil. Ab dem BDF3-Verfahren nimmt α dann schnell ab. Im Folgenden wird gezeigt, dass die Konsistenzordnung eines BDFi-Verfahrens i ist. Somit kann von den BDF-Verfahren mit einer Ordnung höher als zwei aufgrund der zweiten Dahlquist-Barriere auch keine A-Stabilität mehr erwartet werden. Die zweite Dahlquist Barriere besagt, dass A-stabile Mehrschrittverfahren eine Konsistenzordnung kleiner zwei besitzen. Darüber hinaus sind alle BDF-Verfahren L-stabil.

Da die Bestimmung der Ordnung von linearen Mehrschrittverfahren viel leichter ist als im Fall von impliziten Runge-Kutta-Verfahren, wird an dieser Stelle der allgemeine Fall und nicht nur der lineare Fall (wie bei dem Radau-IIa-Verfahren) behandelt. Den lokalen Diskretisierungsfehler erhält man, indem man die exakte Lösung $\mathbf{x}(t)$ in Gleichung (2.179) einsetzt und die linke Seite von der rechten subtrahiert:

$$\boldsymbol{\rho}_{n+1} = \sum_{k=0}^{i} \alpha_k \mathbf{x}(t_{n+1} - kh) - h\beta\dot{\mathbf{x}}(t_{n+1}). \tag{2.180}$$

Hierbei konnte $\mathbf{f}_{n+1}$ durch $\dot{\mathbf{x}}(t_{n+1})$ ersetzt werden, da die exakte Lösung natürlich die Differentialgleichung (2.155) erfüllt. Entwickelt man $\mathbf{x}(t_{n+1} - kh)$ und $\dot{\mathbf{x}}(t_{n+1})$ nun in eine Taylorreihe um t_{n+1}, so kann $\boldsymbol{\rho}_{n+1}$ als

$$\boldsymbol{\rho}_{n+1} = K_0\mathbf{x}(t_{n+1}) + K_1 h\dot{\mathbf{x}}(t_{n+1}) + \ldots + K_q h^q \mathbf{x}^{(q)}(t_{n+1}) + \ldots, \tag{2.181}$$

geschrieben werden, da $\mathbf{x}^{(q)}(t)$ ausschließlich linear auftaucht. Gilt jetzt

$$K_0 = K_1 = \ldots = K_p = 0 \tag{2.182}$$

sowie $K_{p+1} \neq 0$, so ist das Verfahren von der Ordnung p. Denn dann ist

$$\boldsymbol{\rho}_{n+1} = \mathcal{O}(h^{p+1}). \tag{2.183}$$

Die Koeffizienten K_j in der Gleichung (2.181) lassen sich zu

$$K_j = \frac{(-1)^j}{j!} \sum_{k=1}^{i} k^j \alpha_k \tag{2.184}$$

bestimmen. Nachrechnen zeigt dann, dass wie oben behauptet die Konsistenzordnung des BDFi-Verfahrens i ist.

Die Übertragung des BDFi-Verfahrens, zur Lösung von DAEs gelingt sehr einfach. Man beachte jedoch, dass in diesem Fall die Verfahrensvorschrift in die Modellgleichungen eingesetzt wird. In der Regel geht man genau andersherum vor. Ersetzt man in Gleichung (2.179) $\mathbf{f}_{n+1}$ durch $\dot{\mathbf{x}}_{n+1}$ und setzt dies in Gleichung (2.37) ein, so erhält man die diskretisierte DAE

$$\mathbf{F}(\frac{1}{\beta h} \sum_{k=0}^{i} \alpha_k \mathbf{x}_{n+1-k}, \mathbf{x}_{n+1}, t_{n+1}) = \mathbf{0}. \tag{2.185}$$

Die diskretisierte DAE wird dann mit Hilfe des Newton-Verfahrens gelöst. Die dabei verwendete Jacobi-Matrix $\mathbf{J}$ hat die Form

$$\mathbf{J} = \frac{\partial \mathbf{F}}{\partial \mathbf{x}} + \frac{1}{h} \cdot \frac{\partial \mathbf{F}}{\partial \dot{\mathbf{x}}}. \tag{2.186}$$

An dieser Stelle sei darauf hingewiesen, dass eine symbolische Jacobi-Matrix drastische Geschwindigkeitsvorteile gegenüber einer über finite Differenzen berechneten Jacobi-Matrix bietet. Ist $\mathbf{F}$ linear, so wird mit einer symbolischen Jacobi-Matrix lediglich ein Iterationsschritt im Newton-Verfahren benötigt. Werden hingegen finite Differenzen verwendet, um eine Approximation erster Ordnung der Jacobi-Matrix zu berechnen, so sind in der Praxis bis zu vier Iterationen nötig, um den numerischen Fehler des Newton-Verfahrens unter den Diskretisierungsfehler des BDF-Verfahrens zu drücken. Die Berechnung einer symbolischen Jacobi-Matrix kann jedoch bei komplexen Problemen sehr aufwändig sein [Dürrbaum *et al.*, 2002]. Es ist also von dem jeweiligen Problem abhängig, ob es sinnvoll ist, die Jacobi-Matrix symbolisch zu berechnen.

Einen möglichen Ausweg kann hier die so genannte *automatische Differentiation* bieten, welche automatisiert bei der Auswertung einer Funktion neben dem Funktionswert auch die Ableitung der Funktion liefert. Entsprechende Ansätze findet man zum Beispiel in [Campbell & Hollenbeck, 1997; Griewank & Walther, 2006]. In [Nedialkov & Pryce, 2007] wird DAETS, ein numerisches Integrationsverfahren für DAEs vorgestellt. DAETS verwendet automatische Differentiation und ist unter gewissen Umständen konventionellen Verfahren wie zum Beispiel DASSL überlegen. Beispielsweise können Probleme mit Störungsindex fünfzig und höher gelöst werden. Bei der Verwendung von DASSL kann die Jacobi-Matrix wahlweise durch eine vom Benutzer bereitgestellte Routine oder durch finite Differenzen approximiert werden. Weiterhin verwendet DASSL die BDF-Verfahren bis zur Ordnung fünf zur Lösung

von DAEs. Der Algorithmus verwendet folglich eine Ordnungssteuerung, um den Berechnungsaufwand zu senken. Eine Ordnungssteuerung ist leicht zu implementieren, da in diesem Fall nur die Anzahl der verwendeten vorherigen Lösungen schwankt. Da jedoch der Berechnungsaufwand für alle BDF-Verfahren fast identisch ist (es ist jeweils eine Funktionsauswertung pro Iterationsschritt notwendig), ist solch eine Ordnungssteuerung bei flüchtiger Betrachtung nicht sehr sinnvoll. In der Praxis hat sich dieses Vorgehen jedoch, wie in [Shampine, 1994] gezeigt, bewährt.

Selbiges gilt für den in DASSL integrierten Startalgorithmus. DASSL beginnt mit dem BDF1-Verfahren (implizites Euler-Verfahren), da in der Regel nur ein Startwert vorgegeben ist. Unter Verwendung der neu berechneten Lösung ist es dann möglich, das BDF2-Verfahren zu verwenden. Im nächsten Schritt wäre es dann beispielsweise möglich, das BDF3-Verfahren zu verwenden. Auf diese Art und Weise kann also sukzessive die Ordnung des Verfahrens erhöht werden. Dieser Startalgorithmus ist ebenfalls auf der einen Seite einfach zu implementieren, leidet aber auf der anderen Seite darunter, dass dieser Startalgorithmus (zumindest auf den ersten Blick) nicht besonders effizient ist. Dies liegt daran, dass die BDF-Verfahren mit kleiner Ordnung nicht besonders genau sind und die kleinste verwendete Ordnung die Konsistenzordnung bestimmt. Sind die benötigten Startwerte zum Beispiel mit einer Genauigkeit von $\mathcal{O}(2)$ bestimmt worden, so ist auch der Fehler der Gesamtlösung von der Ordnung $\mathcal{O}(2)$. In [Shampine & Zhang, 1990] wird jedoch gezeigt, dass es auch aus theoretischer Sicht sinnvoll ist ein Mehrschrittverfahren durch ansteigende Ordnung zu starten.

Eine Implementierung von DASSL mit einem Interface zu MATLAB findet beispielsweise im SUNDIALS Softwarepaket [Hindmarsh *et al.*, 2005].

Vergleich der Radau-IIa und BDF-Verfahren

Wie in den beiden vorherigen Abschnitten gezeigt, lassen sich DAEs sowohl mit Radau-IIa als auch mit BDF-Verfahren lösen. Folglich stellt sich die Frage, welches Vorgehen das effizientere ist. Diese Frage lässt sich jedoch nicht allgemeingültig beantworten. An dieser Stelle soll daher lediglich die Genauigkeit der Verfahren etwas genauer untersucht werden.

Zunächst wird nochmals die Genauigkeit der BDF-Verfahren betrachtet. Der führende Koeffizient ε des lokalen Diskretisierungsfehlers $\boldsymbol{\rho}_{n+1}$ kann für das BDFi-Verfahren allgemein durch Gleichung (2.184) mit $j = i + 1$ berechnet werden. Dies liefert

$$
\begin{aligned}
i = 2: &\quad \varepsilon = K_3 = -\frac{1}{6}\left(-\frac{4}{3} + 8 \cdot \frac{1}{3}\right) = -\frac{9}{2},\\
i = 3: &\quad \varepsilon = K_4 = \frac{1}{24}\left(-\frac{18}{11} + 16 \cdot \frac{9}{11} - 81 \cdot \frac{2}{11}\right) = -\frac{3}{22},\\
i = 4: &\quad \varepsilon = K_5 = -\frac{1}{120}\left(-\frac{48}{25} + 32 \cdot \frac{36}{25} - 243 \cdot \frac{16}{25} + 1024 \cdot \frac{3}{25}\right) = -\frac{12}{125},\\
i = 5: &\quad \varepsilon = K_6 = \frac{1}{720}\left(-\frac{300}{137} + \ldots - 15625 \cdot \frac{12}{137}\right) = -\frac{30}{411}.
\end{aligned}
$$

Für das Radau-IIa-Verfahren dritter Ordnung lässt sich dieser Fehlerkoeffizient ε (zumindest für lineare Probleme) durch Subtraktion der Gleichung (2.166) von Gleichung (2.167) zu $\varepsilon = \frac{1}{72}$ bestimmen. Eine analoge Betrachtung führt auf den Fehlerkoeffizienten $\varepsilon = \frac{1}{7200}$ für das Radau-IIa-Verfahren fünfter Ordnung. Die Fehlerkoeffizienten der BDF-Verfahren sind also vergleichsweise groß. Als direkte Konsequenz hieraus kann bei den Radau-IIa-Verfahren eine größere Schrittweite gewählt werden, um dieselbe Genauigkeit zu erreichen. Dies kann in der Praxis den Vorteil der BDF-Verfahren, weniger Funktionsauswertungen für einen Integrationsschritt zu benötigen, oftmals ausgleichen.

Trotzdem ist DASSL weiter verbreitet als beispielsweise RADAU5 und das Standardverfahren in vielen Simulationsumgebungen. Ein Grund hierfür ist, dass die meisten praktischen Probleme steif sind. Das Radau-IIa-Verfahren ist jedoch insbesondere bei nicht-steifen Problemen effizienter ist als DASSL. Darüber hinaus ist der Code von DASSL durch seine Verbreitung sehr gut getestet und ist damit besser geeignet für kommerzielle Simulationsumgebungen. Zum Teil wird DASSL in diesen Simulationsumgebungen jedoch nur eingesetzt, um ODEs zu lösen. In diesem Fall wird das mathematische Modell immer in eine ODE transformiert[5]. Der Grund hierfür wird im Folgenden kurz dargelegt.

Wird Gleichung (2.185) mit dem Newton-Verfahren gelöst, so führt die Jacobi-Matrix aus Gleichung (2.186) auf das lineare Gleichungssystem

$$(\frac{\partial \mathbf{F}}{\partial \mathbf{x}} + \frac{1}{h} \cdot \frac{\partial \mathbf{F}}{\partial \dot{\mathbf{x}}}) \cdot \boldsymbol{\delta}^l = \mathbf{F}(\frac{1}{\beta h} \sum_{k=0}^{i} \alpha_k \mathbf{x}^l_{n+1-k}, \mathbf{x}^l_{n+1}, t_{n+1}) \tag{2.187}$$

in $\boldsymbol{\delta}^l$ im l-ten Iterationsschritt. Multiplizieren mit h liefert

$$\underbrace{(h \cdot \frac{\partial \mathbf{F}}{\partial \mathbf{x}} + \frac{\partial \mathbf{F}}{\partial \dot{\mathbf{x}}})}_{\tilde{\mathbf{J}}} \cdot \boldsymbol{\delta}^l = h \cdot \mathbf{F}(\frac{1}{\beta h} \sum_{k=0}^{i} \alpha_k \mathbf{x}^l_{n+1-k}, \mathbf{x}^l_{n+1}, t_{n+1}). \tag{2.188}$$

Wegen

$$\lim_{h \to 0} \tilde{\mathbf{J}} = \frac{\partial \mathbf{F}}{\partial \dot{\mathbf{x}}} = \mathbf{J}_{\dot{\mathbf{x}}}, \tag{2.189}$$

degeneriert die Systemmatrix $\tilde{\mathbf{J}}$ des linearen Gleichungssystems im Falle einer gegen Null strebenden Schrittweite zu $\mathbf{J}_{\dot{\mathbf{x}}}$. Im Falle einer ODE ist $\mathbf{J}_{\dot{\mathbf{x}}}$ stets regulär, im Falle einer DAE hingegen stets singulär. Dies impliziert, dass das lineare Gleichungssystem aus Gleichung (2.188) im Falle einer DAE bei kleinen Schrittweiten sehr schlecht konditioniert ist.

[5] Kann die DAE mit Hilfe der Dummy Derivatives nur auf Störungsindex $\nu_p = 1$ reduziert werden, so werden die verbleibenden algebraischen Gleichungen während der Funktionsauswertung mit einem Newton-Verfahren gelöst.

KAPITEL 3

Gleichungsbasierte Modellreduktion

In diesem Kapitel wird die gleichungsbasierte Modellreduktion vorgestellt. Gleichungsbasierte Modellreduktion stellt eine Methode zur Approximation von DAEs durch weniger komplexe DAEs dar. Somit eignet sich dieses Modellreduktionsverfahren für die Reduktion von Modellen, die der objektorientierten Modellbildung entstammen. Dabei werden auch die im Rahmen dieser Arbeit entwickelten Erweiterungen vorgestellt. So wird die Reduktion für Echtzeitsimulationen ebenso vorgestellt, wie die Erweiterungen des Approximationsalgorithmus für multiple und intervallwertige Szenarien. Außerdem wird ein neues Ranking-Verfahren, basierend auf der Sensitivitätsanalyse, präsentiert.

3.1 Einleitung

Im Folgenden wird der Begriff *Komplexität* einer DAE der Form

$$\mathbf{F}(\mathbf{x}, \dot{\mathbf{x}}, t) = 0 \tag{3.1}$$

als Maß für die Anzahl der notwendigen Rechenoperationen zur Auswertung von $\mathbf{F}$ verwendet. Gleichungsbasierte Modellreduktion approximiert eine gegebene DAE durch eine weniger komplexe DAE. Zur Auswertung der reduzierten DAE sind also weniger Rechenoperation notwendig als zur Auswertung der ursprünglichen DAE. Dabei verbleibt die Differenz zwischen der Lösung der ursprünglichen DAE und der Lösung der reduzierten DAE innerhalb einer vorgegebenen Toleranz. Diese Toleranz kann auf verschiedene Arten definiert werden. Üblich ist die Festlegung einer maximalen Abweichung $\boldsymbol{\epsilon}_{\text{Fehler}}$, die zu jedem Zeitpunkt eingehalten werden muss. Das zugrunde liegende Verfahren wird in [Wichmann, 2004] präsentiert.

Ein Vektorfeld $\mathbf{F}$ aus Gleichung (3.1) lässt sich stets schreiben als

$$F_i(\mathbf{x}, \dot{\mathbf{x}}, t) = \sum_{k=1}^{l_i} \tau_{k_i}(\mathbf{x}, \dot{\mathbf{x}}, t), \quad 1 \leq i \leq N. \tag{3.2}$$

Eine Funktion τ_{k_i} wird dabei im Folgenden als Term bezeichnet. Dann ist l_i die Anzahl der Terme in F_i und τ_{k_i} der k-te Term in F_i. Man beachte, dass der Term τ_{k_i} eine beliebige von $\mathbf{x}$, $\dot{\mathbf{x}}$ und t abhängige Funktion sein kann. Das bedeutet insbesondere, dass τ_{k_i} selber wieder aus einer Summe von Subtermen bestehen kann und so weiter. Die Menge aller dieser Terme und Subterme ist $\mathcal{T}$. Die Manipulation eines Terms aus $\mathcal{T}$, welche die Anzahl der Rechenoperationen zu dessen Auswertung reduziert, wird als *Reduktion* bezeichnet. Mögliche Arten dieser Manipulation werden *Reduktionstechniken* genannt. Mögliche Reduktionstechniken sind beispielsweise die Vernachlässigung von Termen oder Ersetzung von Termen durch Konstanten. Diese und weitere Reduktionstechniken werden in Abschnitt 3.2 vorgestellt. Die Menge aller möglichen Reduktionen für eine Reduktionstechnik ist $\mathcal{K}$.

Der Approximationsalgorithmus besteht aus zwei Phasen. Zunächst werden diejenigen Reduktionen identifiziert, die nur einen geringen Einfluss auf die Lösung der DAE haben. Hierzu sind Verfahren notwendig, die den Einfluss der einzelnen Reduktionen auf die Lösung schätzen. Einige dieser so genannten *Ranking-Verfahren* werden in Abschnitt 3.3 präsentiert. Unter anderem wird dort ein neues äußerst effektives Ranking-Verfahren vorgestellt, das auf einer Sensitivitätsanalyse beruht.

In der zweiten Phase werden die Reduktionen durchgeführt. Die Reihenfolge der Reduktionen wird dabei durch das vorherige Ranking bestimmt. Hier bezeichnet für $\kappa \in \mathcal{K}$

$$\mathbf{F}^{\kappa} = \mathbf{0} \tag{3.3}$$

die DAE, welche durch Durchführung der Reduktion κ entsteht. Die Durchführung der Reduktionen führt auf eine DAE, welche weniger komplex ist. Während die Komplexität sinkt, wird dafür Sorge getragen, dass die Lösung der reduzierten DAE innerhalb der vorgegebenen Fehlertoleranz verbleibt.

In Abschnitt 3.2 werden einige mögliche Reduktionstechniken vorgestellt. Hierzu zählt unter anderem die Ähnlichkeitsreduktion, die erstmals in [Mikelsons *et al.*, 2009b] eingesetzt wurde. Anschließend werden in Abschnitt 3.3 drei Ranking-Verfahren diskutiert. Neben zwei in [Wichmann, 2003] entwickelten Ranking-Verfahren wird hier auch das im Rahmen dieser Arbeit entwickelte Sensitivitätsranking eingeführt. In Abschnitt 3.4 wird der Reduktionsalgorithmus schließlich für eine vorgegebene Fehlergrenze vorgestellt und die Erweiterungen für multiple Szenarien und intervallwertige Szenarien präsentiert. Dieser Algorithmus wurde im Rahmen dieser Arbeit weiterentwickelt [Mikelsons *et al.*, 2009a]. Anstatt einer vorgegebenen Fehlertoleranz kann nun alternativ oder zusätzlich eine Echtzeitumgebung vorgegeben werden. Echtzeitumgebung bezeichnet dabei eine Echtzeit-Hardware sowie einen Takt. Das ursprüngliche Modell wird dann soweit reduziert, dass es in dem vorgegebenen Takt auf der vorgegebenen Hardware simuliert werden kann. In Abschnitt 3.5 wird der neue Approximationsalgorithmus präsentiert.

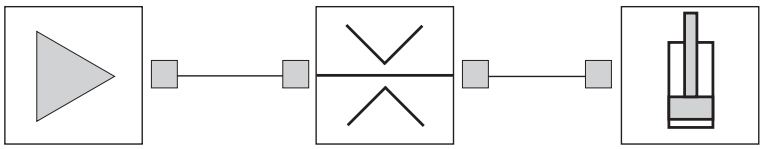


Abbildung 3.1: Objektdiagramm eines einfachen hydraulischen Systems

3.2 Reduktionstechniken

Aus der Literatur sind die Reduktionstechniken Streichen von Termen (Termreduktion), Ersetzen von Termen durch Konstanten (Termsubstitution) sowie Linearisieren von Termen bekannt [Borchers, 1998]. In diesem Abschnitt wird neben diesen bekannten Reduktionstechniken eine weitere neue Reduktionstechnik, die so gennante Ähnlichkeitsbetrachtung, präsentiert. Darüber hinaus wurde die Linearisierung bisher nur im Kontext von analogen Schaltungen verwendet und wird in dieser Arbeit auf die Reduktion beliebiger DAEs übertragen.

Alle in der vorliegenden Arbeit verwendeten Reduktionstechniken werden im Folgenden kurz erklärt und an einem einfachen Beispiel vorgeführt.

Zur Beschreibung der Reduktionstechniken dient ein einfaches hydraulisches System, bestehend aus einer Druckquelle, einer Blende und einem Zylinder. In Abbildung 3.1 ist das entsprechende Objektdiagramm gezeigt. Eine Konstantdruckquelle ist über eine Blende mit einem Zylinder verbunden. Dabei ist der Elastizitätsmodul des Fluids temperatur- sowie druckabhängig modelliert. Mit den in Abschnitt 2.1.2 vorgestellten Algorithmen lässt sich das mathematische Modell schreiben als

$$v = C_2, \tag{3.4}$$

$$\dot{s} = v, \tag{3.5}$$

$$\dot{x} = \frac{1}{m}(\frac{\pi}{4} p \cdot d_P^2 + m \cdot g), \tag{3.6}$$

$$\dot{p} = (\frac{\pi}{4} C_2 \cdot d_P^2 + (p - p_S) \cdot G) \cdot K / (V_A + \frac{\pi}{4} C_1 \cdot d_P^2), \tag{3.7}$$

$$\tag{3.8}$$

wobei

$$K = \beta_0 \beta_1 e^{(T+273)\cdot\beta_2} + (p - p_0)\mu_0, \tag{3.9}$$

$$C_1 = \begin{cases} l, & \text{falls } 0 \leq s - l \leq 0 \\ s, & \text{sonst} \end{cases}, \tag{3.10}$$

$$C_2 = \begin{cases} 0, & \text{falls } 0 \leq s - l \leq 0 \\ x, & \text{sonst} \end{cases}, \tag{3.11}$$

gilt. Das mathematische Modell wurde dabei mit Hilfe von Modelica und der Modelica-Standard-Bibliothek erstellt. Dabei ist

- s: die Position des Kolbens,
- v: die Geschwindigkeit des Kolbens,
- p: der Druck im Zylinder,
- x: Hilfsvariable, die die Geschwindigkeit des Kolbens beschreibt, ohne die Anschläge zu berücksichtigen,
- m: die Masse des Kolbens (1000kg),
- p_S: der Druck der Druckquelle ($2 \cdot 10^7$Pa),
- p_0: Referenzdruck (10^5Pa),
- g: die Erdbeschleunigung ($9,81$m/s),
- d_P: der Durchmesser des Kolbens ($0,05$m),
- l: die Länge des Zylinders($0,5$m),
- G: der Leitwert der Drossel ($10^{-10}\text{m}^3/(\text{s} \cdot \text{Pa})$),
- V_A: Totvolumen im Anschluss A des Zylinders ($2 \cdot 10^{-6}$Pa),
- $\beta_0, \beta_1, \beta_2$: Koeffizienten zur Berechnung des Elastizitätsmoduls ($1,4 \cdot 10^9$, $5,64$,$-0,0059$),
- μ_0: der Reibungskoeffizient (10),
- T: die Temperatur des Fluids (45K).

3.2.1 Termreduktion

Termreduktion bezeichnet das Streichen von Termen aus der DAE. Soll zum Beispiel in dem hydraulischen Beispiel die Gravitation vernachlässigt werden, so wird der zweite Term der rechten Seite in Gleichung (3.6) gestrichen. Die resultierende Gleichung lautet demnach einfach

$$\dot{x} = \frac{1}{m}(\frac{\pi}{4} p \cdot d_P^2). \tag{3.12}$$

Es ist auch möglich, Terme in verschachtelten Ausdrücken zu streichen. Soll in dem Beispiel der Temperatureinfluss auf den Elastizitätsmodul vernachlässigt werden, so wird der Exponent der Exponentialfunktion in Gleichung (3.9) gestrichen. Folglich wird K berechnet durch

$$K = \beta_0 \beta_1 e^0 + (p - p_0)\mu_0 = \beta_0 \beta_1 + (p - p_0)\mu_0. \tag{3.13}$$

Die Termreduktion ist eine äußerst effektive Reduktionsmethode, da die DAE schnell an Komplexität verliert. Darüber hinaus ist die reduzierte DAE in der Regel einfach zu interpretieren. Auf der anderen Seite weicht die Lösung der ursprünglichen DAE mitunter schon nach wenigen Reduktionen stark von der Lösung der reduzierten DAE ab.

3.2.2 Termsubstitution

Bei der Termsubstitution wird der betrachtete Term durch eine Konstante ersetzt. Diese Konstante kann auf verschiedene Arten berechnet werden. Die intuitivste Möglichkeit ist es, den Mittelwert des Terms über die Gesamtsimulationszeit zu verwenden. Folglich muss vor Beginn der Reduktion eine Simulation durchgeführt werden. Dies ist jedoch kein Nachteil gegenüber der Termreduktion, da diese Simulation im Ranking ohnehin notwendig sein wird.

Schwankt die Änderungsrate des Drucks im Zylinder während der Simulation nur wenig, so kann diese beispielsweise durch ihren Mittelwert $\dot{p}_{\text{Mittel}}$ über die Simulation ersetzt werden. Folglich würde Gleichung (3.7) in

$$\dot{p}_{\text{Mittel}} = (\frac{\pi}{4} C_2 \cdot d_P^2 + (p - p_S) \cdot G) \cdot K / (V_A + \frac{\pi}{4} C_1 \cdot d_P^2) \tag{3.14}$$

übergehen. Man beachte, dass $\dot{p}_{\text{Mittel}}$ dann ein numerischer Wert und keine Variable mehr ist.

Termsubstitution ist ähnlich effektiv wie die Termreduktion. Unter Umständen kann es allerdings passieren, dass das reduzierte Modell nur noch schwer zu interpretieren ist. Dies liegt daran, dass nach der Reduktion nicht mehr offensichtlich ist, welcher numerische Wert aus welcher Reduktion herrührt. Ähnlich wie bei der Termreduktion steigt auch hier der Fehler in der Regel schnell an.

3.2.3 Linearisierung von Termen

In [Borchers, 1997] wird Linearisierung als Reduktionstechnik vorgeschlagen[1]; im Folgenden wird diese Reduktionstechnik jedoch modifiziert. In der ursprünglichen Variante wurde der betrachtete Term τ_{k_i} bezüglich der Variablen $\mathbf{x}$ und $\dot{\mathbf{x}}$ linearisiert. Der linearisierte Ausdruck für einen Entwicklungspunkt $(\mathbf{x}_0, \dot{\mathbf{x}}_0)$ lautet folglich

$$\tau_{\text{Lin}} = \sum_{i=0}^{N} (x_i - x_{0,i}) \frac{\partial \tau}{\partial x_i}(\mathbf{x}_0, \dot{\mathbf{x}}_0) + \sum_{i=0}^{N} (\dot{x}_i - \dot{x}_{0,i}) \frac{\partial \tau}{\partial \dot{x}_i}(\mathbf{x}_0, \dot{\mathbf{x}}_0) + \tau(\mathbf{x}_0, \dot{\mathbf{x}}_\mathbf{0}). \tag{3.15}$$

Ein Nachteil dieser Methode ist, dass die so berechneten linearisierten Ausdrücke sehr komplex werden können. Zudem funktioniert dieses Vorgehen nicht bei Termen mit Funktionen, in denen nur Parameter auftauchen, da diese einfach verschwinden würden. Daher wurde die Reduktionstechnik in dieser Arbeit modifiziert.

Linearisieren ist an solchen Stellen sinnvoll, wo eine komplexe Funktion durch eine einfachere ersetzt werden kann. Linearisierung kommt deshalb nur dann zum Einsatz, wenn τ eine komplexe Funktion, wie zum Beispiel eine trigonometrische Funktion oder die Wurzelfunktion ist. In diesem Fall wird diese Funktion bezüglich ihres Arguments linearisiert. Als Entwicklungspunkt kann der Mittelwert des Arguments über die Simulation dienen.

[1] Einschränkend muss hier erwähnt werden, dass diese Arbeit nur die Vereinfachung von Analogschaltungen behandelte und hierfür die AC- und DT-Analyse nutzte

Anstatt die Exponentialfunktion in Gleichung (3.9) zu streichen, wie bei der Termreduktion, könnte dieser Term auch linearisiert werden. Die Linearisierung von e^x mit Null als Entwicklungspunkt liefert $1 + x$. Folglich lautet Gleichung (3.9) nach Anwendung der Linearisierung

$$K = \beta_0\beta_1(1 + (T + 273) \cdot \beta_2) + (p - p_0)\mu_0. \tag{3.16}$$

Die Verwendung von Linearisierung als Reduktionstechnik senkt den Komplexitätsgrad natürlich langsamer als die Verwendung von Termreduktion oder Termsubstitution. Im Gegenzug steigt die Differenz zwischen der Lösung der ursprünglichen DAE und der Lösung reduzierten DAE auch langsamer an.

3.2.4 Ähnlichkeitsbetrachtungen

Die Ähnlichkeitsbetrachtungen basieren darauf, zwei Variablen zu finden, die während der Simulation zu allen Zeitpunkten ähnlich sind. Ähnlich soll in diesem Zusammenhang bedeuten, dass der numerische Wert der beiden Variablen nahe beieinander liegt. Wird ein solches Paar von Variablen a_1 und a_2 gefunden, so wird in der gesamten DAE a_2 durch a_1 ersetzt. In der Folge ist die DAE natürlich überbestimmt und eine Gleichung kann aus der DAE gestrichen werden. Die Gleichung, in der a_2 auftaucht und zu der kleinsten Differenz zwischen der Lösung der ursprünglichen DAE und der Lösung der reduzierten DAE führt, ist diejenige, die gestrichen wird. Das Auffinden der Variablenpaare kann automatisiert geschehen. Alternativ können bei der Modellierung auch Variablen angegeben werden, bei denen eine Ähnlichkeitsbetrachtung durchgeführt werden soll.

Ähnlichkeitsbetrachtungen stellen somit eine vollkommen neue Reduktionstechnik dar. Alle bisherigen Reduktionstechniken manipulieren jeweils einzelne Terme. Die Verwendung von Ähnlichkeitsbetrachtungen ändert die Struktur der DAE. Während also die bisherigen Reduktionstechniken die Komplexität der DAE senkten, senken die Ähnlichkeitsbetrachtungen die Dimension der DAE.

3.3 Ranking-Verfahren

Im vorherigen Abschnitt wurden einige Reduktionstechniken eingeführt. Gleichungsbasierte Modellreduktion basiert darauf, zu schätzen, wie stark sich die Lösung einer DAE ändert, falls eine Reduktion durchgeführt wird. Hierfür werden Ranking-Verfahren verwendet. Es ist in der Regel so, dass mit steigendem zeitlichen Aufwand die Genauigkeit der Schätzung zunimmt. Beispielsweise erhält man das perfekte Ranking durch eine vollständige Simulation nach Durchführung der betrachteten Reduktion. Es gilt also, ein Ranking-Verfahren zu finden, welches einen guten Kompromiss zwischen Laufzeit und Genauigkeit bietet. Ein Ranking-Verfahren $\mathcal{R}$ ist eine Abbildung

$$\mathcal{R} : (\mathbf{F}, \kappa) \mapsto r_\kappa, \tag{3.17}$$

welche eine Reduktion κ und ein Vektorfeld $\mathbf{F}$ einer DAE auf eine reelle Zahl r_κ abbildet. Diese reelle Zahl ist ein Maß für den Einfluss der Reduktion auf die Lösung der DAE und wird *Rankingwert* genannt.

Im Folgenden bezeichnet

$$\mathcal{N}(\mathbf{F}(\mathbf{x}, \dot{\mathbf{x}}, t), \mathbf{u}, \mathcal{I}) \tag{3.18}$$

die von einem numerischen Integrationsverfahren $\mathcal{N}$ (zum Beispiel DASSL) in einem Intervall $\mathcal{I} = [t_0, T]$ berechnete Lösung einer DAE mit Systemeingängen $\mathbf{u}$. Folglich ist $\mathbf{u}$ ein auf dem Intervall $\mathcal{I}$ definiertes Vektorfeld. Dieses Vektorfeld $\mathbf{u}$ wird zusammen mit den Anfangswerten $\mathbf{x}_0$ und einem Parametersatz $\mathbf{p}$ als ein *Szenario* $\mathcal{S}$ bezeichnet. In den meisten Fällen ist nicht die gesamte Lösung der DAE von Interesse, sondern lediglich einige Komponenten. Die numerisch berechnete Lösung

$$\mathbf{x}^\star = \begin{bmatrix} \mathbf{x}^\star_{\text{out}} \\ \mathbf{x}^\star_{\text{int}} \end{bmatrix} = \mathcal{N}(\mathbf{F}(\mathbf{x}, \dot{\mathbf{x}}, t), \mathcal{S}) \tag{3.19}$$

besteht daher aus zwei Komponenten. In $\mathbf{x}^\star_{\text{out}}$ ist die Lösung der N_{out} Ausgangsvariablen enthalten, während $\mathbf{x}^\star_{\text{int}}$ aus den verbleibenden internen Variablen besteht.

Also ist r_κ eine Schätzung für den Unterschied zwischen

$$\mathcal{N}(\mathbf{F}(\mathbf{x}, \dot{\mathbf{x}}, t), \mathcal{S}) \tag{3.20}$$

und

$$\mathcal{N}(\mathbf{F}^\kappa(\mathbf{x}, \dot{\mathbf{x}}, t), \mathcal{S}). \tag{3.21}$$

Man unterscheidet zwischen absoluten und relativen Ranking-Verfahren. Während bei absoluten Ranking-Verfahren der Rankingwert r_κ der Norm einer Schätzung des absoluten Fehlers in $\mathbf{x}^\star_{\text{out}}$ entspricht, stellt dieser bei relativen Ranking-Verfahren lediglich ein abstraktes Maß für den Einfluss der Reduktion auf $\mathbf{x}^\star$ dar. In diesem Fall kann also der Einfluss der Reduktionen auf die Lösung der DAE miteinander verglichen werden, es ist jedoch nicht möglich, anhand des Rankingwerts Rückschlüsse auf die absolute Größe des zu erwartenden Fehlers zu ziehen.

In [Wichmann, 2003] werden verschiedene Ranking-Verfahren vorgestellt. Aus dieser Arbeit werden im Folgenden das Residuenranking und das Einschrittranking kurz diskutiert. Danach wird das Sensitivitätsranking vorgestellt. Dieses Ranking-Verfahren wurde im Rahmen der vorliegenden Arbeit entwickelt und hat sich als äußerst effektiv erwiesen. In Abschnitt 3.3.4 wird die Güte der verschiedenen Ranking-Verfahren anhand eines Beispiels untersucht.

3.3.1 Residuenranking

Hinter dem Residuenranking steckt die Idee, nicht die Lösungen der ursprünglichen DAE und der reduzierten DAE direkt zu vergleichen, sondern zu messen „wie gut“ die Lösung der ursprünglichen DAE die reduzierte DAE löst. Man benötigt also zunächst

die Lösung $\mathbf{x}^\star$ der ursprünglichen DAE, im Folgenden *Referenzlösung* genannt. Für die Referenzlösung gilt

$$\|\mathbf{F}(\mathbf{x}^\star(t_i), \dot{\mathbf{x}}^\star(t_i), \mathbf{u}(t_i), t)\| < \eta, \quad 1 \leq i \leq n_t, \tag{3.22}$$

mit einer im Lösungsverfahren vorgegebene Toleranz η (siehe Abschnitt 2.2.3). Soll nun der Einfluss einer Reduktion $\kappa \in \mathcal{K}$ geschätzt werden, so wird die Referenzlösung $\mathbf{x}^\star$ in die reduzierte DAE aus Gleichung (3.3) eingesetzt. Da die Lösung $\mathbf{x}^\star$ nur an Zeitpunkten $t_0, \ldots, t_{n_t} = T$ im Intervall $\mathcal{I}$ bekannt ist, ergeben sich die n_t Residuen

$$\varrho_i = \left\|\mathbf{F}^\kappa(\mathbf{x}^\star(t_i), \dot{\mathbf{x}}^\star(t_i), \mathbf{u}(t_i), t_i)\right\|, \quad 0 \leq i \leq n_t, \tag{3.23}$$

mit einer beliebigen Norm $\|\cdot\|$. Das Residuum ϱ_i stellt ein Maß dafür dar, wie sehr sich die Referenzlösung zum Zeitpunkt t_i von der Lösung der Gleichung (3.3) unterscheidet. Damit ist

$$\mathcal{R}_{\text{Res}}(\mathbf{F}, \kappa) = \left\| \begin{bmatrix} \varrho_1 \\ \vdots \\ \varrho_{N+1} \end{bmatrix} \right\| \tag{3.24}$$

eine Schätzung für den Einfluss der Reduktion $\kappa \in \mathcal{K}$ auf dem gesamten Intervall $\mathcal{I}$. Die verwendeten Normen sind hierbei frei wählbar. Sinnvoll sind hier zum Beispiel die Euklidische- oder die Maximumsnorm. Bei der inneren Norm (Berechnung von ρ_i) sorgt die Maximumsnorm dafür, dass nur die Komponente des Residuums mit dem größten Fehler berücksichtigt wird, während bei der euklidischen Norm der Fehler in allen Komponenten berücksichtigt wird. Analoges gilt für die äußere Norm (Berechnung $\mathcal{R}_{\text{Res}}$). Aus der Berechnung ist ferner ersichtlich, dass das Residuenranking ein relatives Ranking-Verfahren ist. Aus $\mathcal{R}_{\text{Res}}(\mathbf{F}, \kappa)$ ist es nicht möglich auf den Wert des absoluten Fehlers zwischen der Referenzlösung und der Lösung von Gleichung (3.3) zu schließen.

Darüber hinaus kann das Residuenranking sehr ungenaue Schätzungen liefern. Dieser Fall tritt beispielsweise auf, falls eine Reduktion zu einem großen Residuum führt, dieses aber nur einen großen Fehler in einer internen Variablen zur Folge hat, während der Fehler in den Ausgangsvariablen klein ist oder andersherum. Der große Vorteil des Residuenrankings liegt in dessen Einfachheit sowie der Berechnungsgeschwindigkeit.

3.3.2 Einschrittranking

Im Gegensatz zum Residuenranking ist das Einschrittranking ein absolutes Ranking-Verfahren. Der Wert $\mathcal{R}_{\text{Schritt}}(\mathbf{F}, \kappa)$ ist also eine Schätzung für den Wert des absoluten Fehlers zwischen der Referenzlösung $\mathbf{x}^\star$ und der Lösung von Gleichung (3.3).

Die Idee hinter dem Einschrittranking ist, das perfekte Ranking zu approximieren. Die reduzierte DAE soll also mit einem Verfahren gelöst werden, das nicht besonders genau, dafür aber besonders schnell ist. Hierfür kann das Verfahren auf die Referenzlösung $\mathbf{x}^\star$ zurückgreifen.

Bei der Berechnung der Lösung $\mathbf{x}_n$ einer DAE zu einem Zeitpunkt t_n wird ein nichtlineares Gleichungssystem gelöst (siehe Abschnitt 2.2.3), im Fall von BDF-Verfahren beispielsweise Gleichung (2.185). Hierzu wird in der Regel das Newton-Verfahren verwendet. Als Startwert dient entweder die Lösung $\mathbf{x}_{n-1}$ am vorherigen Zeitpunkt t_{n-1} oder eine Extrapolation der vorherigen Lösungen.

Um eine Abschätzung für den Fehler der Lösung des reduzierten Systems in Gleichung (3.3) zu erhalten, wird im Einschrittranking die Referenzlösung $\mathbf{x}^\star$ als Startwert für das Newton-Verfahren verwendet. Außerdem wird nur eine Iteration durchgeführt. Dies liefert eine Schätzung $\widehat{\mathbf{x}}$ der Lösung von Gleichung (3.3). Folglich ist

$$\mathcal{R}_{\text{Schritt}}(\mathbf{F}, \kappa) = \|\mathbf{x}^\star_{\text{out}} - \widehat{\mathbf{x}}_{\text{out}}\| \tag{3.25}$$

eine Schätzung für den absoluten Fehler.

Das Einschrittranking liefert relativ genaue Schätzungen für den absoluten Fehler. Nachteilig ist jedoch der hohe Berechnungsaufwand.

3.3.3 Sensitivitätsranking

Das Sensitivitätsranking ist ein im Rahmen dieser Arbeit neu entwickeltes Ranking-Verfahren, das auf der adjungierten Sensitivitätsanalyse basiert. Mit Hilfe virtueller Parameter wird dabei basierend auf einer Sensitivitätsanalyse eine Rankingliste erstellt. Im Folgenden wird dieses neue Verfahren präsentiert.

Es ist möglich, während der Berechnung der Lösung einer DAE gleichzeitig eine Sensitivitätsanalyse durchzuführen [Maly & Petzold, 1996; Li & Petzold, 2000]. Eine Sensitivitätsanalyse berechnet den Einfluss von N_p Parametern p_i, $(1 \leq i \leq N_p)$ auf die Lösung einer DAE oder genauer die Sensitivitäten

$$\mathbf{s}_i = \frac{\partial \mathbf{x}}{\partial p_i}, \tag{3.26}$$

mit

$$\mathbf{F}(\mathbf{x}, \dot{\mathbf{x}}, t) = \mathbf{0}. \tag{3.27}$$

Um mit Hilfe einer Sensitivitätsanalyse den Einfluss von Reduktionen auf die Lösung von DAEs zu schätzen, werden virtuelle Parameter p_i^v eingeführt. Sei die Anzahl aller Terme und Subterme N_T. Dann wird jeder Term $\tau \in \mathcal{T}$ durch das Produkt aus dem Term selbst und einem eineindeutig zugehörigen virtuellen Parameter p_i^v ersetzt. Für die virtuellen Parameter gilt

$$p_i^v = 1,\ 1 \leq i \leq N_T. \tag{3.28}$$

Somit haben die virtuellen Parameter keinen Einfluss auf die Lösung. Werden nun die Sensitivitäten bezüglich der virtuellen Parameter berechnet, so erhält man ein Maß für Sensitivität der Lösung bezüglich der Terme. In anderen Worten geben die

Sensitivitäten bezüglich der virtuellen Parameter an, wie sehr sich die Lösung ändert, wenn die Terme manipuliert werden. Die so berechneten Sensitivitäten können also als Rankingwerte verwendet werden. Ein Problem stellt jedoch zunächst der Rechenaufwand dar. Soll die Sensitivitätsanalyse bezüglich N_T virtueller Parameter durchgeführt werden, so erhöht sich die Anzahl der Gleichungen um $N_{sens} = N_T \cdot N$. In diesem Fall wird nämlich die DAE um die Gleichungen

$$\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\mathbf{s}_i + \frac{\partial \mathbf{F}}{\partial \dot{\mathbf{x}}}\dot{\mathbf{s}}_i + \frac{\partial \mathbf{F}}{\partial p_i^v} = \mathbf{0}, \quad 1 \leq i \leq N_T \tag{3.29}$$

erweitert, welche durch Differentiation der DAE nach den virtuellen Parametern entstehen. Da die Anzahl der Terme N_T in der Regel sehr groß ist, lässt sich das Sensitivitätsranking so nicht anwenden.

Einen Ausweg bietet die so genannte adjungierte Sensitivitätsanalyse [Cao *et al.*, 2002]. Bei der adjungierten Sensitivitätsanalyse werden die Sensitivitäten aus Gleichung (3.26) nicht direkt berechnet. Anstatt dessen wird eine Gütefunktion $g(\mathbf{x})$ gewählt. Hieraus wird das Gütefunktional

$$G(\mathbf{x}, \mathbf{p}^v) = \int_0^T g(\mathbf{x}, \mathbf{p}^v, t)dt \tag{3.30}$$

gebildet. Nun ist es möglich die Ableitungen $\frac{\partial G}{\partial p_i^v}$ des Gütefunktionals bezüglich der N_T virtuellen Parameter p_i^v mit relativ wenig Rechenaufwand zu berechnen. Hierzu wird zunächst der Lagrange-Multiplikator $\boldsymbol{\lambda}$ eingeführt und das Funktional

$$I(\mathbf{x}, \mathbf{p}^v) = G(\mathbf{x}, \mathbf{p}^v) - \int_0^T \boldsymbol{\lambda}^T \mathbf{F}(\mathbf{x}, \dot{\mathbf{x}}, t)dt \tag{3.31}$$

betrachtet [Cao *et al.*, 2003]. Wegen Gleichung (3.1) gilt dann

$$\frac{\partial G}{\partial \mathbf{p}^v} = \frac{\partial I}{\partial \mathbf{p}^v} = \int_0^T (\frac{\partial g}{\partial \mathbf{p}^v} + \frac{\partial g}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial \mathbf{p}^v})dt - \int_0^T \boldsymbol{\lambda}^T (\frac{\partial \mathbf{F}}{\partial \mathbf{p}^v} + \frac{\partial \mathbf{F}}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial \mathbf{p}^v} + \frac{\partial \mathbf{F}}{\partial \dot{\mathbf{x}}}\frac{\partial \dot{\mathbf{x}}}{\partial \mathbf{p}^v})dt. \tag{3.32}$$

Mit Hilfe der partiellen Integration ergibt sich der Zusammenhang

$$\int_0^T \boldsymbol{\lambda}^T \frac{\partial \mathbf{F}}{\partial \dot{\mathbf{x}}}\frac{\partial \dot{\mathbf{x}}}{\partial \mathbf{p}^v}dt = (\boldsymbol{\lambda}^T \frac{\partial \mathbf{F}}{\partial \dot{\mathbf{x}}}\frac{\partial \mathbf{x}}{\partial \mathbf{p}^v}) \,|_0^T - \int_0^T \frac{d}{dt}(\boldsymbol{\lambda}^T \frac{\partial \mathbf{F}}{\partial \dot{\mathbf{x}}})\frac{\partial \mathbf{x}}{\partial \mathbf{p}^v}dt. \tag{3.33}$$

Setzt man Gleichung (3.33) in Gleichung (3.32) ein, so erhält man

$$\begin{aligned}\frac{\partial G}{\partial \mathbf{p}^v} = &\int_0^T (\frac{\partial g}{\partial \mathbf{p}^v} - \boldsymbol{\lambda}^T \frac{\partial \mathbf{F}}{\partial \mathbf{p}^v})dt - \int_0^T (-\frac{\partial g}{\partial \mathbf{x}} + \boldsymbol{\lambda}^T \frac{\partial \mathbf{F}}{\partial \mathbf{x}} - \frac{d}{dt}(\boldsymbol{\lambda}^T \frac{\partial \mathbf{F}}{\partial \dot{\mathbf{x}}}))\frac{\partial \mathbf{x}}{\partial \mathbf{p}^v}dt \\ &- (\boldsymbol{\lambda}^T \frac{\partial \mathbf{F}}{\partial \dot{\mathbf{x}}}\frac{\partial \mathbf{x}}{\partial \mathbf{p}^v}) \,|_0^T \,.\end{aligned} \tag{3.34}$$

Folglich gilt

$$\frac{\partial G}{\partial \mathbf{p}^v} = \int_0^T (\frac{\partial g}{\partial \mathbf{p}^v} - \boldsymbol{\lambda}^T \frac{\partial \mathbf{F}}{\partial \mathbf{p}^v}) dt - (\boldsymbol{\lambda}^T \frac{\partial \mathbf{F}}{\dot{\mathbf{x}}} \frac{\partial \mathbf{x}}{\partial p}) \mid_{t=0}, \tag{3.35}$$

falls

$$\boldsymbol{\lambda}^T \frac{\partial \mathbf{F}}{\partial \mathbf{x}} - \frac{d}{dt}(\boldsymbol{\lambda}^T \frac{\partial \mathbf{F}}{\partial \dot{\mathbf{x}}}) = \frac{\partial g}{\partial \mathbf{x}}. \tag{3.36}$$

Gleichung (3.36) ist eine lineare DAE und wird die *adjungierte* DAE genannt. Die adjungierte DAE wird rückwärts in der Zeit gelöst.

Zur Berechnung der Sensitivitäten ist also die Lösung der adjungierten DAE sowie die Berechnung des Integrals aus Gleichung (3.35) notwendig. Die größte Schwierigkeit beim Lösen der adjungierten DAE ist im Allgemeinen das Finden von konsistenten Anfangswerten (zum Zeitpunkt T). Im Falle von DAEs mit differentiellem Index $\nu_d = 0$ oder $\nu_d = 1$, ist dies jedoch durch Lösen des Minimierungsproblems

$$\min_{\lambda} \left\| \begin{bmatrix} \frac{\partial \mathbf{F}}{\partial \dot{\mathbf{x}}} \mid_{t=T} \dot{\lambda} + \frac{\partial \mathbf{F}}{\partial \mathbf{x}} \mid_{t=T} \lambda - \frac{\partial g}{\partial \mathbf{x}}^T \\ \\ \frac{\partial \mathbf{F}}{\partial \dot{\mathbf{x}}} \mid_{t=T} \lambda \end{bmatrix} \right\|_2 \tag{3.37}$$

möglich. Da DAEs in der Regel mindestens auf den Index $\nu_d = 1$ reduziert werden (siehe Abschnitt 2.2.2), stellt die Initialisierung der adjungierten DAE hier kein Problem dar.

Der Rechenaufwand zum Lösen der adjungierten DAE ist genauso hoch wie der Aufwand zum Lösen der zugrunde liegenden DAE. Das Integral aus Gleichung (3.35) kann sehr effizient durch eine Quadraturformel gelöst werden. Somit ist der gesamte Rechenaufwand für das Sensitivitätsranking in etwa so hoch wie für die Referenzsimulation. Dies ist im Vergleich zum Einschrittranking im Allgemeinen deutlich weniger.

Die Gütefunktion ist frei wählbar. Eine sinnvolle Funktion im Rahmen eines Ranking-Verfahrens ist zum Beispiel

$$g(\mathbf{x}) = \sum_{i=1}^{N_{\text{out}}} x_{\text{out},i}^2, \tag{3.38}$$

also die Summe der Quadrate der Ausgangsvariablen.

Das Sensitivitätsranking ist ein relatives Ranking-Verfahren. Trotz seiner vergleichsweise kurzen Berechnungszeit weist es eine hohe Genauigkeit auf. Im kommenden Abschnitt wird die Effizienz beziehungsweise Güte der hier vorgestellten Ranking-Verfahren miteinander verglichen.

3.3.4 Güte der Ranking-Verfahren

Die in den vorherigen Abschnitten vorgestellten Ranking-Verfahren werden in diesem Abschnitt gegenüber gestellt. Hierzu werden diese Ranking-Verfahren anhand

eines Beispiels mit dem perfekten Ranking verglichen. Als Beispielsystem fungiert hier der in Abschnitt 2.1 eingeführte Zweimassenschwinger. Mit Hilfe der Methoden aus Abschnitt 2.1.2 kann das mathematische Modell (Gleichungen (2.13)-(2.36)) entsprechend Gleichung (2.43) geschrieben werden als

$$x_{S_1,\mathrm{rel}} = x_{M_1} - x_L, \tag{3.39}$$
$$x_{S_2,\mathrm{rel}} = x_{M_2} - x_{M_1}, \tag{3.40}$$
$$v_{S_1,\mathrm{rel}} = v_{M_1}, \tag{3.41}$$
$$v_{S_2,\mathrm{rel}} = v_{M_2} - v_{M_1}, \tag{3.42}$$
$$F_{S_1} = c_{S_1} \cdot x_{S_1,\mathrm{rel}} + d_{S_1} \cdot v_{S_1,\mathrm{rel}}, \tag{3.43}$$
$$F_{S_2} = c_{S_2} \cdot x_{S_2,\mathrm{rel}} + d_{S_2} \cdot v_{S_2,\mathrm{rel}}, \tag{3.44}$$
$$\dot{x}_{M_1} = v_{M_1}, \tag{3.45}$$
$$\dot{v}_{M_1} = \frac{1}{m_{M_1}}(F_{S_2} - F_{S_1}), \tag{3.46}$$
$$\dot{x}_{M_2} = v_{M_2}, \tag{3.47}$$
$$\dot{v}_{M_2} = \frac{1}{m_{M_2}}(F_{\mathrm{ext}} - F_{S_2}). \tag{3.48}$$

Zur besseren Übersicht wurden hier die Indizes a und b fallen gelassen. Darüber hinaus wird der Zweimassenschwinger nun durch eine externe Kraft F_{ext}, die an der zweiten Masse angreift, angeregt.

Für alle Ranking-Verfahren werden die Parameter

$$m_{M_1} = 1\mathrm{kg}, \quad m_{M_2} = 100\mathrm{kg}, \quad c_{S_1} = 1000\frac{\mathrm{N}}{\mathrm{m}}, \quad c_{S_2} = 100\frac{\mathrm{N}}{\mathrm{m}}, \tag{3.49}$$
$$d_{S_1} = 1\frac{\mathrm{Ns}}{\mathrm{m}}, \quad d_{S_2} = 1\frac{\mathrm{Ns}}{\mathrm{m}} \quad \text{und} \quad x_L = 0\mathrm{m} \tag{3.50}$$

verwendet. Die externe Kraft hat die Form

$$F_{\mathrm{ext}}(t) = 1500\sin(\frac{t}{6}) + 50\sin(15t), \tag{3.51}$$

und die Simulationszeit beträgt $T = 60\mathrm{s}$. Verwendet man nun noch die Ruhelage als Startzustand, so ist das verwendete Szenario festgelegt. Die Ausgangsvariable ist hier die Position der Masse M_2. In Abbildung 3.2 ist die Referenzlösung dargestellt.

Bei diesem System machen die beiden Reduktionstechniken „Linearisierung“ und „Ähnlichkeitsbetrachtungen“ wenig Sinn. Für die bessere Übersicht wurde zusätzlich auf die „Termsubstitution“ verzichtet. Folglich ist die Termreduktion die einzige verwendete Reduktionstechnik.

Die Durchführung der drei Ranking-Verfahren liefert drei Listen, die die Rankingwerte aller Reduktionen für die drei Verfahren beinhalten. Diese Listen werden aufsteigend nach den Rankingwerten sortiert und im folgenden mit $\mathcal{L}_{\mathrm{Res}}$ (für das Residuenranking), $\mathcal{L}_{\mathrm{Schritt}}$ (für das Einschrittranking) und $\mathcal{L}_{\mathrm{Sens}}$ (für das Sensitivitätsranking) bezeichnet. Um die Güte der drei Ranking-Verfahren zu beurteilen, werden diese drei *Rankinglisten* nun mit der Liste $\mathcal{L}_{\mathrm{Perf}}$ verglichen, die das perfekte Ranking liefert.

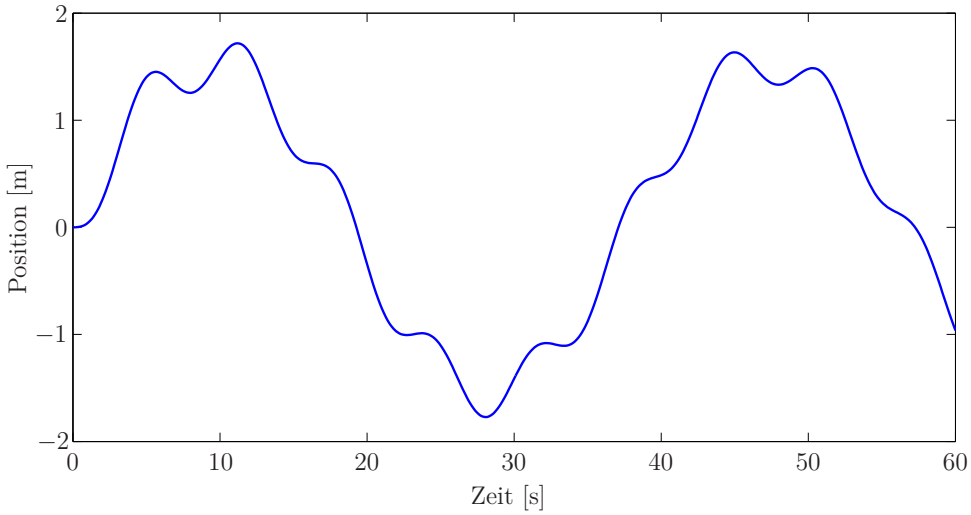


Abbildung 3.2: Position der Masse M_2

Um die Güte des Residuenrankings zu bewerten, wird für jede Reduktion $\kappa \in \mathcal{K}$ die Differenz d^κ zwischen den entsprechenden Stellen in $\mathcal{L}_{\text{Perf}}$ und $\mathcal{L}_{\text{Res}}$ gebildet. Der Wert

$$G_{\text{Res}} = \frac{\sum\limits_{\kappa \in \mathcal{K}} d^\kappa}{|\mathcal{K}|} \tag{3.52}$$

gibt dann an, wie weit die Stelle, an der eine Reduktion κ in der Liste $\mathcal{L}_{\text{Res}}$ steht, durchschnittlich von der *richtigen* Stelle entfernt ist. Die *richtige* Stelle ist dabei die Stelle, an der die Reduktion κ in $\mathcal{L}_{\text{Perf}}$ steht. Analog werden die Werte G_{Schritt} und G_{Sens} berechnet. Mit

$$|\mathcal{K}| = 23 \tag{3.53}$$

ergibt sich

$$G_{\text{Res}} = 3,8, \quad G_{\text{Schritt}} = 3,4 \quad \text{und} \quad G_{\text{Sens}} = 3,27. \tag{3.54}$$

Das genaueste Verfahren ist hier also das Sensitivitätsranking. Ähnlich genau ist das Einschrittranking. Jedoch benötigt dieses 12, 24s für die Berechnung der Rankingliste. Das ist deutlich länger als die 1, 501s, die das Sensitivitätsranking für dieselbe Aufgabe benötigt. Noch schneller ist das Residuenranking mit einer Laufzeit von nur 0, 857s. Jedoch ist das Residuenranking auch das ungenaueste Ranking-Verfahren. Die Berechnungen wurden mit Hilfe der in Abschnitt 4.1 beschriebenen Implementierung auf einem 1,8 GHz Laptop durchgeführt.

Der größte Fehler unterläuft dem Residuenranking bei der Reduktion κ, die den Term $(F^{S_2} - F^{S_1})$ in Gleichung (3.46) streicht. In der Liste $\mathcal{L}_{\text{Res}}$ steht diese Reduktion an der zweiten Stelle, während die Reduktion in $\mathcal{L}_{\text{Perf}}$ erst an der elften Stelle auftaucht. Der Grund hierfür ist, dass das Residuum der Gleichung (3.48) nach Durchführung von κ relativ klein ist, da der Unterschied zwischen den Kräften F_{S_1} und F_{S_2} nicht sehr groß ist. Der Einfluss des Terms ist jedoch sehr groß. Wird der Term $(F_{S_2} - F_{S_1})$

nämlich vernachlässigt, so ist die Masse M_1 nicht mehr schwingungsfähig. Bei dem Einschritt- sowie Sensitivitätsranking sind die Abstände zwischen den berechneten Stellen und den Stellen im perfekten Ranking sehr homogen, also nahe am Durchschnittswert.

Diese Untersuchung besitzt natürlich keine Allgemeingültigkeit. Je nach System können die Ergebnisse variieren. Jedoch zeigten sich für alle im Rahmen dieser Arbeit untersuchten Systeme ähnliche Ergebnisse. So ist die Genauigkeit von Einschrittranking und Sensitivitätsranking meist ähnlich, während das Residuenranking deutlich ungenauer ist. Das Einschrittranking benötigt dafür in der Regel viel mehr Zeit als die beiden anderen Ranking-Verfahren.

Die Geschwindigkeit des Residuen- und Einschrittrankings kann erhöht werden, indem die Lösung nicht an allen berechneten Zeitpunkten zur Bestimmung des Rankingwerts verwendet wird. Dieses Vorgehen steigert zwar die Geschwindigkeit, dafür muss mit Einbußen bei der Genauigkeit gerechnet werden. Zusammenfassend kann gesagt werden, dass das Sensitivitätsranking in den meisten Fällen den besten Kompromiss darstellt.

3.4 Reduktion für eine vorgegebene Fehlertoleranz

Im vorherigen Abschnitt wurden verschiedene Ranking-Verfahren vorgestellt und miteinander verglichen. Im nächsten Abschnitt wird ein Algorithmus vorgestellt, der basierend auf der Rankingliste möglichst viele Reduktionen $\kappa \in \mathcal{K}$ durchführt. Im Rahmen dieser Arbeit wurde der Algorithmus, um zwei neue Funktionalitäten erweitert. So ist es nun möglich mehrere oder sogar intervallwertige Szenarien vorzugeben. Diese Erweiterungen werden in Abschnitt 3.4.2 und 3.4.3 präsentiert.

3.4.1 Approximationsalgorithmus

Das Ziel des Approximationsalgorithmus ist es, die Komplexität der DAE unter der Randbedingung einer vorgegebenen Fehlertoleranz $\boldsymbol{\epsilon}_{\text{Fehler}}$ so weit wie möglich zu senken. Es sollen also so viele Reduktionen wie möglich durchgeführt werden. Um dies zu bewerkstelligen, ist es sinnvoll mit denjenigen Reduktionen zu beginnen, die zu einem mutmaßlich kleinen Fehler führen. Folglich legen die Rankingwerte die Reihenfolge fest, in der die Reduktionen durchgeführt werden.

Die einfachste Möglichkeit eines Approximationsalgorithmus beginnt also mit der Durchführung der ersten Reduktion κ aus $\mathcal{L}$ und prüft danach, ob die Fehlertoleranz $\boldsymbol{\epsilon}_{\text{Fehler}}$ eingehalten wird. Die Fehlertoleranz $\boldsymbol{\epsilon}_{\text{Fehler}}$ wird dabei nur für die N_{out} Ausgangsvariablen vorgegeben. Es wird also die Bedingung

$$\left\| x^{\star}_{\text{out},i} - \mathcal{N}(\mathbf{F}^{\kappa}(\mathbf{x}, \dot{\mathbf{x}}, t), \mathcal{S})_i \right\| \leq \epsilon_{\text{Fehler},i} \quad \text{für} \quad 1 \leq i \leq N_{\text{out}} \tag{3.55}$$

geprüft, wobei $\mathbf{x}^{\star}_{\text{out}}$ wieder die Referenzlösung bezeichnet. Die Norm ist an dieser Stelle wieder beliebig. In dieser Arbeit werden zwei verschiedene Normen zum Einsatz kommen. Diese beiden Normen sind zum einen die Maximumsnorm und zum anderen die euklidische Norm. Bei der Maximumsnorm entspricht $\boldsymbol{\epsilon}_{\text{Fehler}}$ dann der Vorgabe eines maximalen Fehlers, der zu keinem Zeitpunkt überschritten werden darf. Zur Überprüfung dieser Bedingung ist offensichtlich eine Simulation und somit die Vorgabe eines Szenarios notwendig. Somit wird an dieser Stelle deutlich, dass die Einhaltung der vorgegebenen Fehlertoleranz nur für das vorgegebene Szenario garantiert ist. Damit kommt der Wahl des Szenarios eine besondere Bedeutung zu. Hierauf wird in Abschnitt 3.6 nochmals gesondert eingegangen. Ist die Bedingung aus Gleichung (3.55) nicht erfüllt, so wird die Reduktion κ zurückgenommen und der Approximationsalgorithmus fährt mit der zweiten Reduktion aus $\mathcal{L}$ fort. Ist die Bedingung hingegen erfüllt, so fährt der Approximationsalgorithmus sofort mit der zweiten Reduktion aus $\mathcal{L}$ fort. Auf diese Art und Weise werden die Reduktionen aus $\mathcal{L}$ abgearbeitet bis ein Abbruchkriterium erfüllt ist.

An dieser Stelle stellt sich die Frage nach einem sinnvollen Abbruchkriterium. In dieser Arbeit werden zwei verschiedene Abbruchkriterien verwendet. Für das erste Kriterium wird ein Maximalwert $r_{\max}$ für die Rankingwerte eingeführt. Es werden dann nur diejenigen Reduktionen κ durchgeführt, welche einen Rankingwert $r_\kappa < r_{max}$ aufweisen. Dieses Abbruchkriterium hat allerdings einen Nachteil. Gerade bei relativen Ranking-Verfahren ist es schwierig, a priori einen geeigneten Wert für r_{max} zu finden. Für das zweite Abbruchkriterium wird eine Obergrenze n_{fehl} für die Anzahl fehlgeschlagener Reduktionen festgelegt. Während des Approximationsalgorithmus werden in diesem Fall die Anzahl derjenigen Reduktionen gezählt, die zu einer Überschreitung der Fehlergrenze $\boldsymbol{\epsilon}_{\text{Fehler}}$ führen, also die Bedingung aus Gleichung (3.55) verletzen. Ist diese Anzahl größer als n_{fehl}, so bricht der Approximationsalgorithmus ab. Die Motivation hierzu ist darin begründet, dass die Wahrscheinlichkeit für eine erfolgreiche Durchführung einer Reduktion mit steigendem Rankingwert natürlich abnimmt.

Man beachte, dass der berechnete Fehler (linke Seite von Gleichung (3.55)) auch von der Wahl des numerischen Integrationsverfahrens abhängig ist. Hier hat der Benutzer dafür Sorge zu tragen, dass die DAEs hinreichend genau gelöst werden. Weiterhin können die internen Variablen $\mathbf{x}_{\text{int}}$ beliebig weit von der Referenzlösung $\mathbf{x}^{\star}_{\text{int}}$ abweichen. Die Anzahl und Wahl der Ausgangsvariablen ist jedoch dem Benutzer überlassen. Offensichtlich kann es bei diesem Algorithmus aufgrund der zahlreichen Simulationen zu langen Laufzeiten kommen. Daher wird in [Wichmann, 2004] die Beschleunigung dieses Algorithmus durch die Verwendung von Clustern vorgeschlagen.

Cluster

Um die Anzahl der notwendigen Simulationen zu senken, kann die Verwendung von Clustern sinnvoll sein. Hierzu wird die Rankingliste $\mathcal{L}$ in n_c disjunkte Cluster C_i

$$\mathcal{L} = \bigcup_{i=1}^{n_c} C_i, \tag{3.56}$$

mit

$$\mathcal{C} = [C_1, \ldots, C_{n_c}] \tag{3.57}$$

aufgeteilt. Jedes Cluster C_i enthält Reduktionen, die zu einem Fehler führen, der mutmaßlich in derselben Größenordnung liegt. Solche Cluster zu identifizieren gelingt anhand der Rankingwerte der Reduktionen. Die Cluster C_i werden also gebildet aus Reduktionen, welche ähnliche Rankingwerte aufweisen. In einem Cluster finden sich dann Reduktionen mit Rankingwerten, die sich um nicht mehr als einen Faktor r_{Fak} unterscheiden. Zusätzlich ist es sinnvoll, alle Reduktionen κ, die einen Rankingwert r_κ unter einem Schwellenwert $r_{\min}$ aufweisen, in einem Cluster C_1 zusammenzufassen.

Bei der Durchführung des Algorithmus werden dann anstatt einer einzelnen Reduktion κ jeweils alle Reduktionen aus einem Cluster C_i durchgeführt und anschließend die Bedingung aus Gleichung (3.55) geprüft. Ist diese Bedingung erfüllt, so fährt der Algorithmus mit dem nächsten Cluster fort. Ist die Bedingung hingegen verletzt, so werden die Reduktionen aus C_i zunächst rückgängig gemacht. Dann wird das Cluster C_i in zwei neue disjunkte Cluster C_i^1 und C_i^2 aufgeteilt. Der Approximationsalgorithmus fährt dann mit C_i^1 und anschließend mit C_i^2 fort.

Wird ein relatives Ranking verwendet, so ist es schwierig, einen geeigneten Wert für r_{Fak} zu finden. Dementsprechend wird in [Wichmann, 2004] vorgeschlagen, in diesem Fall auf die Verwendung von Clustern zu verzichten. Es hat sich jedoch als sehr effizient herausgestellt, auch bei relativen Ranking-Verfahren zumindest das Cluster C_1 für Reduktionen mit sehr kleinen Rankingwerten zu bilden. Auf diese Art und Weise können alle Reduktionen, welche nahezu keinen Einfluss haben, auf einmal durchgeführt werden. In Algorithmus 1 ist der Approximationsalgorithmus für eine Reduktionstechnik oder mehrere Reduktionstechniken, ein Ranking-Verfahren $\mathcal{R}$ und ein numerisches Lösungsverfahren $\mathcal{N}$ mit der Verwendung von Clustern in Pseudo-Code dargestellt. Für eine Reduktion $\kappa \in \mathcal{K}$ macht κ^{-1} die Reduktion rückgängig. Im nachfolgenden Beispiel wird dieser Algorithmus auf den Zweimassenschwinger aus Abschnitt 2.1 beziehungsweise 3.3.4 angewendet.

Beispiel 3.4.1. Das mathematische Modell des Zweimassenschwingers, beschrieben durch die Gleichungen (3.39) bis (3.48), wird in diesem Beispiel für das in Abschnitt 3.3.4 angegebene Szenario reduziert. Das verwendete Ranking ist hier das Sensitivitätsranking und Termreduktion die einzige verwendete Reduktionstechnik. Als Ausgangsvariable wird wie auch in Abschnitt 3.3.4 die Position der Masse M_2 gewählt. Der maximale erlaubte Fehler wird auf $\epsilon_{\text{Fehler}} = 0,045\text{m}$ gesetzt, was in etwa 2% der maximalen Auslenkung entspricht. Das reduzierte Modell lautet dann

$$x_{M_1} = \frac{c_{S_2} \cdot x_{M_2}}{(c_{S_1} + c_{S_2})}, \tag{3.58}$$

$$x_{S_2,\text{rel}} = x_{M_2} - x_{M_1}, \tag{3.59}$$

$$F_{S_2} = c_{S_2} \cdot x_{S_2,\text{rel}}, \tag{3.60}$$

$$\dot{x}_{M_2} = v_{M_2}, \tag{3.61}$$

$$\dot{v}_{M_2} = \frac{1}{m_{M_2}}(F_{\text{ext}} - F_{S_2}). \tag{3.62}$$

Algorithmus 1 Approximationsalgorithmus für eine vorgegebene Fehlergrenze

Eingang: DAE ($\mathbf{F}$), Fehlergrenze ($\boldsymbol{\epsilon}_{\text{Fehler}}$), Scenario ($\mathcal{S}$)
Ausgang: reduzierte DAE ($\mathbf{G}$)

```
[x*_out  x*_int]^T = N(F(x, ẋ, t), S)
G = F
L = R(F, K)
L = sort(L)
Berechne C aus L
while C ≠ ∅ do
  C = C \ C_1
  for all κ ∈ C_1 do
    G = G^κ
  end for
  [x_out  x_int]^T = N(G(x, ẋ, t), S)
  ε_i = ||x*_out,i − x_out,i||, 1 ≤ i ≤ N_out
  if ∃ i : ε_i ≥ ε_Fehler,i then
    for all κ ∈ S_1 do
      G = G^(κ^-1)
    end for
    Teile C_1 in C_1^1 und C_1^2
    C = [C_1^1, C_1^2, S]
  end if
end while
```

Gleichung (3.58) ist dabei aus Gleichung (3.39) hervorgegangen, indem der Term $\dot{v}_{M_1}$ zu Null gesetzt, die Dämpfung vernachlässigt und x_{M_1} isoliert wurde.

Das Streichen des Terms $\dot{v}_{M_1}$ kann als Betrachtung des Grenzfalls $m_{M_1} = 0$ angesehen werden. Zudem gibt es keine Dämpfungseinflüsse mehr in dem Modell. Der tatsächlich auftretende Fehler der Position von M_1 beträgt $0,0367$m. In Abbildung 3.3 ist die Lösung des ursprünglichen Modells und die Lösung des reduzierten Modells gezeigt. Die beiden Kurven liegen offensichtlich beinahe übereinander. Das Bode-Diagramm zeigt die Unterschiede zwischen ursprünglichem und reduziertem Modell besser. In Abbildung 3.4 ist das Bode-Diagramm für die Übertragungsfunktion

$$G(s) = \frac{x_{M_2}}{F_{\text{ext}}} \tag{3.63}$$

gezeigt. Es ist zu sehen, dass die Resonanzüberhöhung bei $0,15$Hz durch die Reduktion verstärkt wird. Grund hierfür ist die vernachlässigte Dämpfung, welche auch für den glatteren Übergang des Phasengangs von $0°$ auf $-180°$ bei dem ursprünglichen Modell verantwortlich ist. Die Amplitude des reduzierten Modells fällt danach mit $44,53$dB pro Dekade genau wie das ursprüngliche Modell ab. Bei $5,27$Hz ist bei dem ursprünglichen Modell jedoch zusätzlich die Resonanzüberhöhung der Masse M_1 zu sehen, die das reduzierte Modell natürlich nicht zeigt. Das gilt auch für den mit der Resonanzüberhöhung einhergehenden Abfall im Phasengang. Nach der zweiten Resonanzüberhöhung fällt die Amplitude des ursprünglichen Modells mit $67,755$dB pro Dekade. Der leichte Anstieg im Phasengang ist der Dämpfung zuzuschreiben.

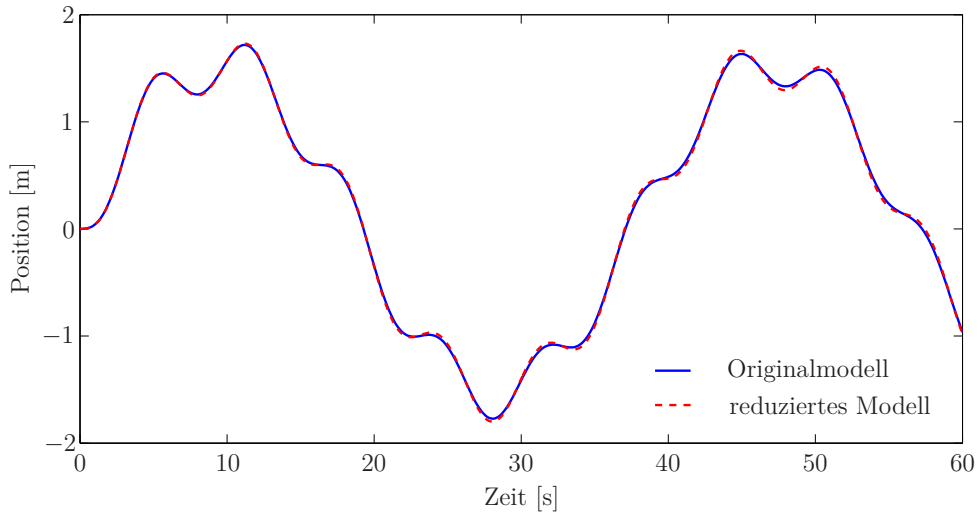


Abbildung 3.3: Position der Masse M_2 (mit externer Anregung F_{ext})

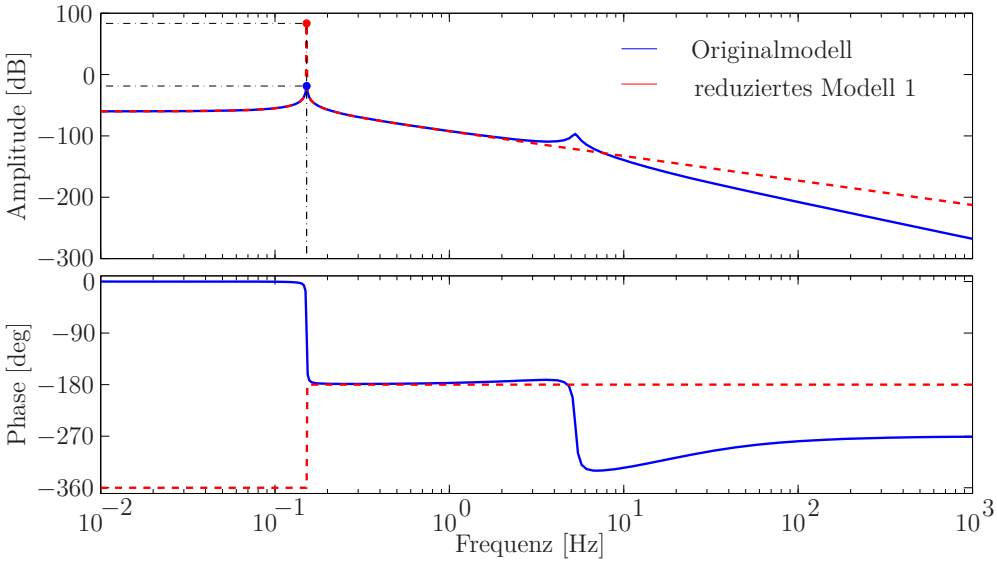


Abbildung 3.4: Bode-Diagramm der Übertragungsfunktion G

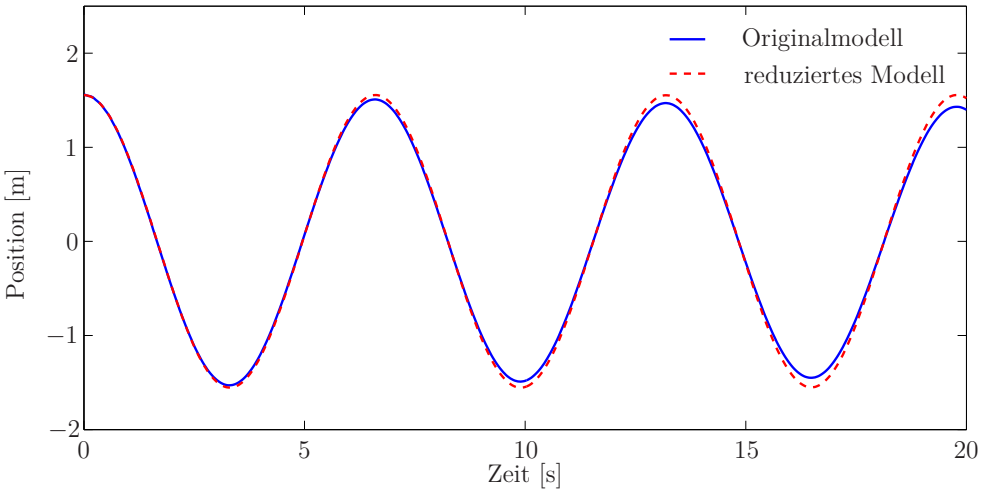


Abbildung 3.5: Position der Masse M_2 (mit den Startwerten aus Gleichung (3.64))

Im Bode-Diagramm ist gut zu erkennen, dass es sowohl Bereiche gibt, in denen die Plots eine gute Übereinstimmung aufweisen, als auch Bereiche, in denen sich das Verhalten des reduzierten Modells deutlich von dem Verhalten des Originalmodells unterscheidet. Ein Beispiel hierfür ist der signifikante Unterschied in der Resonanzüberhöhung. Im folgenden Abschnitt werden zwei Ansätze beschrieben, um die Bereiche, in denen das Verhalten des reduzierten Modells und das Verhalten des Originalmodells gut übereinstimmt, zu vergrößern.

3.4.2 Erweiterung für multiple Szenarien

Die Verwendung von Szenarien für die Reduktion hat den Vorteil, dass Modelle für für bestimmte Zwecke reduziert werden können. Die Reduktion unter Vorgabe eines bestimmten Szenarios liefert sehr effiziente Modelle. Auf der anderen Seite muss für die Reduktion ein Szenario gefunden werden, das dafür sorgt, dass alle physikalischen Effekte von Interesse in dem reduzierten Modell erhalten bleiben. Zudem ist es oftmals wünschenswert, dass das reduzierte Modell für mehrere Parametersätze oder Systemeingänge verwendet werden kann und der Fehler weiterhin innerhalb einer gewissen Fehlertoleranz verbleibt. Im Rahmen dieser Arbeit wurde dieser Nachteil durch die Erweiterung für multiple Szenarien aufgehoben. Die Reduktion aus dem Beispiel in dem vorherigen Abschnitt liefert ein Modell, das sich in dem für die Reduktion verwendeten Szenario mit dem ursprünglichen Modell nahezu identisch ist. Betrachtet man die beiden Modelle jedoch für ein anderes Szenario, so kann sich ein völlig anderes Bild ergeben. In Abbildung 3.5 ist die Position der Masse M_2 für ein anderes, als das in der Reduktion verwendete Szenario gezeigt. Diesmal wird der Zweimassenschwinger nicht durch eine externe Kraft angeregt. Satt dessen werden die Startwerte

$$x_{M_1} = 1,56\text{m}, \quad \dot{x}_{M_1} = 0,01\frac{\text{m}}{\text{s}}, \quad x_{M_2} = 17,1\text{m} \quad \text{und} \quad \dot{x}_{M_2} = 0,19\frac{\text{m}}{\text{s}} \tag{3.64}$$

verwendet und die Simulationszeit auf 20s verkürzt. Offensichtlich ist der auftretende Fehler in diesem Szenario größer. Bereits nach zehn Sekunden ist die Fehlertoleranz, die für das in der Reduktion verwendete Szenario vorgegeben war, überschritten. Der maximal auftretende Fehler beträgt $0,12$m. Der Grund hierfür liegt in der Vernachlässigung der Dämpfung. Damit wird das Abklingen der Schwingung nicht mehr wiedergegeben. Somit würde bei einer längeren Simulation der Fehler weiter ansteigen.

In bisherigen Arbeiten wurde das oben beschriebene Problem stets außer Acht gelassen. Im Rahmen dieser Arbeit wird der Algorithmus in einem ersten Schritt erweitert, so dass nun auch die Vorgabe mehrerer Szenarien möglich ist. Die Realisierung dieser Möglichkeit ist auf eine sehr natürliche Art und Weise geschehen.

Sind n_{Szen} Szenarien $\mathcal{S}_i$ gegeben, so setzt sich die Referenzlösung $\mathbf{x}^\star$ aus den n_{Szen} Referenzlösungen $\mathbf{x}^{\star,i}$ $(1 \leq i \leq n_{\text{Szen}})$ der einzelnen Szenarien zusammen. Es ist also

$$\begin{aligned}\mathbf{x}^\star &= \left[\mathcal{N}(\mathbf{F}(\mathbf{x}, \dot{\mathbf{x}}, t), \mathcal{S}_1) \quad N(\mathbf{F}(\mathbf{x}, \dot{\mathbf{x}}, t), \mathcal{S}_2) \quad \dots \quad \mathcal{N}(\mathbf{F}(\mathbf{x}, \dot{\mathbf{x}}, t), \mathcal{S}_{n_{\text{Szen}}})\right] \\ &= \left[\mathbf{x}^{\star,1} \quad \mathbf{x}^{\star,2} \quad \dots \quad \mathbf{x}^{\star,n_{\text{Szen}}}\right]. \end{aligned} \tag{3.65}$$

Für jede Reduktion ergibt sich der Rankingwert

$$\kappa = \sum_{i=1}^{n_{\text{Szen}}} \kappa^i \tag{3.66}$$

aus der Summe der Rankingwerte κ^i für die n_{Szen} Szenarien. Schließlich wird die Bedingung, ob eine Reduktion zu einem Fehler innerhalb der Fehlertoleranz führt (Gleichung (3.55)), zu

$$\left\| x^{\star,j}_{\text{out},i} - \mathcal{N}(\mathbf{F}^\kappa(\mathbf{x}, \dot{\mathbf{x}}, t), \mathcal{S}_j)_i \right\| \leq \epsilon^j_{\text{Fehler},i} \tag{3.67}$$

für

$$1 \leq i \leq N_{\text{out}} \text{ und } 1 \leq j \leq n_{\text{Szen}}. \tag{3.68}$$

Dabei wird für jedes Szenario $\mathcal{S}_j$ eine eigene Fehlertoleranz $\boldsymbol{\epsilon}^j$ vorgegeben. Im folgenden Beispiel wird die Reduktion des Zweimassenschwingers für zwei Szenarien durchgeführt.

Beispiel 3.4.2. In Beispiel 3.4.1 wurde die Reduktion für nur ein Szenario durchgeführt. Hier wird das im vorherigen Abschnitt eingeführte Szenario hinzugefügt. Für die externe Kraft gilt also

$$F_{\text{ext}}(t) = 0, \tag{3.69}$$

und die Startwerte aus Gleichung (3.64) werden verwendet. Der verwendete Parametersatz ist derselbe wie im ersten Szenario. Die Fehlertoleranz beträgt $0,05$m für

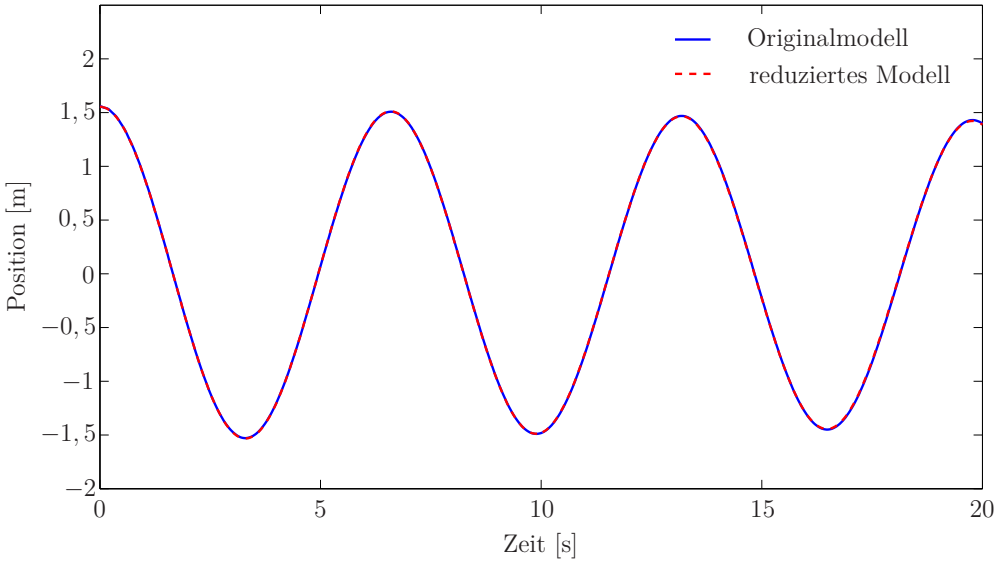


Abbildung 3.6: Position der Masse M_2 (mit den Startwerten aus Gleichung (3.64))

das zweite Szenario. Wieder wird das Sensitivitätsranking verwendet, um die Rankingliste zu erstellen. Das reduzierte Modell lautet dann

$$x_{M_1} = \frac{v_{M_2} \cdot d_{S_2} + x_{M_2} \cdot c_{S_2}}{c_{S_1} + c_{S_2}}, \tag{3.70}$$

$$x_{S_2,\text{rel}} = x_{M_2} - x_{M_1}, \tag{3.71}$$

$$v_{S_2,\text{rel}} = v_{M_2}, \tag{3.72}$$

$$F_{S_2} = c_{S_2} \cdot x_{S_2,\text{rel}} + d_{S_2} \cdot v_{S_2,\text{rel}}, \tag{3.73}$$

$$\dot{x}_{M_2} = v_{M_2}, \tag{3.74}$$

$$\dot{v}_{M_2} = \frac{1}{m_{M_2}}(F_{\text{ext}} - F_{S_2}). \tag{3.75}$$

$$\tag{3.76}$$

Es ist zu sehen, dass die Dämpfung von S_2 erhalten geblieben ist. Jedoch ist die Dämpfung nur noch abhängig von der Geschwindigkeit v_{M_2} der zweiten Masse M_2. Gleichwohl ist der Fehler, welcher bei der Reduktion für ein Szenario durch Vernachlässigung der Dämpfung entstanden war, viel geringer. Dies kann auch in Abbildung 3.6 beobachtet werden. Hier liegen die beiden Kurven nun fast übereinander und der maximale Fehler beträgt nur noch $0,011$m. In Abbildung 3.7 ist das Bode-Diagramm der Übertragungsfunktion aus Gleichung (3.63) für das ursprüngliche Modell, das für ein Szenario reduzierte Modell (reduziertes Modell 1) und das für zwei Szenarien reduzierte Modell (reduziertes Modell 2) gezeigt. Die Resonanzüberhöhung bei $0,15$Hz des für beide Szenarien reduzierten Modells ist nun identisch zu der Resonanzüberhöhung des ursprünglichen Modells. Die Dämpfung sorgt nun dafür, dass die Übergänge im Phasengang glatter sind. Es ist jedoch auch zu beobachten, dass das für zwei Szenarien reduzierte Modell für Frequenzen ab 8Hz weiter von dem ursprünglichen Modell entfernt ist als das für ein Szenario reduzierte Modell. Der Grund hierfür liegt auf der Hand. In beiden Szenarien tauchen keine Frequenzen über 8Hz

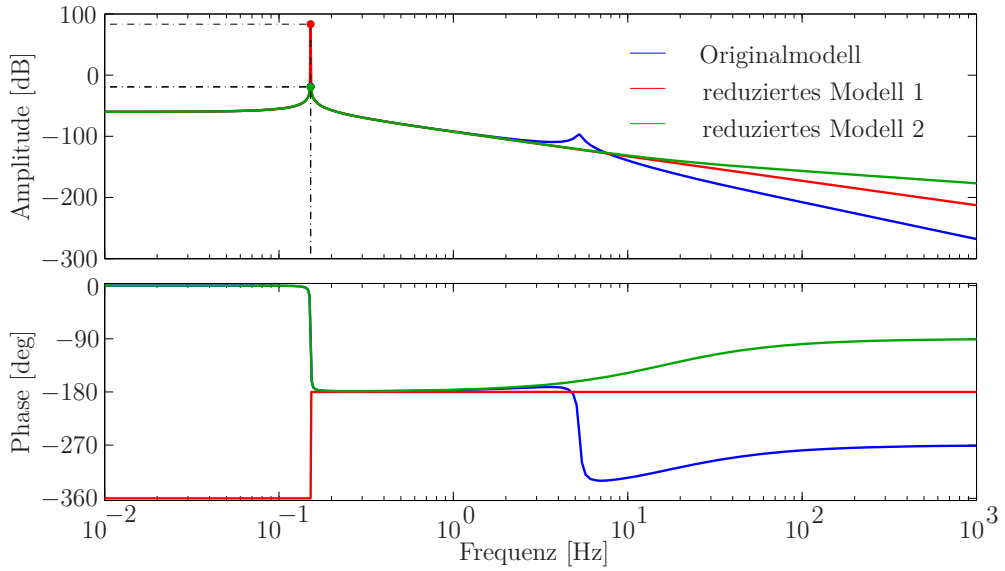


Abbildung 3.7: Bode-Diagramm der Übertragungsfunktion G

auf. Durch entsprechende Szenarienwahl wäre es auch möglich, ein reduziertes Modell zu generieren, welches für hohe Frequenzen geeignet ist. Mit steigender Anzahl an verschiedenen Szenarien werden jedoch Originalmodell und reduziertes Modell ähnlicher.

3.4.3 Erweiterung für intervallwertige Szenarien

Im vorherigen Abschnitt wurde die Erweiterung des Approximationsalgorithmus für mehrere Szenarien vorgestellt. In der Praxis ist es häufig der Fall, dass das reduzierte Modell nicht für mehrere Szenarien, sondern für ganze Bereiche von Szenarien die Fehlertoleranz nicht überschreiten soll. Es ist also wünschenswert Intervalle für die Parameter angeben zu können sowie die Möglichkeit zu haben, Anfangswerte und Systemeingänge intervallwertig zu gestalten. Eine Möglichkeit wäre, die Intervalle zu diskretisieren und die Reduktion für diese diskreten Szenarien durchzuführen. Jedoch ist auf diese Art und Weise nicht sichergestellt, dass das reduzierte Modell auch zwischen den diskreten Punkten innerhalb der Fehlertoleranz verbleibt. Der Zweimassenschwinger eignet sich hier als Beispiel. Soll der Zweimassenschwinger für einen ganzen Bereich von externen Anregungen reduziert werden, so besteht eine Möglichkeit darin, das gewünschte Frequenzspektrum zu diskretisieren. In der Folge wird während der Reduktion der Verbleib innerhalb der vorgegebenen Fehlertoleranz für alle Punkte der Diskretisierung überprüft. Liegt nun die Resonanzfrequenz im betrachteten Intervall, gehört aber nicht zu diskreten Punkten, so wird das reduzierte Modell mit großer Sicherheit die Fehlertoleranz verletzen. Daher wurde im Rahmen dieser Arbeit der Reduktionsalgorithmus erweitert, so dass nun auch die Vorgabe intervallwertiger Szenarien möglich ist.

Darüber hinaus existiert ein zweiter Grund, Intervalle anstatt von vielen diskreten Punkten zu verwenden. In der Regel sind in einem physikalischen System nicht alle

Parameter exakt bekannt. Eine gängige Methode besteht darin, die mit Unsicherheiten behafteten Parameter durch Intervalle abzubilden.

Um Modelle mit intervallwertigen Szenarien in den Approximationsalgorithmus einzubinden ist ein geeignetes numerisches Integrationsverfahren notwendig. Hierzu wird im nächsten Abschnitt zunächst eine Intervallarithmetik eingeführt. Danach wird ein numerisches Integrationsverfahren für intervallwertige Modelle vorgestellt und die Integration in den Approximationsalgorithmus erläutert.

Intervallarithmetik

Ein Intervall $\mathfrak{a}$ ist definiert als die Menge reeller Zahlen, die zwischen einer gegebenen oberen und unteren Grenze liegen. Also:

$$\mathfrak{a} = [\underline{a}, \overline{a}] = \{r \in \mathbb{R} \mid \underline{a} \leq r \leq \overline{a}\}\,. \tag{3.77}$$

Die untere Grenze wird dabei meist *Infimum* und die obere Grenze *Supremum* genannt. Ein n-dimensionaler Intervallvektor

$$\mathfrak{a} = \begin{bmatrix} \mathfrak{a}_1 & \mathfrak{a}_2 & \dots & \mathfrak{a}_n \end{bmatrix}^T \tag{3.78}$$

kann dabei geometrisch als n-dimensionaler Quader oder Box interpretiert werden. Die elementaren Operationen sind definiert durch

$$\mathfrak{a} \circ \mathfrak{b} = \{a \circ b \mid a \in \mathfrak{a}, b \in \mathfrak{b}\} \tag{3.79}$$

mit $\circ \in \{+, -, \cdot, /\}$. Auf dieselbe Art und Weise können auch sämtliche Funktionen für Intervalle definiert werden. Für eine Funktion f heißt $f(\mathfrak{a})$ dann Intervallauswertung der Funktion f.

Die Intervallauswertung $f(\mathfrak{a})$ einer Funktion f ist also ein Intervall, das $f(r)$ für alle $r \in \mathfrak{a}$ beinhaltet. Diese *natürliche* Intervallerweiterung hat allerdings einen großen Nachteil. Im Allgemeinen ist das Intervall $f(\mathfrak{a})$ größer als nötig. Die Intervallauswertung einer Funktion neigt also zur Überschätzung. Ein Grund hierfür ist das so genannte *Abhängigkeitsproblem*. Taucht eine Variable mehrmals in einer Gleichung auf, so ist Intervallarithmetik nicht in der Lage, dies zu identifizieren [Neher *et al.*, 2008].

Beispiel 3.4.3. Gegeben sei die Funktion

$$f(x) = \frac{1+x}{x}. \tag{3.80}$$

Auf dem Intervall $\mathfrak{a} = [1, 2]$ gilt

$$f(a) \in \left[\frac{3}{2}, 2\right] \quad \forall\, a \in \mathfrak{a}. \tag{3.81}$$

Die Intervallauswertung liefert jedoch

$$f(\mathfrak{a}) = \frac{1 + [1,2]}{[1,2]} = [2,3] \cdot \left[\frac{1}{2}, 1\right] = [1,3]\,. \tag{3.82}$$

Für die Funktion $g(x) = 1+\frac{1}{x}$ gilt offensichtlich $f(x) = g(x)$. Die Intervallauswertung von g liefert hier jedoch

$$g(\mathfrak{a}) = \left[\frac{3}{2}, 2\right], \tag{3.83}$$

da die Variable x nur noch einmal auftaucht. Das Abhängigkeitsproblem kann in einigen Fällen durch Umformulierung gelöst werden. Alternative grundlegende Möglichkeiten zur Minderung des Abhängigkeitsproblems werden in [Moore, 1966; Ratschek & Rokne, 1984] vorgestellt.

Neben dem Abhängigkeitsproblem ist auch der *Wrapping-Effekt* ein Grund für die Überschätzung. Das klassische Beispiel für den Wrapping-Effekt stammt aus [Moore, 1966]:

Beispiel 3.4.4. In diesem Beispiel wird der Einfluss des Wrapping-Effekts veranschaulicht. Hierzu wird die ODE

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \mathbf{x} \tag{3.84}$$

mit

$$\mathbf{x}(0) = \mathbf{x}_0 \tag{3.85}$$

verwendet. Der Vektor $\dot{\mathbf{x}}$ steht zu jedem Zeitpunkt senkrecht auf $\mathbf{x}$. Im Falle eines mechanischen Beispiels steht der Geschwindigkeitsvektor stets normal zu dem Positionsvektor. Die Lösung lässt sich also schreiben als

$$\mathbf{x}(t) = \underbrace{\begin{bmatrix} \cos t & \sin t \\ -\sin t & \cos t \end{bmatrix}}_{\mathbf{R}} \mathbf{x}_0. \tag{3.86}$$

Man beachte dabei, dass $\mathbf{R}$ eine Rotationsmatrix ist. Obwohl intervallwertige ODEs formal erst im nächsten Abschnitt eingeführt werden, wird nun die Lösung für einen intervallwertigen Startwert betrachtet. Ein intervallwertiger Startwert $\mathfrak{x}_0$ stellt ein Rechteck im $\mathbb{R}^2$ dar. Folglich ist die Lösung über die Zeit in diesem Fall ein rotierendes Rechteck.

Versucht man nun die Lösung an einem Zeitpunkt $t \neq 0$ mit einem Intervallvektor (Box) einzuschließen, so muss das Volumen dieser Box größer sein als das Volumen des ursprünglichen Rechtecks. Dieser Effekt ist in Abbildung 3.8 veranschaulicht. Die Lösung der ODE (3.86) ist für das Startquadrat mit der Kantenlänge $\frac{1}{5}$ zu den Zeitpunkten $t_0 = 0, t_1 = \frac{\pi}{4}$ und $t_2 = \frac{\pi}{2}$ gezeigt. Die durchgezogenen Rechtecke stellen dabei die exakte Lösung dar. Das gestrichelte Rechteck bei $\mathfrak{x}_1$ zeigt die Box, die die exakte Lösung einschließt und minimales Volumen hat. Das gepunktete Rechteck bei $\mathfrak{x}_3$ schließt wiederum die Lösung im nächsten Schritt ein.

Um den Einfluss des Abhängigkeitsproblems und des Wrapping-Effekts zu mindern, werden zur Lösung von ODEs mit intervallwertigen Parametern und Anfangswerten

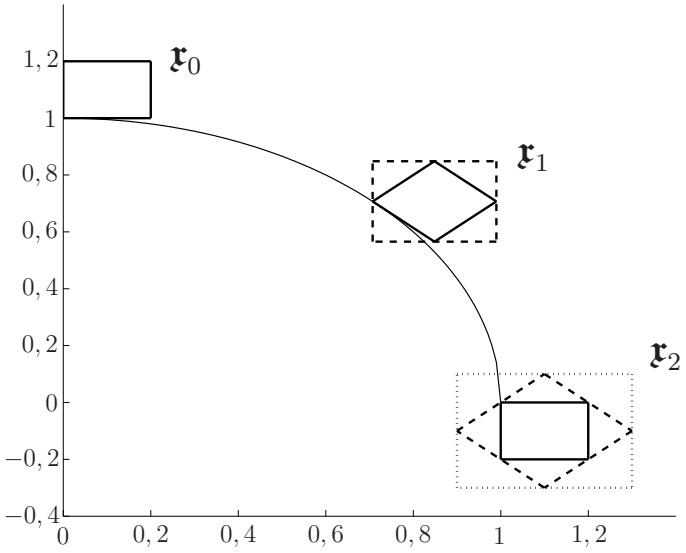


Abbildung 3.8: Veranschaulichung des Wrapping-Effekts

nicht die natürliche Intervallerweiterung, sondern *Taylor-Modelle* verwendet [Berz, 1997]. Das Taylor-Modell T_f einer $(q+1)$-mal differenzierbaren Funktion f besteht aus einer abgebrochenen Taylorreihe p^q der Funktion um einen Entwicklungspunkt $\mathbf{x}_0$ sowie einer intervallwertigen Abschätzung $\mathfrak{r}$ des Restglieds [Neher, 2005]:

$$T_f = p^q + \mathfrak{r}. \tag{3.87}$$

Dabei ist p^q ein Polynom der Ordnung q. Im skalaren Fall kann das Restglied geschrieben werden als

$$\mathfrak{r} = \frac{f^{(q+1)}(x_0 + (x - x_0) \cdot \mathfrak{d})}{(1+q)!}(x - x_0)^{q+1} \tag{3.88}$$

mit

$$\mathfrak{d} = [0, 1]\,. \tag{3.89}$$

Aus Gründen der Übersichtlichkeit wird hier auf die Notation des vektorwertigen Falls verzichtet. Das Restglied wird dann mit Hilfe der natürlichen Intervallerweiterung berechnet. Für das Taylor-Modell von f gilt dann

$$f \in T_f, \tag{3.90}$$

und ist somit ein Einschluss der Funktion f. Die Verwendung von Taylor-Modellen führt zu geringerer Überschätzung als die natürliche Intervallerweiterung. Bei der Rechnung mit Taylor-Modellen wird die symbolische Form der Polynome verwendet, so dass hier fast kein Abhängigkeitsproblem oder Wrapping-Effekt auftritt. So wurde in [Berz & Makino, 1998] eine Arithmetik für Taylor-Modelle entwickelt. Für die Taylor-Modelle $T_f = p_f^q + \mathfrak{r}_f$ und $T_g = p_g^q + \mathfrak{r}_g$ zweier Funktionen $f : \mathfrak{x} \mapsto \mathbb{R}$ und $g : \mathfrak{x} \mapsto \mathbb{R}$ gilt beispielsweise

$$f \pm g \in T_{f \pm g} = (p_f^q \pm p_g^q) + (\mathfrak{r}_f \pm \mathfrak{r}_g), \tag{3.91}$$

denn

$$f \pm g \in T_f \pm T_g = (p_f^q + \mathfrak{r}_f) \pm (p_g^q + \mathfrak{r}_g) = (p_f^q \pm p_g^q) + (\mathfrak{r}_f \pm \mathfrak{r}_g). \tag{3.92}$$

Die Konstruktion des Taylor-Modells der Summe zweier Funktionen ist also recht einfach. Die Konstruktion des Taylor-Modells, des Produktes zweier Funktion gestaltet sich jedoch schon schwieriger. Es gilt

$$f \cdot g \in T_f \cdot T_g = p_f^q \cdot p_g^q + p_f^q \cdot \mathfrak{r}_g + p_g^q \cdot \mathfrak{r}_f + \mathfrak{r}_f \cdot \mathfrak{r}_g. \tag{3.93}$$

Das Polynom $p_f^q \cdot p_g^q$ hat dabei die Ordnung $2q$. Daher wird $p_f^q \cdot p_g^q$ aufgeteilt in ein Polynom $p_{f\cdot g}^q$ der Ordnung q, in dem alle Terme bis zur Ordnung q enthalten sind und ein Polynom $p_{f\cdot g}^{2q}$ der Ordnung $2q$. Dann kann das Taylor-Modell von $f \cdot g$ geschrieben werden als

$$T_{f\cdot g} = p_{f\cdot g}^q + \underbrace{p_{f\cdot g}^{2q}(\mathfrak{x}) + p_f^q(\mathfrak{x})\mathfrak{r}_g + p_g^q(\mathfrak{x})\mathfrak{r}_f + \mathfrak{r}_f \cdot \mathfrak{r}_g}_{\mathfrak{r}_{f\cdot g}}. \tag{3.94}$$

Mit Hilfe dieser Arithmetik können basierend auf den Taylor-Modellen der elementaren Funktionen Taylor-Modelle für beliebige Funktionen bestimmt werden. Zur Auswertung der Polynome wurden dabei effiziente Methoden entwickelt, die die Überschätzung minimieren [Makino & Berz, 2004].

Beispiel 3.4.5. In diesem Beispiel wird ein Taylor-Modell einer trigonometrischen Funktion bestimmt. Sei

$$f : [0, 2] \mapsto \mathbb{R}, x \mapsto \sin(x), \tag{3.95}$$

$$q = 3 \tag{3.96}$$

und

$$x_0 = 0, \tag{3.97}$$

dann gilt

$$\sin(x) \in x - \frac{x^3}{6} + \frac{\sin(x \cdot \mathfrak{d})}{24}x^4, \quad \forall x \in [0, 2] \tag{3.98}$$

mit

$$\mathfrak{d} = [0, 1]\,. \tag{3.99}$$

Somit kann das Taylor-Modell geschrieben werden als $T_f = p_3 + \mathfrak{r}$ mit

$$p_3 = x - \frac{x^3}{6} \tag{3.100}$$

und

$$\mathfrak{r} = \left[0, \frac{2}{3}\right]. \tag{3.101}$$

Für $x \in [0, 1]$ gilt sogar die bessere Abschätzung

$$\mathfrak{r} = \left[0, \frac{1}{24}\right]. \tag{3.102}$$

Intervallwertige ODEs

Mit Hilfe der im letzten Abschnitt erläuterten Intervallarithmetik ist es nun möglich, intervallwertige ODEs und entsprechende numerische Integrationsverfahren einzuführen. Während verschiedene Verfahren und Softwarepakete für die Lösung von intervallwertigen ODEs existieren, gibt es nach Kenntnis des Autors nur ein Verfahren, um intervallwertige DAEs zu lösen. In [Nedialkov, 2006] werden die bekannten Verfahren zur Lösung von intervallwertigen ODEs und die entsprechenden Softwarepakete vorgestellt. In [Rauh *et al.*, 2009] wird ein Verfahren zur Lösung von intervallwertigen DAEs präsentiert. Hierzu wird das numerische Integrationsverfahren für intervallwertige ODEs VALENCIA-IVP um ein Intervall-Newton-Verfahren erweitert.

Im Rahmen dieser Arbeit wird VSPODE von Lin und Stadtherr verwendet, da es zu den leistungsfähigsten Verfahren zur Lösung von intervallwertigen ODEs zählt und eine C++-Implementierung für wissenschaftliche Zwecke verfügbar ist [Lin & Stadtherr, 2006].

Eine ODE

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t), \tag{3.103}$$

mit intervallwertigen Startwerten

$$\mathbf{x}(t_0) = \mathbf{x}_0 \in \mathfrak{x}_0 \tag{3.104}$$

und intervallwertigen Parametern $\mathfrak{p}$ *heißt intervallwertige ODE.*

Die Menge der Lösungen, die aus den Startwerten $\mathfrak{x}_0$ entstammen, wird mit

$$\mathfrak{x}(t, \mathfrak{x}_0) = \{\mathbf{x}(t, \mathbf{x}_0) \mid \mathbf{x}_0 \in \mathfrak{x}_0\} \tag{3.105}$$

bezeichnet, wobei $\mathbf{x}(t, \mathbf{x}_0)$ die zum Startwert $\mathbf{x}_0$ gehörende Lösung ist.

Ziel eines numerischen Integrationsverfahrens ist es, für jeden Zeitpunkt t_i einen Intervallvektor $\mathfrak{x}_i$ zu finden, so dass

$$\mathfrak{x}(t_i, \mathfrak{x}_0) \subseteq \mathfrak{x}_i, \quad 1 \leq i \leq n_t \tag{3.106}$$

gilt. Die meisten numerischen Integrationsverfahren, so auch VSPODE, gehen dabei in zwei Phasen vor:

1. In der ersten Phase wird ein Intervallvektor $\tilde{\mathfrak{x}}_i$ bestimmt, so dass Gleichung (3.103) für alle $\mathbf{x}_i \in \mathfrak{x}_i$ auf dem Intervall $[t_i, t_{i+1}]$ eindeutig lösbar ist und

   $$\mathfrak{x}(t, \mathfrak{x}_i) \subseteq \tilde{\mathfrak{x}}_i, \quad \forall t \in [t_i, t_{i+1}] \tag{3.107}$$

   gilt. Dabei kann $\tilde{\mathfrak{x}}_i$ die exakte Lösung grob überschätzen.

2. In der zweiten Phase wird unter Verwendung von $\tilde{\mathfrak{x}}_i$ ein engerer Einschluss der Lösung $\mathfrak{x}_{i+1} \subseteq \tilde{\mathfrak{x}}_i$ berechnet, so dass

$$\mathfrak{x}(t_{i+1}, \mathfrak{x}_0) \subseteq \mathfrak{x}_{i+1} \tag{3.108}$$

gilt.

In [Corliss & Rihm, 1996] wird gezeigt, dass Gleichung (3.103) für alle $\mathbf{x}_i \in \mathfrak{x}_i$ auf dem Intervall $[t_i, t_{i+1}]$ eindeutig lösbar ist, falls

$$\sum_{j=0}^{q} f^{[j]}(\mathbf{x}_i)(t - t_i)^j + f^{[q+1]}(\tilde{\mathfrak{x}}_i)(t - t_i)^{q+1} \subseteq \tilde{\mathfrak{x}}_i \tag{3.109}$$

für alle $t \in [t_i, t_{i+1}]$ mit $\mathbf{x}_i \in \tilde{\mathfrak{x}}_i$ gilt. Dabei bezeichnet $f^{[j]}(\mathbf{x}_i)$ die Koeffizienten der Taylorreihe von f für einen Entwicklungspunkt t_i und einer Lösung $\mathbf{x}_i$ von Gleichung (3.103) an der Stelle t_i. In [Nedialkov *et al.*, 2001] wird eine effiziente Methode vorgestellt, um sowohl $\tilde{\mathfrak{x}}_i$ als auch die Schrittweite $h_i = t_{i+1} - t_i$ aus Gleichung (3.109) zu bestimmen. Eine weitere Möglichkeit besteht darin, Gleichung (3.109) durch Fixpunktiteration zu lösen. Die ersten Verfahren zur Lösung von intervallwertigen ODEs aus [Moore, 1966] und [Lohner, 1988] basierten auf dieser Idee und $q = 0$. In diesem Fall lautet die Iterationsvorschrift

$$\tilde{\mathfrak{x}}_i^{l+1} = \mathfrak{x}_i + [0, h_i]\, f(\tilde{\mathfrak{x}}_i^l). \tag{3.110}$$

Diese Intervall-Iteration wird durchgeführt bis

$$\tilde{\mathfrak{x}}_i^{l+1} \subseteq \tilde{\mathfrak{x}}_i^l \tag{3.111}$$

gilt. Man beachte, dass man im reellwertigen Fall das implizite Euler-Verfahren mit Fixpunktiteration erhält.

In der zweiten Phase wird nun basierend auf $\tilde{\mathfrak{x}}_i$ ein Taylor-Modell $T_{x_{i+1}}$ von $\mathbf{x}_{i+1}$ erstellt. Die Intervallauswertung von $T_{x_{i+1}}$ liefert dann den Intervallvektor $\mathfrak{x}_{i+1}$.

Sei also die Lösung von Gleichung (3.103) an der Stelle t_i für alle $\mathbf{x}_0 \in \mathfrak{x}_0$ gegeben durch das Taylor-Modell

$$T_{x_i} = p_i^q + \mathfrak{r}_i. \tag{3.112}$$

Dann gilt

$$\mathfrak{x}(t_{i+1}, \mathfrak{x}_0) \subseteq \mathfrak{x}_i + \sum_{j=1}^{q} h_i^j f^{[j]}(\mathfrak{x}_i) + \tilde{\mathfrak{z}}_{i+1}, \tag{3.113}$$

wobei $\tilde{\mathfrak{z}}_{i+1}$ das Restglied in der Form von Gleichung (3.88) ist und durch

$$\mathfrak{z}_{i+1} = h^{q+1} f^{[q+1]}(\tilde{\mathfrak{x}}_i) \tag{3.114}$$

eingeschlossen werden kann. Anstatt eine Intervallauswertung der Koeffizienten der Taylorreihe $f^{[j]}$ zu berechnen, verwendet VSPODE Taylor-Modelle $T_{f^{[j]}}$, um diese

Koeffizienten einzuschließen und verwendet hierzu die im vorherigen Abschnitt vorgestellte Arithmetik. Dabei ist zu beachten, dass die entsprechenden Taylorpolynome bezüglich $\mathbf{x}$ und nicht bezüglich t aufgestellt werden. Die Berechnung aller notwendigen Taylorkoeffizienten erfolgt mit Hilfe automatischer Differentiation. Hierzu werden die Softwarepakete TADIFF beziehungsweise FADBAD++ verwendet [Bendtsen & Stauning, 1996, 1997]. Formal liefert dieses Vorgehen

$$\mathfrak{x}_{i+1} = T_{x_i} + \sum_{j=1}^{q} h_i^j T_{f^{[j]}} + \mathfrak{z}_{i+1}. \tag{3.115}$$

Die auf diese Art und Weise konstruierte Lösung bringt jedoch einen Nachteil mit sich. Die Größe eines Intervalls, welches durch Summenbildung aus zwei Intervallen entsteht, ist gleich der Summe der Größe der beiden Summanden. Damit impliziert Gleichung (3.91), dass das Intervall $\mathfrak{r}_{i+1}$ größer ist als $\mathfrak{r}_i$. Die Größe des Restglieds steigt also monoton. Dieses Problem kann durch Anwendung des Mittelwertsatzes der Differentialrechnung aufgehoben werden. Hierauf wird an dieser Stelle jedoch nicht weiter eingegangen. Weitere Details findet man in [Lin & Stadtherr, 2007].

Integration von intervallwertigen Szenarien in den Approximationsalgorithmus

Im Rahmen dieser Arbeit wurde das oben beschriebene Verfahren VSPODE in den Approximationsalgorithmus integriert, um die Vorgabe intervallwertiger Szenarien zu ermöglichen. Zwar ist VSPODE nicht in der Lage, intervallwertige DAEs zu lösen, doch stellt dies keine wesentliche Einschränkung dar. In den meisten objektorientierten Simulationsumgebungen werden die mathematischen Modelle ohnehin in ODEs transformiert. Im Folgenden werden die Modifikationen bei den in Abschnitt 3.3 vorgestellten Ranking-Verfahren sowie der neue Approximationsalgorithmus präsentiert.

Ranking-Verfahren Von den vorgestellten Ranking-Verfahren ist lediglich das Residuenranking für die Vorgabe von intervallwertigen Szenarien geeignet. Jedoch liefert VSPODE als Lösung lediglich die Zustände $\mathfrak{x}$ und nicht den Intervallvektor $\dot{\mathfrak{x}}$. Dieser muss folglich durch eine zusätzliche Intervallauswertung von f beschafft werden. Dies führt zu einer noch geringeren Güte des Residuenrankings bedingt durch Überschätzung.

Das Einschrittranking basiert auf einem vereinfachten Newton-Verfahren zur Lösung eines nichtlinearen Gleichungssystems. Bei der Verwendung von VSPODE wird jedoch kein nichtlineares Gleichungssystem gelöst. Alternativ könnte die erste Phase von VSPODE durch eine Iteration von Gleichung (3.110) mit der Referenzlösung als Startwert ersetzt werden. Da der Großteil der Laufzeit jedoch für die Berechnung der Koeffizienten der Taylorpolynome benötigt wird, welche in der zweiten Phase ohnehin benötigt werden, scheint auch dieser Ansatz nicht besonders vielversprechend.

Bisher ist es nicht möglich, mit VSPODE eine Sensitivitätsanalyse durchzuführen. Folglich kann auch das Sensitivitätsranking nicht verwendet werden.

Wie schon oben erwähnt, hängt die Genauigkeit und Laufzeit von VSPODE maßgeblich von der Ordnung der verwendeten Taylorpolynome ab. Daher wird im Rahmen dieser Arbeit das Lösen der intervallwertigen ODE mit VSPODE und Taylorpolynomen der Ordnung $q = 3$ als Ranking-Verfahren verwendet. Die Ordnungen $q = 1$ und $q = 2$ sind hierzu nicht geeignet, da in diesen Fällen die Integration häufig abbricht, da die Größe der Intervalle zu stark anwächst.

Approximationsalgorithmus Laut Algorithmus 1 würde eine Reduktion κ durchgeführt, falls

$$\left\| \mathfrak{x}^{\star}_{\text{out},i} - \mathfrak{x}_{\text{out},i} \right\| < \epsilon_{\text{Fehler},i},\ 1 \leq i \leq N_{\text{out}}. \tag{3.116}$$

Für zwei Intervalle $\mathfrak{a}$ und $\mathfrak{b}$ ist die Subtraktion gemäß Gleichung (3.79) definiert durch

$$\mathfrak{a} - \mathfrak{b} = \left[\overline{a} - \underline{b}, \underline{a} - \overline{b} \right]. \tag{3.117}$$

In der Folge repräsentiert Gleichung (3.116) die durch die Reduktion κ bedingte Änderung der Lösung nicht richtig. Ein besseres Kriterium stellt die Ungleichung

$$\max_{Op \in \{\text{Sup,Inf}\}} \left| Op(\mathfrak{x}^{\star}_{\text{out},i}) - Op(\mathfrak{x}_{\text{out},i}) \right| < \epsilon_{\text{Fehler},i}, \quad 1 \leq i \leq N_{out} \tag{3.118}$$

dar [Mikelsons & Brandt, 2009]. Die linke Seite von Gleichung (3.118) berechnet dabei das Maximum der Abweichung zwischen Supremum und Infimum der Referenzlösung $\mathfrak{x}^{\star}$ und $\mathfrak{x}$. Mit $\boldsymbol{\epsilon}_{\text{Fehler}}$ wird also die maximale Verschiebung bezüglich der Referenzlösung begrenzt.

Beispiel 3.4.6. Wie schon weiter oben erwähnt, entspricht VSPODE im reellwertigen Fall einem impliziten Integrationsverfahren mit Fixpunktiteration. Dies führt zu der Vermutung, dass VSPODE (wie auch alle anderen numerischen Verfahren für intervallwertige Szenarien) nicht für steife ODEs geeignet ist. Folglich lässt sich der Zweimassenschwinger nicht für die in den bisherigen Beispielen verwendeten Szenarien reduzieren. Statt dessen wird hier ein anderes Szenario verwendet. Die Parameter werden gewählt als

$$m_{M_1} = \left[1, \frac{11}{10}\right] \text{kg}, \quad m_{M_2} = 2\text{kg}, \quad c_{S_1} = 2\frac{\text{N}}{\text{m}}, \quad c_{S_2} = 1\frac{\text{N}}{\text{m}}, \tag{3.119}$$

$$d_{S_1} = 0,2\frac{\text{Ns}}{\text{m}}, \quad d_{S_2} = 0,2\frac{\text{Ns}}{\text{m}} \quad \text{und} \quad x^L = 0\text{m}. \tag{3.120}$$

Ferner wird der Zweimassenschwinger nicht durch eine externe Kraft angeregt, und es werden die Anfangswerte

$$x_{M_1} = 1\text{m}, v_{M_1} = 0\frac{\text{m}}{\text{s}}, x_{M_2} = \left[\frac{3}{2}, \frac{8}{5}\right] \text{m sowie} \quad v_{M_2} = 0\frac{\text{m}}{\text{s}} \tag{3.121}$$

verwendet. Man beachte hierbei, dass sowohl m_{M_1} als auch x_{M_2} intervallwertig sind. Die entsprechende Referenzlösung ist in Abbildung 3.9 dargestellt. Die Reduktion

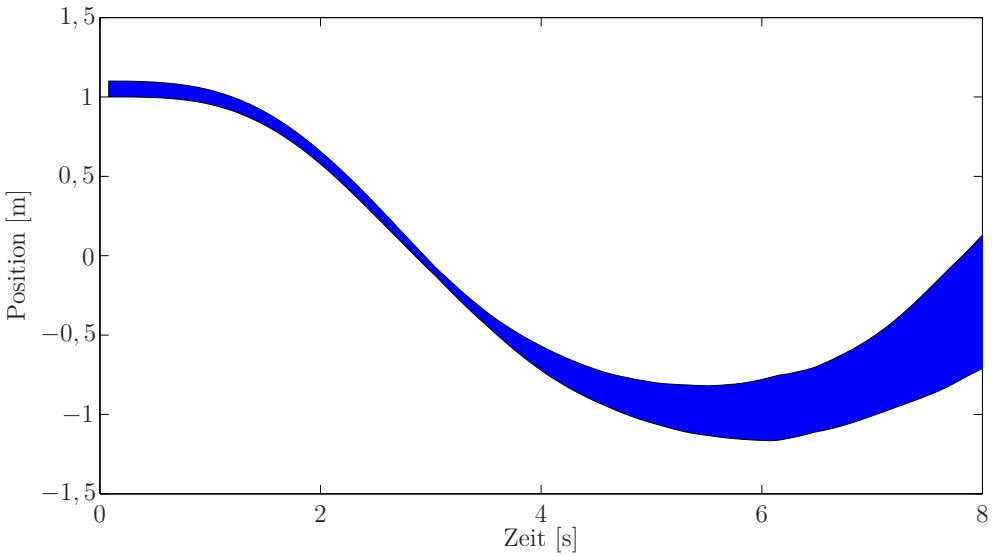


Abbildung 3.9: Intervallwertige Referenzlösung

mit einer Fehlertoleranz von $\epsilon = 0,25\text{m}$ für die Position x_{M_2} der zweiten Masse und einer Simulationszeit von 8s liefert das Modell

$$x_{S_1,\text{rel}} = x_{M_1} - x_L, \tag{3.122}$$
$$x_{S_2,\text{rel}} = x_{M_2}, \tag{3.123}$$
$$F_{S_1} = c_{S_1} \cdot x_{S_1,\text{rel}}, \tag{3.124}$$
$$F_{S_2} = c_{S_2} \cdot x_{S_2,\text{rel}}, \tag{3.125}$$
$$\dot{x}_{M_1} = v_{M_1}, \tag{3.126}$$
$$\dot{v}_{M_1} = \frac{1}{m_{M_1}}(F_{S_2} - F_{S_1}), \tag{3.127}$$
$$\dot{x}_{M_2} = v_{M_2}, \tag{3.128}$$
$$\dot{v}_{M_2} = \frac{1}{m_{M_2}}(F_{\text{ext}} - F_{S_2}). \tag{3.129}$$

Aufgrund des im Vergleich zu den vorherigen Beispielen geänderten Szenario kann m_{M_1} nun natürlich nicht mehr vernachlässigt werden. Daher ist der Unterschied zwischen dem ursprünglichen und dem reduzierten Modell lediglich in der vernachlässigten Dämpfung zu finden. In Abbildung 3.10 sind die Lösungen beider Modelle gezeigt.

3.5 Reduktion für Echtzeitsimulationen

Im vorherigen Abschnitt wurde die gleichungsbasierte Reduktion für eine vorgegebene Fehlergrenze vorgestellt. In der Praxis ist es jedoch häufig der Fall, dass ein Modell nicht reduziert werden soll bis eine Fehlergrenze erreicht ist. Vielmehr ist es gewünscht ein Modell zu reduzieren, bis es innerhalb einer bestimmten Zeitdauer

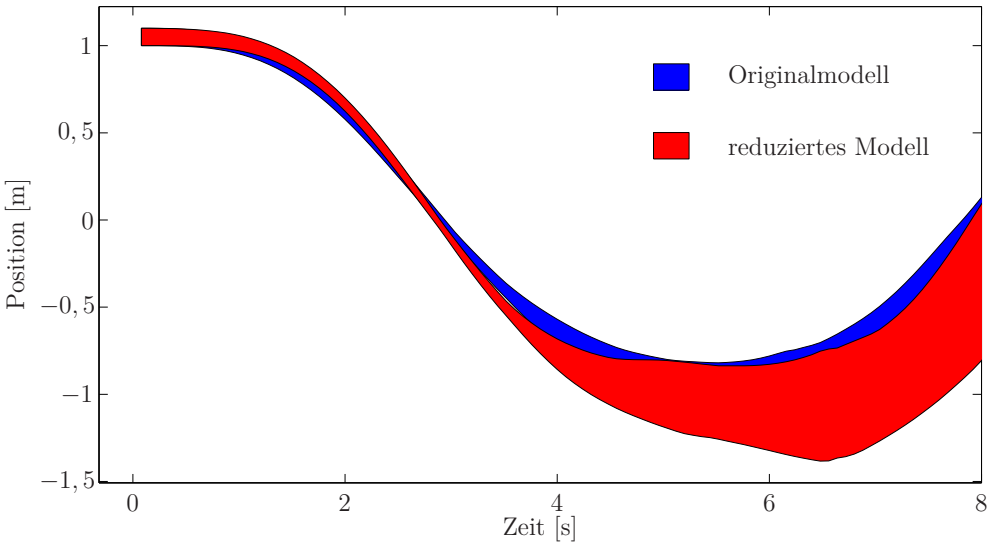


Abbildung 3.10: Vergleich der Referenzlösung mit der Lösung des reduzierten Modells

simuliert werden kann. Dieser Wunsch tritt insbesondere auf, falls das Modell in Echtzeit simuliert werden soll. Eine der wesentlichen im Rahmen dieser Arbeit entwickelten Neuerungen der gleichungsbasierten Modellreduktion ist die Möglichkeit, ein Modell solange zu reduzieren, bis es auf einer vorgegebenen Echtzeit-Hardware in einem vorgegebenen Echtzeit-Takt simuliert werden kann. Der entsprechende Algorithmus wird in Abschnitt 3.5.2 vorgestellt. Zusätzlich wird eine Erweiterung des Algorithmus vorgestellt, so dass bei der Reduktion für Echtzeitsimulationen auch eine Fehlertoleranz vorgegeben werden kann.

Aufgrund der einfachen Verfügbarkeit leistungsfähiger Rechner und der zunehmenden Mechatronisierung ehemals rein mechanischer Systeme hat sich sich der Einsatzbereich von Echtzeitsimulationen vergrößert. So zählen heutzutage modellbasierte Regelung, virtuelle Inbetriebnahme oder Simulatoren zu denjenigen Technologien, bei denen auf die Echtzeitsimulation nicht verzichtet werden kann. Während bei Simulatoren meist leistungsfähige PC-Hardware zum Einsatz kommt, steht bei Regelungsaufgaben oftmals, schon allein aus Kostengründen, nur ein Mikrocontroller zur Verfügung. An dieser Stelle fällt es dann der Modellbildung zu, ein Modell zu generieren, welches so komplex ist, dass es die gewünschten Effekte abbildet, und gleichzeitig einfach genug, um in Echtzeit auf der gewünschten Echtzeit-Hardware simuliert werden zu können.

Damit ein Modell in Echtzeit simuliert werden kann, ist es notwendig, dass die Simulation schneller läuft als die Realzeit. Diese Bedingung ist jedoch keineswegs hinreichend. Zusätzlich muss am Ende von jedem Echtzeit-Takt garantiert ein Ergebnis zur Verfügung stehen. Dies bedeutet, dass die Modellkomplexität, die Echtzeit-Hardware und der Echtzeit-Takt aufeinander abgestimmt sein müssen. Auf den Begriff der Echtzeitsimulation wird in Abschnitt 3.5.1 näher eingegangen.

Mit Hilfe der gleichungsbasierten Modellreduktion für Echtzeitsimulationen kann der

Abbildung 3.11: DRIVASSIST® Simulator(links) und Lufthansa Flugsimulator (rechts)

Schritt der Anpassung eines Modells auf die Echtzeit-Hardware automatisiert werden. Dies ist insbesondere im Hinblick auf die Möglichkeit der automatisierten Code Erzeugung für Echtzeit-Systeme in vielen objektorientierten Simulationsumgebungen eine wichtige Errungenschaft. Bisher konnte der Code für die Echtzeit-Hardware zwar automatisiert erzeugt werden, eine Garantie, dass dieses Modell im gewünschten Echtzeit-Takt simuliert werden kann, gab es aber nicht.

3.5.1 Echtzeitsimulation

Eine Echtzeitsimulation läuft im Gegensatz zu einer Offline-Simulation synchron mit der Echtzeit. In der Folge kann der Mensch (Human-in-the-Loop) oder reale Hardware (Hardware-in-the-Loop) mit der Simulation in Echtzeit interagieren. Simulatoren sind ein weit verbreitetes Beispiel für Human-in-the-Loop. So werden beispielsweise Piloten zu einem großen Teil ihrer Ausbildung am Simulator geschult. Auch in der Automobilindusrie werden Simulatoren eingesetzt. Beispielsweise kann das Zusammenspiel von Fahrzeug, Mensch und den modernen Fahrassistenzsystemen nur schwer analytisch bewertet werden, da insbesondere die Rolle des Fahrers mit großen Unsicherheiten behaftet ist [Mikelsons & Brandt, 2010]. An dieser Stelle bietet sich der Einsatz eines Fahrsimulators an, um die Akzeptanz und Funktion neuer Assistenzsysteme zu untersuchen [Hesse *et al.*, 2009]. Weitere Vorteile für den Einsatz von Simulatoren liegen auf der Hand:

- Der Einsatz eines Simulators spart Kosten, da die Verwendung des realen Systems in der Regel teurer ist.
- Umgebungsbedingungen, wie zum Beispiel Wetter, können kontrolliert werden.
- Der Einsatz eines Simulators ist viel sicherer als die Verwendung des realen Systems. Auch Situationen, die im Grenzbereich des realen Systems liegen, können gefahrlos getestet werden.

In Abbildung 3.11 ist ein Fahrsimulator auf Basis des Simulatorkonzeptes DRIVASSIST® sowie ein Flugsimulator der Lufthansa gezeigt.

Bei einer Hardware-in-the-Loop-Simulation kommuniziert nicht ein Mensch, sondern eine reale Komponente mit der Echtzeitsimulation. Auf diese Art und Weise können

zum Beispiel Steuer- und Regelalgorithmen schnell und kostengünstig getestet werden. Eine virtuelle Inbetriebnahme ist ein Beispiel für eine Hardware-in-the-Loop-Simulation. Hier sind die Eingänge einer realen speicherprogrammierbare Steuerung (SPS) mit einem Modell der Dynamik des echten Systems verbunden, welchem beliebige Lasten aufgeprägt werden können. Auf diese Art und Weise wird das reale Steuergerät in dem virtuellen System getestet. Um die Aussagekraft der virtuellen Inbetriebnahme weiter zu erhöhen, kann zusätzlich der Datenaustausch auf dieselbe Art wie in der Realität vorgenommen werden. Häufig wird daher bei der virtuellen Inbetriebnahme derselbe Datenbus wie im realen System verwendet.

Ein weiteres Einsatzgebiet von Echtzeitsimulationen sind modellbasierte beziehungsweise modellprädiktive Regler. Hier wird ein Modell des realen physikalischen Systems in Echtzeit simuliert, um eine Regelung zu realisieren [Camacho & Bordons, 2004].

Im folgenden Abschnitt wird erläutert, wie ein Echtzeitsystem funktioniert, um danach auf die Besonderheiten der numerischen Integration in Echtzeit einzugehen.

Echtzeitsysteme

Echtzeitsysteme sind Computersysteme, die innerhalb genauer zeitlicher Vorgaben auf Ereignisse aus der Umgebung reagieren müssen. Daher beruht der korrekte Ablauf des Systems nicht nur auf der Richtigkeit der Ausgabe, sondern auch auf der Pünktlichkeit der Ausgabe [Buttazzo, 2005]. Verspätet eintreffende Ergebnisse können nutzlos oder in gewissen Systemen sogar gefährlich sein.

In Abbildung 3.12 sind die wichtigsten Zeitpunkte eines Taktes in einer Echtzeitsimulation gezeigt. Zum Anfangszeitpunkt T_a beginnt der Takt. Zum Zeitpunkt T_s beginnen die numerischen Berechnungen. Zwischen den Zeitpunkten T_a und T_s werden die Systemeingänge $\mathbf{u}$ aktualisiert. Hierzu werden die Analog/Digital-Konverter ausgelesen, die die analogen Eingänge digitalisieren. Im Falle des Fahrsimulators wäre ein solcher Eingang beispielsweise der Lenkradwinkel. Nach Beendigung des Integrationsschrittes zum Zeitpunkt T_f kommt ein Digital/Analog-Konverter zum Einsatz, um das Ergebnis von außen verfügbar zu machen. Zusätzlich können in der Zeitspanne zwischen T_f und T_d von außen gesteuerte Ereignisse, wie zum Beispiel das Ändern des Werts eines Parameters abgearbeitet werden. Zum Zeitpunkt T_d ist der Takt schließlich beendet. Spätestens hier muss das Ergebnis der numerischen Berechnungen vorliegen.

Die Zeitspanne zwischen dem Beginn des Taktes T_a und der Beendigung der numerischen Berechnungen heißt response time [Cooling & Cooling, 2003]. An dieser Stelle sei angemerkt, dass die response time bei einer Echtzeitsimulation nicht unbedingt besonders klein sein muss. Es ist ein verbreitetes Missverständnis, dass eine Echtzeitsimulation zwingendermaßen schnell ist. Vielmehr muss die response time vorhersagbar und beschränkt sein. Darüber hinaus muss die Zeit, die für einen Intgrationsschritt benötigt wird, natürlich kleiner sein als die Schrittweite.

Diese Bedingungen an die response time haben Folgen für die numerische Integration. Hierauf wird im nächsten Abschnitt eingegangen.

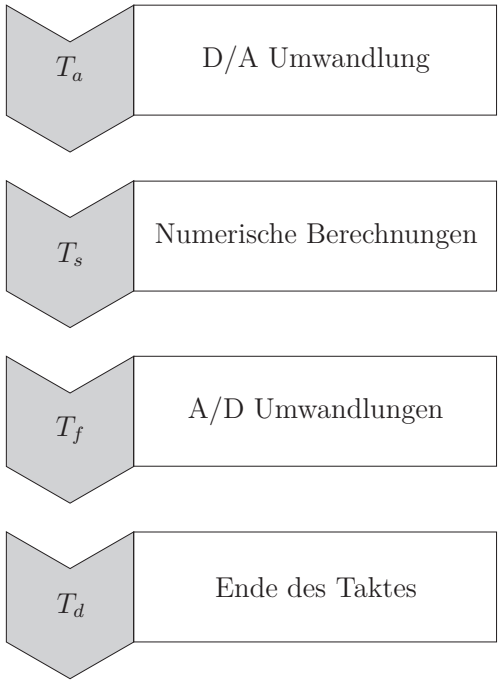


Abbildung 3.12: Ablauf eines Echtzeit-Taktes

Echtzeitfähige Integrationsverfahren

Eine beschränkte und vorhersagbare response time verlangt nach einem numerischen Integrationsverfahren, dessen Aufwand beschränkt und vorhersagbar ist. In der Folge muss auf Schrittweitensteuerung und iterative Algorithmen verzichtet werden. Somit können beispielsweise keine impliziten Verfahren bei Echtzeitsimulationen verwendet werden. Ein möglicher Ausweg an dieser Stelle ist die Begrenzung der maximalen Iterationen. Trotzdem stellt sich die Frage, welche numerischen Integrationsverfahren überhaupt für Echtzeitsimulationen geeignet sind.

Auf den ersten Blick erscheinen Mehrschrittverfahren gut geeignet für Echtzeitsimulationen, da pro Integrationsschritt lediglich eine Funktionsauswertung notwendig ist. Auf der anderen Seite verwenden Mehrschrittverfahren die Lösung an vorherigen Zeitpunkten (siehe auch Abschnitt 2.2.3), um diese zu interpolieren. Demzufolge wird vorausgesetzt, dass die Lösung in gewissem Sinne glatt ist. Häufig ist jedoch der Systemeingang bei Echtzeitsimulationen unstetig. Dies führt zu ungenauen Ergebnissen bei der Verwendung von Mehrschrittverfahren. Können in einem Zeitschritt Ereignisse auftreten, so dass das Verfahren neu gestartet werden muss, so haben Mehrschrittverfahren einen weiteren Nachteil. Der Neustart eines Mehrschrittverfahrens ist relativ aufwendig, da mehrere Startwerte benötigt werden (siehe auch Abschnitt 2.2.3). Dennoch sind Mehrschrittverfahren relativ weit verbreitet im Bereich der Echtzeitsimulation.

Explizite Einschrittverfahren verwenden keine Lösungen von vorherigen Schritten und können auf einfache Art und Weise neu gestartet werden. Darüber hinaus ist der numerische Aufwand vorhersagbar und beschränkt. Der Nachteil dieser Verfahren liegt in der geringen Effizienz bei steifen Modellen.

Implizite Einschrittverfahren sind zwar für die Lösung steifer Differentialgleichungen geeignet, benötigen jedoch die Lösung eines nichtlinearen Gleichungssystems in jedem Integrationsschritt. In der Folge ist der einzige Weg, um den numerischen Aufwand zu begrenzen, eine maximale Anzahl für die Iterationen im Newton-Verfahren festzulegen. Dieses Vorgehen senkt jedoch sowohl die Stabilität als auch die Genauigkeit des Verfahrens.

An dieser Gegenüberstellung ist schon zu sehen, dass die Echtzeitsimulation eines Systems, dessen mathematisches Modell eine steife ODE oder DAE darstellt, eine große Herausforderung ist. In beiden Fällen liegt diese Schwierigkeit in dem Verzicht auf implizite Verfahren begründet. Da sich jede DAE in eine (steife) ODE transformieren lässt, betrachten wir im Folgenden vorwiegend ODEs.

Trotzdem ist die Frage nach einem geeigneten numerischen Lösungsverfahren noch nicht beantwortet. In [Beutlich, 2009] wird das explizite Euler-Verfahren für Echtzeitsimulation mit Hilfe von Modelica generierten Modellen vorgeschlagen. Diese Wahl erscheint jedoch äußerst fragwürdig in Hinblick auf steife oder moderat steife Modelle. Wie in Abschnitt 2.2.3 gezeigt, ist es bei der Lösung steifer ODEs wünschenswert, dass das numerische Integrationsverfahren L-stabil ist.

Eine Alternative stellen die linear-impliziten Integrationsverfahren dar [Hairer & Wanner, 1991; Arnold *et al.*, 2007]. In dieser Klasse lassen sich insbesondere A-stabile und L-stabile Verfahren finden, die somit für die Integration steifer ODEs geeignet sind. In [Rill, 1994] wird beispielsweise das linear-implizite Euler-Verfahren zur Echtzeitsimulation von Kraftfahrzeugen verwendet. Die Verfahrensvorschrift erhält man durch Linearisierung von $\mathbf{f}(\mathbf{x}_{n+1})$ in der Verfahrensvorschrift des impliziten Euler-Verfahrens. Diese ist gegeben durch

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h \cdot \mathbf{f}(\mathbf{x}_{n+1}). \tag{3.130}$$

Die Linearisierung liefert

$$\mathbf{f}(\mathbf{x}_{n+1}) \approx \mathbf{f}(\mathbf{x}_n) + \underbrace{\frac{\partial \mathbf{f}}{\partial \mathbf{x}} |_{\mathbf{x}_n}}_{\mathbf{J}} (\mathbf{x}_{n+1} - \mathbf{x}_n), \tag{3.131}$$

woraus für das linear-implizite Euler-Verfahren

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h \cdot (\mathbf{f}(\mathbf{x}_n) + \mathbf{J} \cdot (\mathbf{x}_{n+1} - \mathbf{x}_n)) \tag{3.132}$$

$$\Leftrightarrow \mathbf{x}_{n+1} = \mathbf{x}_n + h \cdot (\mathbf{E} - h \cdot \mathbf{J})^{-1} \cdot \mathbf{f}(\mathbf{x}_n) \tag{3.133}$$

folgt. Hierbei ist $\mathbf{E}$ die Einheitsmatrix. Somit beruht ein Integrationsschritt auf der Lösung eines linearen Gleichungssystems und nicht mehr auf der Lösung eines nichtlinearen Gleichungssystems. Da der numerische Aufwand zur Lösung eines linearen Gleichungssystems (beispielsweise Gauss-Elimination) beschränkt und vorhersagbar ist, ist das linear-implizite Euler-Verfahren somit für Echtzeitsimulationen geeignet.

Darüber hinaus ist das linear-implizite Euler-Verfahren A-stabil. Betrachtet man die Testgleichung (2.141), so ist klar, dass hier

$$\mathbf{f}(\mathbf{x}_{n+1}) = \mathbf{f}(\mathbf{x}_n) + \mathbf{J}(\mathbf{x}_{n+1} - \mathbf{x}_n) \tag{3.134}$$

gilt. Dies impliziert sofort A-Stabilität. Neben der A-Stabilität *erbt* das linear-implizite Euler-Verfahren auch die L-Stabilität des impliziten Euler-Verfahrens.

Der einzige Nachteil dieser Methode ist, dass das linear-implizite Euler-Verfahren die Konsistenzordnung eins besitzt, da die Konsistenzordnung durch die Linearisierung nicht beeinträchtigt wird. Folglich liegt es nahe, anstatt des linear-impliziten Euler-Verfahrens die linear-implizite Trapezregel zu verwenden. Die Verfahrensvorschrift für die Trapezregel ist gegeben durch

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \frac{h}{2}(\mathbf{f}(\mathbf{x}_{n+1}, t_{n+1}) + \mathbf{f}(\mathbf{x}_n, \mathbf{t}_n)). \tag{3.135}$$

Dasselbe Vorgehen wie oben liefert die linear-implizite Trapezregel

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h \cdot (\mathbf{E} - \frac{h}{2} \cdot \mathbf{J})^{-1} \cdot \mathbf{f}(\mathbf{x}_n). \tag{3.136}$$

Die implizite Trapezregel hat ebenso wie die Trapezregel die Konsistenzordnung zwei. Jedoch erbt die linear-implizite Trapezregel auch die Stabilitätseigenschaften der Trapezregel. Hieraus folgt, dass die linear-implizite Trapezregel nicht L-stabil ist (siehe Abschnitt 2.2.3), was ein Nachteil in zweifacher Hinsicht ist. Zum einen ist die mit linear-impliziten Verfahren berechnete Lösung in der Regel relativ ungenau, da die Linearisierung aus Gleichung (3.131) dem Abbruch des Newton-Verfahrens nach einer Iteration entspricht. Ist das numerische Integrationsverfahren zusätzlich nicht L-stabil, so wird die Lösung nicht numerisch gedämpft und der Fehler kann sich sehr schnell akkumulieren. Zum anderen ist die linear-implizite Trapezregel nicht zum Lösen steifer ODEs geeignet.

In [Nikitin, 2006] wird eine Familie von linear-impliziten Verfahren mit der Konsistenzordnung drei vorgestellt, um die Navier-Stokes-Gleichungen zu lösen. In dieser Familie lässt sich ein Verfahren finden, das sowohl A-stabil als auch L-stabil ist. Dieses Verfahren wird hier zum ersten mal für Echtzeitsimulationen verwendet. Die Verfahrensvorschrift lautet

$$\mathbf{k}_1 = \mathbf{x}_n + \frac{2h}{3}\mathbf{L} \cdot \mathbf{f}(\mathbf{x}_n, t_n), \tag{3.137}$$

$$\mathbf{k}_2 = \mathbf{L}(\mathbf{x}_n - \frac{h}{2}\mathbf{J} \cdot \mathbf{k}_1 + \frac{h}{3}\mathbf{f}(\mathbf{x}_n, t_n) + \frac{h}{3}\mathbf{f}(\mathbf{k}_1, t_n + \frac{2h}{3})), \tag{3.138}$$

$$\bar{\mathbf{k}} = \frac{9}{4}\mathbf{k}_1 - \frac{3}{4}\mathbf{k}_2 - \frac{1}{2}\mathbf{x}_n, \tag{3.139}$$

$$\mathbf{k}_3 = \mathbf{L}(\mathbf{x}_n - \frac{h}{2}\mathbf{J} \cdot \bar{\mathbf{k}} + \frac{h}{4}\mathbf{f}(\mathbf{x}_n, t_n) + \frac{3h}{4}\mathbf{f}(\mathbf{k}_1, t_n + \frac{2h}{3})), \tag{3.140}$$

$$\mathbf{x}_{n+1} = \mathbf{L}(\mathbf{x}_n - \frac{h}{2}\mathbf{J} \cdot \bar{\mathbf{k}} + \frac{h}{4}\mathbf{f}(\mathbf{x}_n, t_n) + \frac{3h}{4}\mathbf{f}(\mathbf{k}_2, t_n + \frac{2h}{3})), \tag{3.141}$$

mit

$$\mathbf{L} = (\mathbf{E} - \frac{h}{2}\mathbf{J})^{-1}. \tag{3.142}$$

Das Verfahren benötigt also die Invertierung einer Matrix (oder die Lösung von vier linearen Gleichungssystemen) und drei Funktionsauswertungen. Den Nachweis für

die A- und L-Stabilität sowie der Konsistenzordnung ist in [Nikitin, 2006] zu finden. Im Folgenden wird dieses Verfahren LI3 genannt.

Bei der Wahl eines numerischen Integrationsverfahrens für Echtzeitsimulationen ist also entscheidend, wie steif die vorliegende ODE ist. Im Falle einer nicht steifen ODE kann ein explizites Runge-Kutta-Verfahren für die Echtzeitsimulation verwendet werden. Im Falle steifer ODEs ist die Verwendung von linear-impliziten Verfahren sinnvoll. Oben wurden zwei dieser Verfahren vorgestellt, die sowohl A- als auch L-stabil sind. Der Preis für das genauere Verfahren LI3 ist der höhere Aufwand. Hierfür kann jedoch eine größere Taktzeit realisiert werden.

Für die Schätzung des Aufwands spielt es natürlich eine große Rolle, ob die Jacobi-Matrix numerisch oder symbolisch berechnet wird. Die numerische Berechnung der Jacobi-Matrix kann unter Umständen zwar weniger Aufwand mit sich bringen, senkt jedoch auch die Genauigkeit des Verfahrens. Ist es nun nach Wahl eines numerischen Integrationsverfahrens nicht möglich, ein mathematisches Modell in Echtzeit zu simulieren, weil die Berechnung eines Integrationsschrittes zu lange dauert, gibt es verschiedene Auswege. Die Schrittweite kann erhöht werden, um mehr Zeit für einen Integrationsschritt zur Verfügung zu haben. Hierdurch wird die berechnete Lösung natürlich ungenauer, da die einzige Iteration in diesem Fall zusätzlich mit einer ungenauen Matrix durchgeführt wird. Folglich wird in Kapitel 4 bei der Reduktion für Echtzeitsimulationen mit einem linear impliziten numerischen Integrationsverfahren stets die symbolische Jacobi-Matrix verwendet.

Alternativ kann man versuchen, das Integrationsverfahren noch effizienter zu gestalten. Ist die ODE steif, da zwei Subsysteme auf unterschiedlichen Zeitskalen operieren, also die Dynamik eines Subsystems viel schneller als die Dynamik des anderen Subsystems ist, so kann *multi-rate Integration* die Simulation beschleunigen. Die Idee dahinter ist, die beiden Subsysteme mit unterschiedlichen Schrittweiten zu integrieren, so dass lediglich das Subsystem mit der schnelleren Dynamik mit einer sehr kleinen Schrittweite gelöst wird. Details zu dieser Technik findet man in [Howe, 1998; Laffitte & Howe, 1997; Palusinski, 1985].

Der Gedanke der multi-rate Integration kann noch weiter getrieben werden. Lässt sich das Gesamtsystem in steife und nicht steife Subsysteme zerlegen, so ist es zielführend sogar verschiedene Integrationsverfahren zu verwenden: Beispielsweise linear-implizite Verfahren für die steifen Subsysteme und explizite Verfahren für die nicht steifen Subsysteme. In diesem Fall spricht man von *mixed-mode Integration* [Schiela & Olsson, 2000].

Eine weitere Technik, um Echtzeitsimulationen zu beschleunigen, ist die so genannte *inline Integration*. Bei dieser Methode verschwimmt die Grenze zwischen Modell und Integrationsverfahren. Hier wird die Verfahrensvorschrift (oder die Diskretisierung des Verfahrens) in die Modellgleichungen geschrieben. In der Folge wird das Tearing noch effizienter [Elmqvist *et al.*, 1995b].

Alle diese Techniken zielen auf das numerische Integrationsverfahren ab. Im Rahmen dieser Arbeit wird ein alternativer Ansatz vorgestellt. Anstatt das Integrationsverfahren effizienter zu gestalten, wird die Komplexität des mathematischen Modells

Rechenoperation	Gleitkommaoperationen
$\sin x, \cos x, \ldots$	23
$\log x$	20
$\exp x$	20
x^y	41
$\sqrt{x}$	15

Tabelle 3.1: Anzahl der benötigten Gleitkommaoperationen für einige Rechenoperationen [Minka, 2009]

reduziert. Im nächsten Abschnitt wird der Approximationsalgorithmus zur Reduktion für Echtzeitsimulationen vorgestellt.

3.5.2 Approximationsalgorithmus

Um ein Modell zu reduzieren, damit es in Echtzeit simuliert werden kann, muss sichergestellt werden, dass am Ende jeden Taktes ein Integrationsschritt abgeschlossen ist. An dieser Stelle kommt die Frage auf, wie garantiert werden kann, dass mit dem Ende eines jeden Echtzeit-Taktes ein Ergebnis zur Verfügung steht.

Die Leistungsfähigkeit von Rechnern kann in *Floating Point Operations per second* (FLOPS) angegeben werden. Die Anzahl der FLOPS gibt an, wie viele Gleitkommaoperationen pro Sekunde auf dem Rechner ausgeführt werden können. Als Gleitkommaoperation zählen hier Addition, Subtraktion sowie Multiplikation. Alle anderen Rechenoperationen werden im Rechner auf Basis dieser Grundoperationen ausgeführt. So benötigt die Berechnung einer Quadratwurzel in der Implementierung einer Standard-Bibliothek beispielsweise 15 Gleitkommaoperationen. Die Anzahl der benötigten Gleitkommaoperationen sind hier der Lightspeed Toolbox für Matlab entnommen. Tom Minka entwickelte diese Toolbox, um den Aufwand von Algorithmen plattformunabhängig zu messen. In Tabelle 3.1 sind die benötigten FLOPs für einige Rechenoperationen aufgelistet. In der Folge lässt sich natürlich berechnen, wie viele Gleitkommaoperationen innerhalb eines Echtzeit-Taktes ausgeführt werden können. Für einen Rechner mit einer Rechenleistung von $\mathcal{F}_T$ FLOPS ergibt sich für einen Echtzeit-Takt h_{RT} eine maximale Anzahl von

$$\mathcal{F}_T^{h_{RT}} = \mathcal{F}_T \cdot h_{RT} \tag{3.143}$$

FLOPs für einen Integrationsschritt. Ziel des Approximationsalgorithmus ist es also, das ursprüngliche Modell soweit zu reduzieren, dass der tatsächliche numerische Aufwand für einen Integrationsschritt kleiner wird als $\mathcal{F}_T^{h_{RT}}$. Während $\mathcal{F}_T$ unter Verwendung diverser Benchmarks leicht gemessen werden kann (beispielsweise mittels LINPACK [Dongarra *et al.*, 2003; Dongarra, 1979]), muss die tatsächlich benötigte Anzahl von FLOPs berechnet werden und hängt vom gewählten Lösungsverfahren sowie von dem Modell ab. Während das explizite Euler-Verfahren nur eine Funktionsauswertung benötigt, sind für das LI3-Verfahren drei Funktionsauswertungen, eine Auswertung der Jacobi-Matrix, die Lösung von vier linearen Gleichungssystemen sowie einige elementare Operationen notwendig. Sei $\mathcal{F}(g)$ der Operator, welcher die

benötigten Gleitkommaoperationen zur Auswertung eines Ausdrucks g zählt. Dann lässt sich die Anzahl der notwendigen Operationen für einen Integrationsschritt mit dem expliziten Euler-Verfahren schreiben als

$$\mathcal{F}_{\text{Euler}} = c_{\text{event}} \cdot \mathcal{F}(\mathbf{f}) + \mathcal{F}_{\text{event}}, \tag{3.144}$$

während sich die Anzahl der notwendigen Operationen für das LI3-Verfahren schreiben lässt als

$$\mathcal{F}_{\text{LI3}} = c_{\text{event}} \cdot (\mathcal{F}(\mathbf{f}) + \mathcal{F}(\frac{\partial \mathbf{f}}{\partial \mathbf{x}}) + \mathcal{F}_{\text{LSE}}) + \mathcal{F}_{\text{event}}, \tag{3.145}$$

wobei $\mathcal{F}_{\text{LSE}}$ die maximale Anzahl von FLOPs zur Lösung der vier linearen Gleichungssysteme bezeichnet. Weiterhin gilt

$$c_{\text{event}} = \begin{cases} 1, & \text{falls } \mathbf{f} \text{ unstetig} \\ 3, & \text{sonst} \end{cases} . \tag{3.146}$$

Denn im Fall einer Unstetigkeit von $\mathbf{f}$, müssen innerhalb eines Taktes nicht einer, sondern drei Integrationsschritte durchgeführt werden. Hier wird davon ausgegangen, dass im Rahmen einer Echtzeitsimulation keine dichte Ausgabe verfügbar ist. Andernfalls, würde $c_{\text{event}} = 2$ gelten. Darüber hinaus bezeichnet $\mathcal{F}_{\text{event}}$ die maximale Anzahl von FLOPs zum Reinitialisieren des numerischen Integrationsverfahrens nach einer Unstetigkeit (im Falle einer DAE).

Die Anzahl der Gleitkommaoperationen zur Lösung eines linearen Gleichungssystems ist natürlich abhängig vom Verfahren. Mit Hilfe der Gauss-Elimination werden beispielsweise

$$\mathcal{F}_{LSE} = \frac{1}{3}n^3 + \frac{3}{2}n^2 - \frac{5}{6}n \tag{3.147}$$

Gleitkommaoperationen benötigt, um ein lineares Gleichungssystem der Dimension n zu lösen. Bei dem LI3-Verfahren besitzen alle vier linearen Gleichungssysteme dieselbe Systemmatrix $\mathbf{L}^{-1} = (\mathbf{E} - \frac{h}{2}\mathbf{J})$. Daher ist es effizienter, die vier linearen Gleichungssysteme durch eine LU-Zerlegung ($\frac{2}{3}n^3$ FLOPs) sowie vier Vorwärts- und Rückwärtseliminationen (je n^2 FLOPs) zu lösen. Der Aufwand für alle vier linearen Gleichungssysteme beträgt dann lediglich

$$\mathcal{F}_{LSE} = \frac{2}{3}n^3 + 8n^2. \tag{3.148}$$

Ein Ansatz zur Reinitialisierung eines numerischen Integrationsverfahrens für DAEs ist die so genannte Event-Iteration [Lundvall *et al.*, 2006]. Diesem Ansatz folgend gilt

$$\mathcal{F}_{event} = 2^{n_{\text{zero}}} \cdot n_{\text{iter}}^{\text{event}}(\mathcal{F}(\mathbf{F}) + \mathcal{F}(\mathbf{J}_{\text{event}})), \tag{3.149}$$

wobei n_{zero} die Anzahl der Nullstellenfunktionen, $n_{\text{iter}}^{\text{event}}$ die maximale Anzahl von Newton-Iterationen während des Reinitialisierungsprozesses und $\mathbf{J}_{\text{event}}$ die Jacobi-Matrix bezüglich der Unbekannten in der Reinitialisierung bezeichnet. Die unbekannten Variablen sind hier zum einen die Variablen, welche nur algebraisch auftreten und zum anderen die Ableitungen derjenigen Variablen, welche auch differentiell auftreten.

Mit diesem Wissen kann $\mathcal{F}_{LI3}$ oder analog $\mathcal{F}_{\mathcal{N}}$ für ein beliebiges Lösungsverfahren $\mathcal{N}$ bestimmt und der Approximationsalgorithmus erweitert werden. Im Gegensatz zum ursprünglichen Approximationsalgorithmus wird nun keine Fehlertoleranz mehr vorgegeben. Stattdessen wird der Echtzeit-Takt h_{RT}, die Anzahl von FLOPS $\mathcal{F}_T$, welche die gewünschte Echtzeit-Hardware ausführen kann sowie das Lösungsverfahren $\mathcal{N}$ vorgegeben. Wie oben beschrieben kann hieraus die Anzahl der FLOPs $\mathcal{F}_T^{h_{RT}}$, welche die Echtzeit-Hardware in einem Echtzeit-Takt ausführen kann, leicht berechnet werden. Da das reduzierte Modell jedoch nicht nur in Echtzeit simuliert, sondern sich auch möglichst wenig von dem ursprünglichen Modell unterscheiden soll, wird hier zunächst auch eine Referenzsimulation durchgeführt sowie ein Ranking berechnet. Die Verwendung von Clustern ist an dieser Stelle jedoch offensichtlich nicht zielführend. Daher werden die Reduktionen entsprechend des Rankings, beginnend mit dem kleinsten Rankingwert, nacheinander durchgeführt und anstatt von Gleichung (3.55) wird getestet, ob die Bedingung

$$\mathcal{F}_T^{h_{RT}} < \mathcal{F}_{\mathcal{N}} \tag{3.150}$$

erfüllt ist. Der Algorithmus 2 zeigt das Vorgehen nochmals in Pseudocode.

Algorithmus 2 Approximationsalgorithmus für eine vorgegebene Echtzeitumgebung

Eingang: ODE $\mathbf{f}$, Echtzeit-Takt (h_{RT}), Rechenleistung ($\mathcal{F}_T$), Scenario ($\mathcal{S}$)
Ausgang: reduzierte ODE ($\mathbf{g}$)

```
[x*_out  x*_int]^T = N(f(x, ẋ, t), S)
g = f
L = R(F, K)
L = sort(L)
while L ≠ ∅ und F_T^{h_RT} < F_N do
    κ = L_1
    L = L \ L_1
    g ⇐ g^κ
end while
```

Berücksichtigung einer Fehlertoleranz

Bei dem obigen Vorgehen ist keine Fehlertoleranz vorgegeben. Folglich wird der Fehler auch nicht anhand von Simulationen kontrolliert. Ist das Ranking ungenau, kann es passieren, dass das Simulationsergebnis des reduzierten Modells schon nach wenigen Reduktionen stark von der Referenzlösung abweicht. Aufgrund dessen ist für dieses Vorgehen ein möglichst genaues Ranking essentiell. Eine andere Möglichkeit besteht darin, zusätzlich zu der Zielhardware auch eine Fehlertoleranz $\boldsymbol{\varepsilon}_{\text{Fehler}}$ anzugeben. In diesem Fall werden die Reduktionen ebenfalls ohne die Verwendung von Clustern durchgeführt, jedoch wird nach jeder Reduktion überprüft, ob Gleichung (3.55) erfüllt ist.

Die Kontrolle des Fehlers motiviert zusätzlich, das Ranking dahingehend zu modifizieren, dass die Anzahl der eingesparten FLOPs Einfluss auf den Rankingwert hat.

Algorithmus 3 Approximationsalgorithmus für Echtzeitsimulationen unter Berücksichtigung einer Fehlergrenze

Eingang: ODE ($\mathbf{f}$), Echtzeit-Takt (h_{RT}), Rechenleistung ($\mathcal{F}_T$), Fehlertoleranz (ε), Scenario ($\mathcal{S}$)
Ausgang: reduzierte ODE $\mathbf{g}$

$\begin{bmatrix}\mathbf{x}^\star_{\text{out}} & \mathbf{x}^\star_{\text{int}}\end{bmatrix}^T = \mathcal{N}(\mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}, t), \mathcal{S})$
$\mathbf{g} = \mathbf{f}$
$\mathcal{L} = \mathcal{R}(\mathbf{F}, \mathcal{K})$
$\mathcal{L} = \text{sort}(\mathcal{L})$
while $\mathcal{L} \neq \emptyset$ **und** $\mathcal{F}_T^{h_{RT}} < \mathcal{F}_\mathcal{N}$ **do**
  $\kappa = \mathcal{L}_1$
  $\mathbf{g} = \mathbf{g}^\kappa$
  $\begin{bmatrix}\mathbf{x}_{\mathbf{out}} & \mathbf{x}_{\mathbf{int}}\end{bmatrix}^T = \mathcal{N}(\mathbf{G}(\mathbf{x}, \dot{\mathbf{x}}, t), \mathcal{S})$
  $\varepsilon_i = \left\| x^\star_{\mathbf{out},i} - x_{\mathbf{out},i} \right\|,\ 1 \leq i \leq N_{\mathbf{out}}$
  if $\exists\, i : \varepsilon_i \geq \varepsilon_{\mathbf{Fehler},i}$ **then**
    $\mathbf{g} = \mathbf{g}^{\kappa^{-1}}$
  end if
end while

So kann es beispielsweise vorteilhafter sein, eine Reduktion durchzuführen, die viele FLOPs einspart und einen vergleichsweise hohen Rankingwert aufweist, als viele Reduktionen durchzuführen die zwar alle einen niedrigen Rankingwert aufweisen, dafür aber kaum FLOPs einsparen.

Ist r^κ der Rankingwert für eine Reduktion $\kappa \in \mathcal{K}$ und τ der zugehörige Term, so kann man beispielsweise durch

$$\widetilde{r}^\kappa = \frac{r^\kappa}{\mathcal{F}(\tau)} \tag{3.151}$$

einen neuen Rankingwert $\widetilde{r}^\kappa$ berechnen, welcher die Anzahl der gesparten Operationen berücksichtigt.

Führt man die Aktualisierung auf der gesamten Rankingliste aus, kann die Laufzeit des Algorithmus gesenkt werden. Algorithmus 3 zeigt den entsprechenden Algorithmus in Pseudocode.

Beispiel 3.5.1. Würde an dieser Stelle wieder der Zweimassenschwinger als Beispiel dienen, so würde je nach verfügbarer Rechenleistung und Fehlertoleranz wieder die Dämpfung und/oder die Masse m_{M_1} vernachlässigt werden. Daher wird der Approximationsalgorithmus hier auf das einfache hydraulische Beispiel aus den Gleichungen (3.4) bis (3.11) und dem Objektdiagramm aus Abbildung 3.1 angewendet.

Zwar macht die Reduktion für Echtzeitsimulationen eigentlich nur für komplexe Systeme Sinn. Jedoch kann schon dieses einfache System nicht ohne weiteres in Echtzeit simuliert werden, da das Modell sehr steif ist. Die Steifigkeit rührt von dem Unterschied in der Dynamik der Mechanik und der Hydraulik [Mikelsons *et al.*, 2010b]. Aufgrund dieser Steifigkeit ist es nicht möglich, explizite Verfahren zu verwenden.

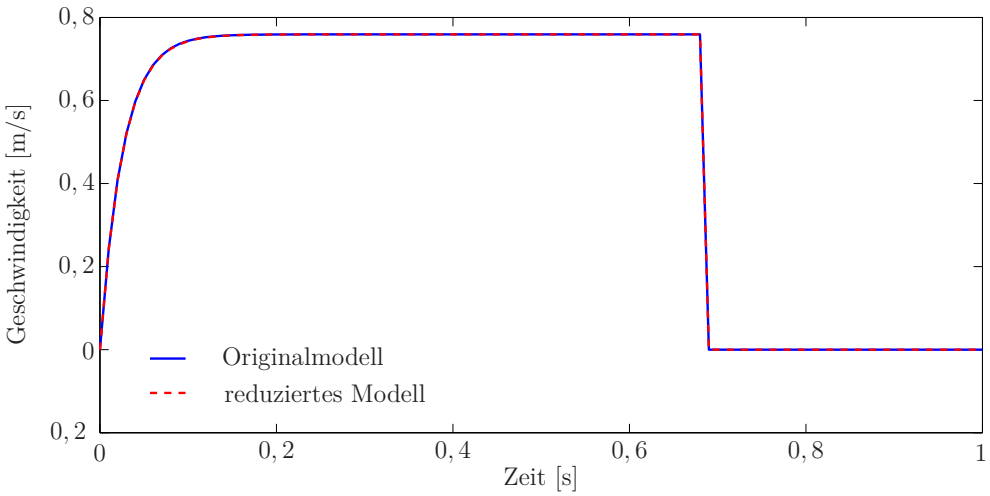


Abbildung 3.13: Vergleich der Referenzlösung mit der Lösung des reduzierten Modells

Linear-implizite Verfahren können angewendet werden, verlangen jedoch nach einer relativ kleinen Schrittweite.

Wendet man den Approximationsalgorithmus auf dieses Modell an, so werden zum einen die Anzahl der benötigten FLOPs für einen Integrationsschritt gesenkt und zum anderen ist das reduzierte Modell nicht mehr steif, so dass ein explizites Integrationsverfahren verwendet werden kann.

In dem verwendeten Szenario wird der Kolben bis an den Anschlag gefahren und verweilt dort. Die Simulationszeit beträgt dabei 1s. Im Gegensatz zu den bisherigen Beispielen wird hier sowohl die Termreduktion als auch die Termsubstitution eingesetzt. Die Auswertung von $\mathbf{f}$ benötigt 536 FLOPs, während die Auswertung der Jacobi-Matrix $\mathbf{J}$ 844 FLOPs benötigt. Zur Lösung der drei linearen Gleichungssysteme sind nach Gleichung (3.148) 141 FLOPs notwendig. Die Anzahl der maximal benötigten FLOPs für einen Integrationsschritt beträgt hier also

$$\mathcal{F}_{LI3} = 4146. \tag{3.152}$$

Für die Reduktion wird eine Fehlertoleranz von $0,002\frac{\mathrm{m}}{\mathrm{s}}$ für die Geschwindigkeit v des Kolbens sowie eine Echtzeitumgebung mit maximal 400 FLOPs für einen Integrationsschritt vorgegeben. Abbildung 3.13 zeigt die Geschwindigkeit des Kolbens für das ursprüngliche und das reduzierte Modell. Das reduzierte Modell lässt sich dabei schreiben als

$$v = C_2, \tag{3.153}$$
$$\dot{s} = v, \tag{3.154}$$
$$\dot{x} = \frac{1}{m}(\frac{\pi}{4} p \cdot d_P^2 + m \cdot g), \tag{3.155}$$
$$1,02 \cdot 10^7 = (\frac{\pi}{4} C_2 \cdot d_P^2 + (p - p_S) \cdot G) \cdot K/V_A, \tag{3.156}$$

mit

$$K = \beta_0 \beta_1, \tag{3.157}$$

$$C_2 = \begin{cases} 0, & \text{falls } s - l \leq 0 \\ x, & \text{sonst} \end{cases} . \tag{3.158}$$

Die ODE für die Hydraulik ist also verschwunden, da $\dot{p}$ durch einen konstanten Wert ersetzt wurde. In der Folge wird der Druck p aus der algebraischen Gleichung (3.156) bestimmt. Das Modell kann also noch weiter vereinfacht werden, indem p in Gleichung (3.156) isoliert und in Gleichung (3.155) eingesetzt wird. Dieses Vorgehen liefert die Gleichung

$$\dot{x} = \frac{1}{m}(\frac{\pi}{4}(p_s + 1,02 \cdot 10^7 \frac{V_A}{K \cdot G} - \frac{\pi}{4} C_2 \frac{d_P^2}{G}) \cdot d_P^2 + m \cdot g). \tag{3.159}$$

Damit entfällt der steife Anteil der ursprünglichen ODE. Das reduzierte Modell benötigt nur noch 360 FLOPs für einen Integrationsschritt mit dem LI3-Verfahren und könnte darüber hinaus nun auch mit einem expliziten Verfahren gelöst werden.

3.6 Szenarienwahl

Im Gegensatz zu vielen anderen Ansätzen in der Modellreduktion ist für die gleichungsbasierte Modellreduktion ein Szenario erforderlich. Dies ist zum einen ein Nachteil, da das reduzierte Modell die vorgegebene Fehlertoleranz nur in diesem Szenario einhält. Für andere Szenarien ist es im Allgemeinen schwer, Voraussagen über das Verhalten des Modells zu treffen.

Auf der anderen Seite ist die Vorgabe von Szenarien natürlich ein Vorteil. So können für verschiedene Zwecke verschiedene Modelle generiert werden. Diese spezialisierten Modelle sind für ihr Szenario natürlich hoch effizient. Zusätzlich ist es im Rahmen dieser Arbeit gelungen, auch die Vorgabe mehrerer oder intervallwertiger Szenarien zu ermöglichen.

Dennoch stellt sich natürlich die Frage nach geeigneten Szenarien für die Reduktion. Im Allgemeinen ist diese Frage nur sehr schwer zu beantworten. Deshalb sollen an dieser Stelle einige Orientierungspunkte gegeben werden.

Eine mögliche Wahl für ein Szenario ist das so genannte *worst-case*-Szenario. Dieses Szenario fordert also einen höheren Detaillierungsgrad als diejenigen Szenarien, welche bei der späteren Verwendung des reduzierten Modells tatsächlich zum Einsatz kommen. Soll zum Beispiel ein Modell für die Längsdynamik einer Baumaschine durch gleichungsbasierte Modellreduktion gewonnen werden, so stellt ein hochdynamisches Start-Stop Manöver den worst-case dar und ist somit als Szenario geeignet. Diese Art von Szenario wurde beispielsweise in [Mikelsons *et al.*, 2009a] gewählt.

Ein weiteres für die Reduktion geeignetes Szenario ist ein für dieses Modell typisches Szenario, von dem die tatsächlich auftretenden Szenarien nicht allzu weit abweichen. Soll beispielsweise ein Fahrzeugmodell für Fahrten auf der Landstraße aus

einem komplexen Fahrzeugmodell erzeugt werden, so stellt ein Doppelspurwechsel bei 80km/h ein typisches Szenario dar. Alle Manöver, die bei einer normalen Landstraßenfahrt (ohne Manöver im fahrdynamischen Grenzbereich) auftreten, weisen eine ähnliche Dynamik wie der doppelte Spurwechsel auf. Mit Hilfe dieser Strategie ist es auch möglich, konventionelle Modelle mit Hilfe der gleichungsbasierten Modellreduktion zu generieren.

Werden in dem mathematischen Modell stückweise definierte Funktionen verwendet, sollte beachtet werden, dass in dem reduzierten Modell nur diejenigen Zweige erhalten bleiben, welche für die Simulation des vorgegebenen Szenarios notwendig sind. Durch den Wegfall der anderen Modellzweige können zwar hocheffiziente Modelle entstehen, diese können dann jedoch auch nur für bestimmte Szenarien verwendet werden.

Ein wichtiger Bestandteil eines Szenarios sind die Ausgangsvariablen. So macht es in der Regel einen großen Unterschied, welche Variablen als Ausgangsvariablen gewählt werden. Beispielsweise ist in der Fahrdynamik die Position des Fahrzeugs viel weniger empfindlich gegenüber Modelländerungen als die Geschwindigkeit oder Beschleunigung. Jedoch besitzen Beschleunigung und Geschwindigkeit in vielen Fällen die größere Aussagekraft.

Im folgenden Kapitel wird dies anhand von Beispielen aus der Fahrzeugdynamik gezeigt. Darüber hinaus wird die Anwendung der gleichungsbasierten Modellreduktion auf komplexe mechatronische Systeme präsentiert und erläutert, warum die Reduktionsmethode einen ersten Schritt auf dem Weg zu generischen Modellen darstellt.

KAPITEL 4

Anwendungsbeispiele

Im vorherigen Kapitel wurde die gleichungsbasierte Modellreduktion eingeführt. Die präsentierten Beispiele zeigten jedoch lediglich die Funktionsweise der Reduktion. Um die Effektivität sowie Praxistauglichkeit des Verfahrens zu demonstrieren, werden in diesem Kapitel Beispiele aus dem Baumaschinensektor sowie aus der Fahrzeugdynamik herangezogen. Die verwendete Implementierung wird in Abschnitt 4.1 beschrieben. In Abschnitt 4.4 wird darüber hinaus ein Weg aufgezeigt, um generische Modelle dynamischer Systeme zu erstellen. Hierbei bedeutet generisch, dass Modelle nicht nur generisch bezüglich ihres Detaillierungsgrades, sondern auch generisch bezüglich ihrer Dimension sind. Dieser Ansatz wird anhand eines nichtlinearen Modells aus der Fahrzeugdynamik veranschaulicht.

4.1 Implementierung

Das im vorherigen Kapitel vorgestellte Verfahren zur gleichungsbasierten Modellreduktion wurde zunächst in MATLAB implementiert, um die Leistungsfähigkeit des Verfahrens und der Erweiterungen zu untersuchen. Hierbei wurde die Maple-Toolbox für MATLAB zum Verarbeiten der symbolischen Variablen verwendet. Für die Implementierung des Algorithmus ist jedoch nicht nur die Verarbeitung von symbolischen Variablen, sondern auch die Verarbeitung und Manipulation von symbolischen Gleichungen notwendig. Daher wurde im Rahmen dieser Arbeit eine Möglichkeit geschaffen, symbolische Gleichungen und DAEs in MATLAB zu repräsentieren. Im Anhang findet man hierzu eine ausführliche Beschreibung.

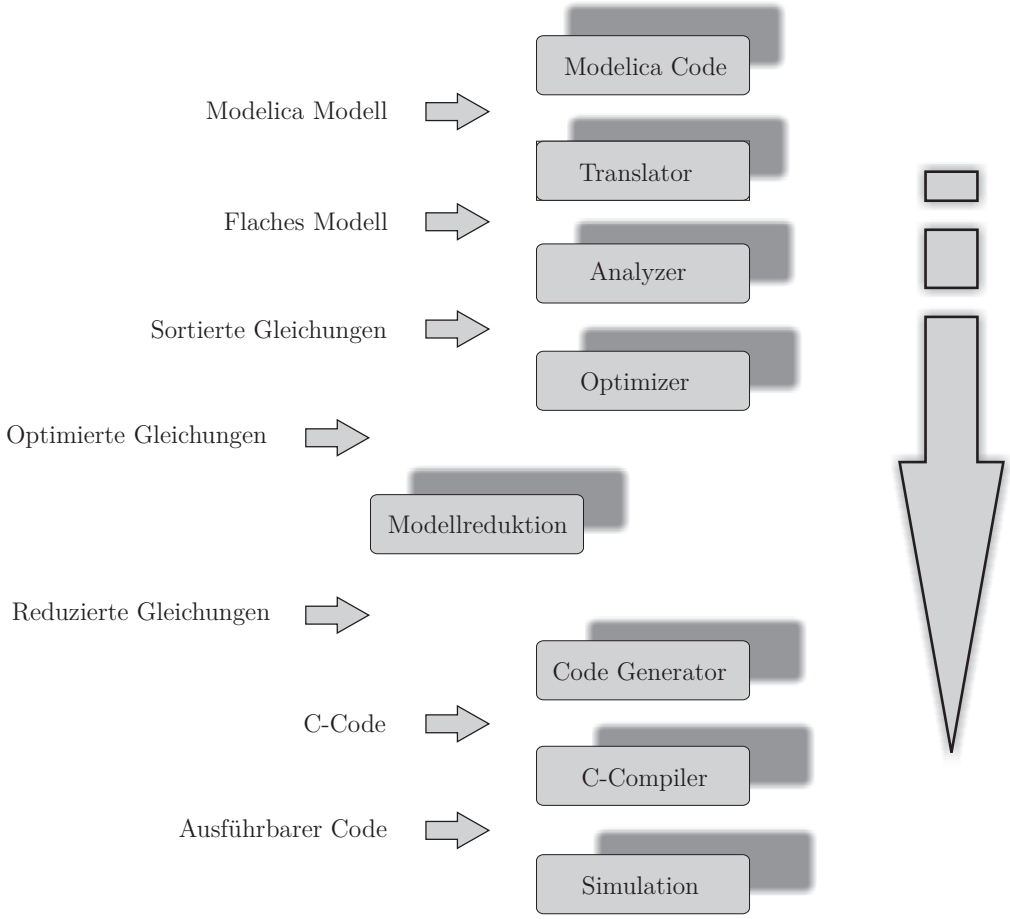


Abbildung 4.1: Struktur des OpenModelica Compilers

Der für die Simulationen notwendige Code wird automatisiert unter Verwendung der in Abschnitt 2.1.2 gezeigten Methoden erzeugt. Darüber hinaus wird kein MATLAB, sondern ein C-Code erzeugt, um die Funktionsauswertungen zu beschleunigen. Der entsprechende C-Code wird zusammen mit vorgefertigten C-Dateien, die die Schnittstelle zu MATLAB bilden (`mex` Files), kompiliert und ist so von MATLAB aus zugänglich. Die Erzeugung des C-Codes basiert auf dem *Code Generation Package* von Maple [Wittkopf, 2008].

Zur Lösung der entstehenden DAEs können verschiedene Lösungsverfahren verwendet werden. Als Standardverfahren wird IDAS aus dem SUNDIALS Softwarepaket verwendet [Hindmarsh *et al.*, 2005]. Diese Methode ist eine C-Implementierung von DASSL mit Nullstellensuche und der Möglichkeit einer Sensitivitätsanalyse[1]. Darüber hinaus können sowohl ein BDF-Verfahren mit fester Ordnung und Schrittweite als auch die in Abschnitt 3.5.1 beschriebenen numerischen Integrationsverfahren gewählt werden. Die für die Integration notwendigen Jacobi-Matrizen können mit den in Abschnitt 2.2.3 beschriebenen Vor- und Nachteilen symbolisch oder numerisch erzeugt werden. Das implementierte Modellreduktionsverfahren erlaubt neben der Vorgabe und entsprechender Verarbeitung von Nullstellenfunktionen auch die Verwendung von Kennfeldern und gesampelten Signalen. Es können also Variablen ange-

[1]Die entsprechende Modifikation von DASSL heißt DASPK.

geben werden, die jeweils nur nach Ablauf einer vorgegebenen Zeitspanne aktualisiert werden. Diese Technik ist beispielsweise notwendig, um Steuergeräte realitätsgetreu zu modellieren.

Zur Erleichterung des Imports von existierenden Modellen existiert eine MathML-Schnittstelle [Sandhu, 2003]. So kann die Modellierung beispielsweise in der Modelica basierten Simulationsumgebung MathModelica geschehen. Das fertige Modell kann dann über Mathematica und die MathML-Schnittstelle in das Modellreduktionsverfahren importiert und anschließend reduziert werden.

Neben der Implementierung in MATLAB wurde im Rahmen der Arbeit auch ein Plug-In für den OpenModelica-Compiler geschaffen. In Abbildung 4.1 ist die strukturelle Einbindung des Plug-Ins in den OpenModelica-Compiler zu sehen. Im Gegensatz zu dem Plug-In ist der OpenModelica-Compiler in der Programmiersprache MetaModelica geschrieben [Pop & Fritzson, 2006]. Das Plug-In ist also nicht direkt in den OpenModelica-Compiler integriert, sondern verwendet diesen lediglich für die symbolischen Manipulationen, zur Erzeugung des Codes für die Simulationen sowie zur Durchführung der Simulationen. Das Plug-In bietet zur Zeit noch nicht alle Funktionen der MATLAB-Implementierung, jedoch ist die volle Funktionalität geplant.

4.2 Anwendung auf Beispiele aus der Fahrzeugdynamik

Um die Wirkungsweise zu verdeutlichen und die Effektivität der im vorherigen Abschnitt präsentierten MATLAB-Implementierung zu demonstrieren, wird nun die Anwendung des Modellreduktionsverfahrens auf Modelle aus der Fahrzeugdynamik beschrieben. Im nächsten Abschnitt wird zunächst ein kurzer Überblick über die verwendeten Fahrzeugmodelle gegeben, um die vorgenommenen Reduktionen später interpretieren zu können. Im Anschluss werden Fahrzeugmodelle sowohl für eine vorgegebene Fehlertoleranz als auch für Echtzeitsimulationen reduziert. Darüber hinaus wird auch die Vorgabe von intervallwertigen Szenarien getestet.

4.2.1 Verwendete Fahrzeugmodelle

In diesem Abschnitt werden nur die in dieser Arbeit verwendeten Modelle kurz erläutert. Eine ausführliche Beschreibung zur Modellierung und Simulation von Kraftfahrzeugen findet man beispielsweise in [Schramm *et al.*, 2010] oder [Pacejka, 2006]. An dieser Stelle werden zunächst ein räumliches Zweispurmodell und anschließend zwei vereinfachte Modelle präsentiert.

Räumliches Zweispurmodell

In Abbildung 4.2 ist das Schema eines räumlichen Zweispurmodells gezeigt. Das hier verwendete räumliche Zweispurmodell berücksichtigt alle sechs Freiheitsgrade

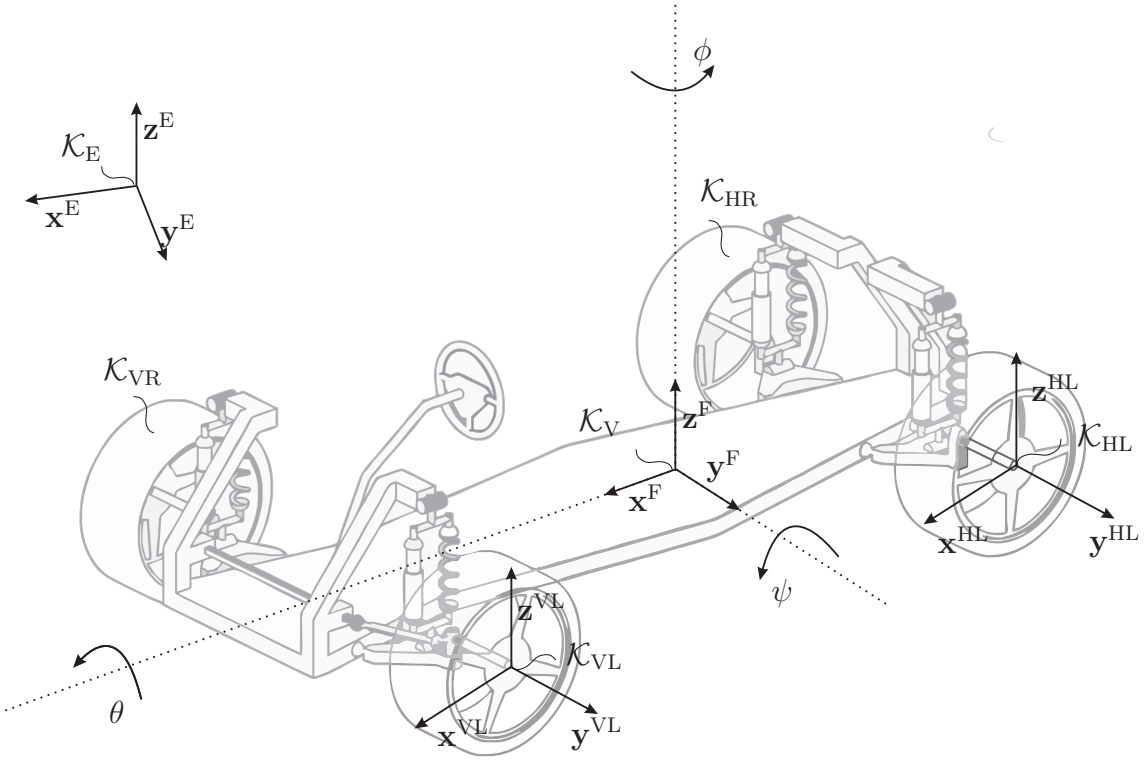


Abbildung 4.2: Räumliches Zweispurmodell [Schramm *et al.*, 2010]

des Chassis. Hierbei sind die drei rotatorischen Freiheitsgrade gegeben durch das Wanken (Rotation um die $x-$Achse um den Winkel θ), das Nicken (Rotation um die $y-$Achse um den Winkel ϕ) und das Gieren (Rotation um die $z-$Achse um den Winkel ψ). Darüber hinaus verfügt das Modell über vier unabhängige Räder. Jedes Rad hat einen rotatorischen und einen translatorischen Freiheitsgrad. Während der rotatorische Freiheitsgrad die Drehung des Reifens repräsentiert, beschreibt der translatorische Freiheitsgrad die Vertikalbewegung des Reifens. Dabei sind die Räder über Feder-Dämpfer-Elemente mit dem Chassis verbunden. Zusätzlich sind hier Stabilisatoren modelliert. Im Gegensatz zu dem Schema in Abbildung 4.2 können sich die Räder also relativ zum Chassis lediglich in vertikaler Richtung bewegen. Insgesamt verfügt das Modell über vierzehn Freiheitsgrade. In der Folge sind Chassis und Räder natürlich mit entsprechenden Massen und Trägheiten behaftet. Lenkwinkel und Antriebsmoment stellen die Systemeingänge dar. Dabei wirkt das Antriebsmoment nur auf die Vorderachse und wird hier gleichmäßig auf die beiden Räder verteilt.

Bewegungsgleichungen des Chassis Der Impulsatz für das Chassis liefert

$$m_{\mathrm{F}}\ddot{\mathbf{r}}_{\mathrm{F}}^{\mathrm{E}} = \sum_{i\in\{\mathrm{VL,VR,HL,HR}\}} \sum_{j\in\{k,s\}} \mathbf{F}_{i,j}^{\mathrm{E}}, \tag{4.1}$$

wobei $\mathbf{F}_{i,k}^{\mathrm{E}}$ die Kräfte der Radaufhängung und $\mathbf{F}_{i,s}^{\mathrm{E}}$ die Reifenkräfte bezeichnet. Der Superindex gibt hierbei an, in welchem Koordinatensystem der entsprechende Vektor referenziert ist. Weiterhin bezeichnet m_{F} die Masse des Fahrzeugs. Während an dieser Stelle nicht weiter auf die Bestimmung der Radaufhängungskräfte und

Radaufstandskräfte eingegangen wird, wird im nächsten Abschnitt ein Modell zur Bestimmung der Reifenkräfte präsentiert. Analog zum Impulssatz kann auch der Drallsatz formuliert werden, um die Orientierung des Fahrzeugs zu bestimmen.

Nichtlineares Reifenmodell Die stationären Reifenkräfte werden mit Hilfe einer vereinfachten Version des *magic-formula*-Reifenmodells berechnet [Bakker *et al.*, 1987]. Das magic formula Reifenmodell ist kein physikalisches Modell, sondern basiert auf der Approximation von Messdaten. Dabei werden die Reifenkräfte abhängig vom Schlupf s_i, des Schräglaufwinkels α_i, der Radaufstandskraft $F_{z_i}^i$ und der Reibungskoeffizienten μ_i berechnet durch

$$\begin{bmatrix} F_{i,x,\text{stat}} \\ F_{i,y,\text{stat}} \end{bmatrix} = \begin{bmatrix} \frac{s_i}{s_{n,i}} \cdot F_{i,\text{gesamt}} \\ \frac{\tan(\alpha_i)}{s_{n,i}} \cdot F_{i,\text{gesamt}} \end{bmatrix}, \quad i \in \{\text{VL}, \text{VR}, \text{HL}, \text{HR}\}. \tag{4.2}$$

Dabei ist

$$F_{i,\text{gesamt}} = \sqrt{\frac{\tan^2(\alpha_i)}{s_{n,i}^2} \cdot F_{i,y,\text{ref}}^2 + \frac{s_i^2}{s_{n,i}^2} F_{i,x,\text{ref}}^2}, \tag{4.3}$$

$$F_{i,x,\text{ref}} = \mu_x \sin(c_x \cdot \text{atan}(100 \cdot b_x s_{n,i})) \cdot F_{i,z}, \tag{4.4}$$

$$F_{i,y,\text{ref}} = \mu_y \sin(c_y \cdot \text{atan}(\frac{180}{\pi} b_y \cdot \text{atan}(s_{n,i}))) \cdot F_{i,z}, \tag{4.5}$$

$$s_{n,i} = \sqrt{\tan^2(\alpha_i) + s_i^2}. \tag{4.6}$$

Hier bezeichnen $\mu_x, c_x, b_x, \mu_y, c_y$ und b_y die beispielsweise durch Messungen gewonnenen Reifenparameter. Die dynamischen Reifenkräfte berücksichtigen zusätzlich die Verformung der Reifen und werden durch

$$\dot{\mathbf{F}}_{i,s} = \begin{bmatrix} \dot{F}_{i,x}^i \\ \dot{F}_{i,y}^i \end{bmatrix} = \begin{bmatrix} \frac{|r\omega_i^w|}{S_{x_i}}(F_{i,x,\text{stat}} - F_{x_i}) \\ \frac{|r\omega_i^w|}{S_{y_i}}(F_{i,y,\text{stat}} - F_{y_i}) \end{bmatrix} \tag{4.7}$$

berechnet, wobei S_{x_i} und S_{y_i} geignete Zeitkonstanten sind, r_R den Radius und $\dot{\boldsymbol{\rho}}_{\text{R},i}$ die Winkelgeschwindigkeit des Reifens i bezeichnet.

Bewegungsgleichungen der Reifen Jeder Reifen besitzt einen translatorischen und einen rotatorischen Freiheitsgrad. Der translatorische Freiheitsgrad beschreibt die Kompression der Aufhängung in vertikaler Richtung und der rotatorische Freiheitsgrad beschreibt die Drehung des Reifens. Der Impulssatz liefert somit

$$m_\text{R} \ddot{\mathbf{z}}_{\text{R}_i}^\text{E} = \mathbf{F}_{i,z}^\text{E} - \mathbf{F}_\text{Auf}^\text{E} - m_\text{R}\mathbf{g} \quad i \in \{VL, VR, HL, HR\}, \tag{4.8}$$

wobei $\mathbf{F}_{i,z}^\text{E}$ die Radnormalenaufstandskraft und $\mathbf{F}_\text{Auf}^\text{E}$ die von der Aufhängung auf den Reifen wirkende Kraft beschreibt. Darüber hinaus ist m_R die Masse eines Reifens und $\mathbf{g}$ die Erdbeschleunigung.

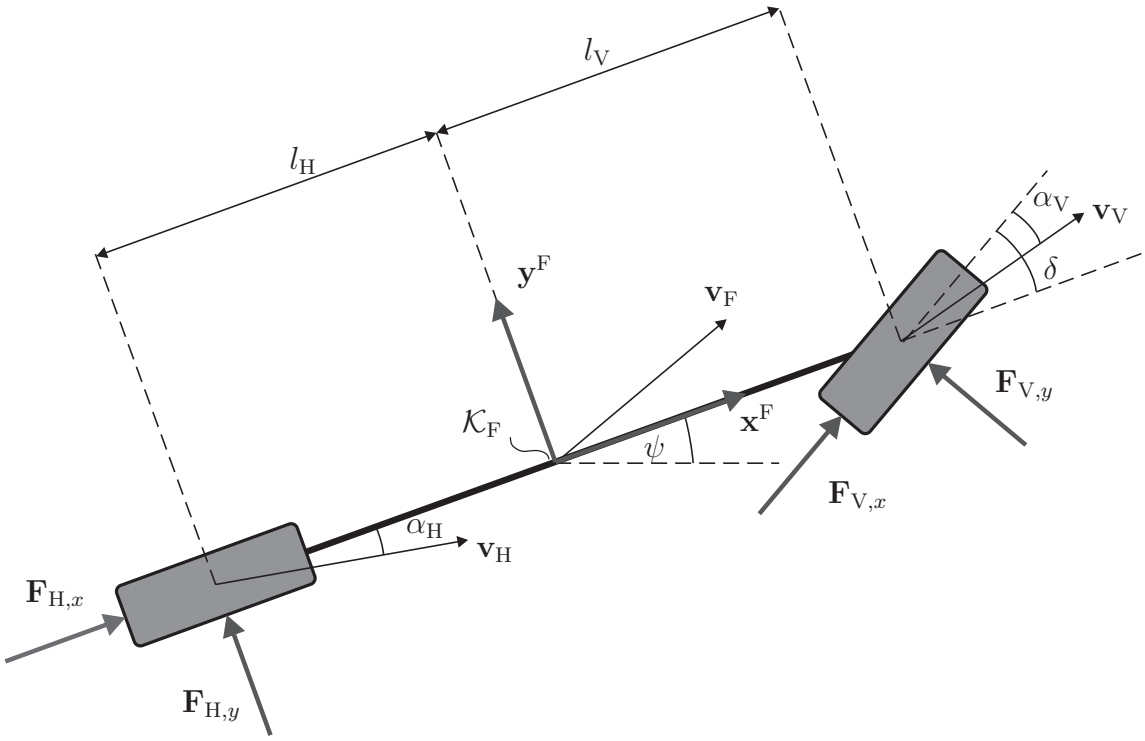


Abbildung 4.3: Einspurmodell

Die rotatorische Bewegung des Reifens kann mit Hilfe des Drallsatzes geschrieben werden als

$$\boldsymbol{\Theta}_\mathrm{R} \cdot \ddot{\boldsymbol{\rho}}^i_{\mathrm{R},i} = \mathbf{M}^\mathrm{A}_i - r_\mathrm{R}\mathbf{F}^i_{i,x}\; i \in \{VL, VR, HL, HR\}\,, \tag{4.9}$$

wobei $\mathbf{M}^A_i$ das Antriebsmoment am $i-$ten Reifen, $\boldsymbol{\Theta}_\mathrm{R}$ die Trägheit des $i-$ten Reifens und $\mathbf{F}^i_{i,x}$ die Reifenkraft in Umlaufrichtung ist.

Nichtlineares Einspurmodell

Das im vorherigen Abschnitt beschriebene räumliche Zweispurmodell beschreibt die Dynamik eines Kraftfahrzeugs für viele Anwendungen zu detailliert. Daher existieren auch einfachere Modelle. Die Annahme von Symmetrie entlang der Längsachse des Fahrzeugs führt auf das nichtlineare Einspurmodell. Das Modell hat dann natürlich nur noch zwei unabhängige Reifen. Zusätzlich verschwindet der rotatorische Freiheitsgrad, der das Wanken des Fahrzeugs beschreibt. In der Regel wird darüber hinaus die Vertikaldynamik und das Nicken vernachlässigt. Es verbleiben also fünf Freiheitsgrade. Drei Freiheitsgrade beschreiben die Bewegungen des Chassis in der Ebene und die beiden verbleibenden Freiheitsgrade beschreiben die Drehung der Reifen. In Abbildung 4.3 ist das entsprechende Modell skizziert. Benötigt werden außerdem die Parameter l_z und Θ_z, die den Abstand des Fahrzeugschwerpunkts vom Boden sowie das Trägheitsmoment des Fahrzeugs bezüglich der vertikalen Achse bezeichnen. Das mathematische Modell kann dann relativ kompakt notiert werden. Dabei sind die Reifenkräfte in den Radkoordinatensystemen $\mathcal{K}_\mathrm{V}$ und $\mathcal{K}_\mathrm{H}$ angegeben, wobei der Index V das vordere Rad und der Index H das hintere Rad bezeichnet.

Die verbleibenden Größen sind im Inertialsystem angegeben. Auf die entsprechende Kenntlichmachung wird hier verzichtet. Die entsprechende ODE lautet dann

$$\begin{bmatrix} \ddot{x}_\mathrm{F} \\ \ddot{y}_\mathrm{F} \\ \ddot{\psi} \\ \ddot{\rho}_\mathrm{V} \\ \ddot{\rho}_\mathrm{H} \\ \dot{F}_{\mathrm{V},x} \\ \dot{F}_{\mathrm{V},y} \\ \dot{F}_{\mathrm{H},x} \\ \dot{F}_{\mathrm{H},y} \end{bmatrix} = \begin{bmatrix} \frac{1}{m_\mathrm{F}}(\cos(\psi+\delta)F_{\mathrm{V},x} + F_{\mathrm{H},x} - \sin(\psi+\delta)F_{\mathrm{V},y} - F_{\mathrm{H},y}) \\ \frac{1}{m_\mathrm{F}}(\sin(\psi+\delta)F_{\mathrm{V},x} + F_{\mathrm{H},x} + \cos(\psi+\delta)F_{\mathrm{V},y} + F_{\mathrm{H},y}) \\ \frac{1}{\Theta}{}_z(l_\mathrm{V}(\sin(\delta)F_{\mathrm{V},x} + \cos(\delta)F_{\mathrm{V},y}) - l_H F_{\mathrm{H},y}) \\ \frac{1}{\Theta_\mathrm{R}}(M_A - r_\mathrm{R}F_{\mathrm{V},x}) \\ -\frac{1}{\Theta_\mathrm{R}}r_\mathrm{R}F_{\mathrm{H},x} \\ \frac{|r\dot{\rho}_\mathrm{V}|}{\sigma_x}(F_{\mathrm{V},x,\mathrm{stat}} - F_{\mathrm{V},x}) \\ \frac{|r\dot{\rho}_\mathrm{V}|}{\sigma_y}(F_{\mathrm{V},y,\mathrm{stat}} - F_{\mathrm{V},y}) \\ \frac{|r\dot{\rho}_\mathrm{H}|}{\sigma_x}(F_{\mathrm{H},x,\mathrm{stat}} - F_{\mathrm{H},x}) \\ \frac{|r\dot{\rho}_\mathrm{H}|}{\sigma_y}(F_{\mathrm{H},y,\mathrm{stat}} - F_{\mathrm{H},y}) \end{bmatrix}, \tag{4.10}$$

wobei $\dot{\rho}_\mathrm{V}$ und $\dot{\rho}_\mathrm{H}$ die Umdrehungsgeschwindigkeit der Räder beschreibt. Die stationären Reifenkräfte werden mit Hilfe der magic formula wie im vorherigen Abschnitt in den Gleichungen (4.2) bis (4.7) berechnet. Der hierzu benötigte Schlupf s_i beziehungsweise $s_{n,i}$ und der Schräglaufwinkel α_i können dabei durch

$$s_i = \frac{r\dot{\rho}_i - v_{i,x}}{\max(|r\dot{\rho}_i|, |v_{i,x}|)}, \qquad \alpha_i = -\arctan(\frac{v_{i,y}}{|r\dot{\rho}_i|}) \tag{4.11}$$

und

$$s_{n,i} = \sqrt{\tan^2(\alpha_i) + s_i^2} \tag{4.12}$$

für $i \in \{\mathrm{V}, \mathrm{H}\}$ berechnet werden. Dabei ergeben sich die Radaufstandskräfte zu

$$F_{\mathrm{V},z} = \frac{l_\mathrm{H}}{l} m_\mathrm{F} g - \frac{l_z}{l}(\cos(\delta)F_{\mathrm{V},x} - \sin(\delta)F_{\mathrm{V},y} + F_{\mathrm{H},x}) \tag{4.13}$$

$$F_{\mathrm{H},z} = \frac{l_\mathrm{V}}{l} m_\mathrm{F} g + \frac{l_z}{l}(\cos(\delta)F_{\mathrm{V},x} - \sin(\delta)F_{\mathrm{V},y} + F_{\mathrm{H},x}), \tag{4.14}$$

während die Geschwindigkeiten in den Koordinatensystemen der Räder durch

$$v_{\mathrm{V},x} = \cos(\psi+\delta)(\dot{x}_\mathrm{F} - l_\mathrm{V}\dot{\psi}\sin\psi) + \sin(\psi+\delta)(\dot{y}_\mathrm{F} + l_\mathrm{V}\dot{\psi}\cos\psi), \tag{4.15}$$

$$v_{\mathrm{V},y} = -\sin(\psi+\delta)(\dot{x}_\mathrm{F} - l_\mathrm{V}\dot{\psi}\sin\psi) + \cos(\psi+\delta)(\dot{y}_\mathrm{F} + l_\mathrm{V}\dot{\psi}\cos\psi), \tag{4.16}$$

$$v_{\mathrm{H},x} = \cos\psi(\dot{x}_\mathrm{F} + l_\mathrm{H}\dot{\psi}\sin\psi) + \sin\psi(\dot{y}_\mathrm{F} - l_\mathrm{H}\dot{\psi}\cos\psi), \tag{4.17}$$

$$v_{\mathrm{H},y} = -\sin\psi(\dot{x}_\mathrm{F} - l_\mathrm{H}\dot{\psi}\sin\psi) + \cos\psi(\dot{y}_\mathrm{F} - l_\mathrm{H}\dot{\psi}\cos\psi) \tag{4.18}$$

berechnet werden können.

Lineares Einspurmodell

Bereits in [Riekert & Schunck, 1940] wurde das lineare Einspurmodell eingeführt. Es wird noch heute verwendet, um fundamentale Effekte der Fahrzeugdynamik zu erklären. Das Modell von Riekert und Schunk basiert auf dem nichtlinearen Einspurmodell, verwendet aber zusätzlich noch die Annahmen, dass

- der Lenkwinkel δ des Fahrzeugs klein ist, so dass die Näherungen $\sin\delta \approx \delta$ und $\cos\delta \approx 1$ verwendet werden können,
- der Gierwinkel ψ des Fahrzeugs ψ klein ist, so dass die Näherungen $\sin\psi \approx \psi$ und $\cos\psi \approx 1$ verwendet werden können,
- die Reifenkräfte linear von der Giergeschwindigkeit $\dot{\psi}$ und der Quergeschwindigkeit $\dot{x}_y = v_y$ abhängen,
- die Längsgeschwindigkeit $v_x = \dot{x}_{\mathrm{F}}^{\mathrm{E}}$ annähernd konstant ist.

Mit diesen Annahmen lässt sich das lineare Einspurmodell in der linearen Zustandsform schreiben als

$$\begin{bmatrix} \dot{v}_y \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} -\frac{C_\alpha^{\mathrm{H}} + C_\alpha^{\mathrm{V}}}{m_{\mathrm{F}} \cdot v_x} & \left(\frac{C_\alpha^{\mathrm{H}} \cdot l_{\mathrm{H}} - C_\alpha^{\mathrm{V}} \cdot l_{\mathrm{V}}}{m_{\mathrm{F}} \cdot v_x} - v_x\right) \\ \frac{C_\alpha^{\mathrm{H}} \cdot l_{\mathrm{H}} + C_\alpha^{\mathrm{V}} \cdot l_{\mathrm{V}}}{J_z \cdot v_x} & -\left(\frac{C_\alpha^{\mathrm{H}} \cdot {l_{\mathrm{H}}}^2 + C_\alpha^{\mathrm{V}} \cdot {l_{\mathrm{V}}}^2}{J_z \cdot v_x}\right) \end{bmatrix} \begin{bmatrix} v_y \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} \frac{C_\alpha^{\mathrm{V}}}{m_{\mathrm{F}}} \\ \frac{C_\alpha^{\mathrm{V}} \cdot l_{\mathrm{V}}}{J_z} \end{bmatrix} \delta. \tag{4.19}$$

Die Geschwindigkeit v_y ist dabei ohne gesonderte Kennzeichnung im fahrzeugfesten Koordinatensystem $\mathcal{K}_{\mathrm{F}}$ gegeben. Die Parameter C_α^V und C_α^H heißen Schräglaufsteifigkeiten und entspringen dem linearisierten Reifenmodell (hier nicht magic formula) [Schramm *et al.*, 2010].

4.2.2 Reduktion für eine vorgegebene Fehlertoleranz

In diesem Abschnitt wird das räumliche Zweispurmodell für eine vorgegebene Fehlertoleranz reduziert. Die Parameter des Modells werden so gewählt, dass das Modell einem Mittelklasse-Fahrzeug entspricht. Die Reduktion wird für eine Kurvenfahrt, eine beschleunigte Kurvenfahrt und einen Doppelspurwechsel durchgeführt. Dabei werden stets die Querbeschleunigung und die Gierbeschleunigung als Ausgangsvariablen gewählt. Für die Reduktionen wurden Termreduktion und Ähnlichkeitsbetrachtungen als Reduktionstechnik gewählt. Als Ranking-Verfahren kommen hier das Einschrittranking und das Residuenranking zum Einsatz.

Beschleunigte Kurvenfahrt

Bei der beschleunigten Kurvenfahrt wird der Lenkwinkel an den Reifen innerhalb der ersten Sekunde des Manövers kontinuierlich von 0° auf 1° erhöht. Das Fahrzeug startet mit einer Geschwindigkeit von 12m/s und wird während des gesamten Manövers mit einem Antriebsmoment von 490Nm beschleunigt. Die Gesamtdauer des Manövers beträgt zehn Sekunden. Die beiden Reduktionstechniken werden

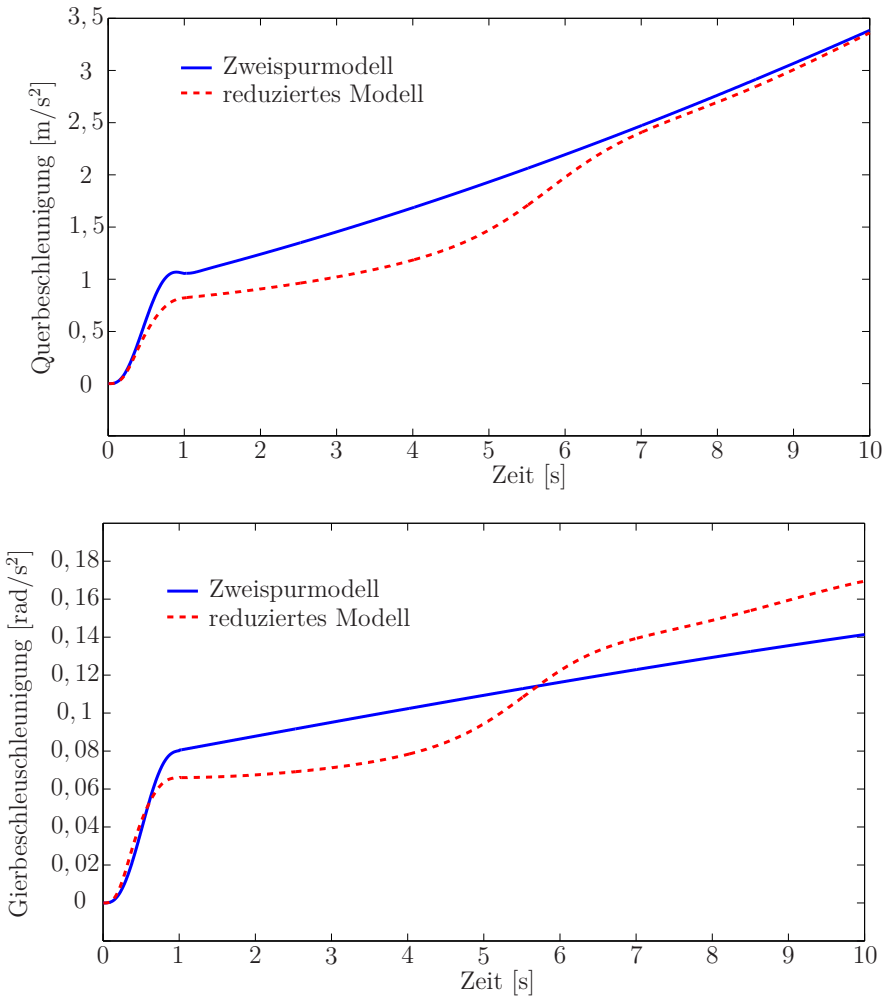


Abbildung 4.4: Querbeschleunigung und Gierbeschleunigung des ursprünglichen und des reduzierten Modells (beschleunigte Kurvenfahrt)

nicht zeitgleich, sondern nacheinander angewendet. So kommt zunächst die Termreduktion zum Einsatz. Hierbei wird eine Fehlertoleranz von 10% vorgegeben. Die Ausgangsvariablen des reduzierten Modells dürfen zu jedem Zeitpunkt also maximal zehn Prozent von den Ausgangsvariablen des ursprünglichen Modells abweichen. Anschließend erst kommen die Ähnlichkeitsbetrachtungen zum Einsatz. Hier wird eine Fehlertoleranz von 15% gewählt. Zur Erstellung der Rankingliste kommt dabei das Einschrittranking zum Einsatz.

Nach Durchführung der Termreduktion ist die Vertikalbewegung der Reifen ebenso wie die Roll- und Nickbewegung verschwunden. Das reduzierte Modell ist folglich ein ebenes Zweispurmodell. Neben den verschwundenen Freiheitsgraden wurden weitere Reduktionen durchgeführt, die nicht so einfach physikalisch interpretiert werden können. So sind beispielsweise im Reifenmodell einige Terme verschwunden. Die Ähnlichkeitsbetrachtungen reduzieren das Modell weiter auf ein Einspurmodell. So werden die Raddrehzahlen und die Reifenkräfte nicht mehr für jedes Rad einzeln,

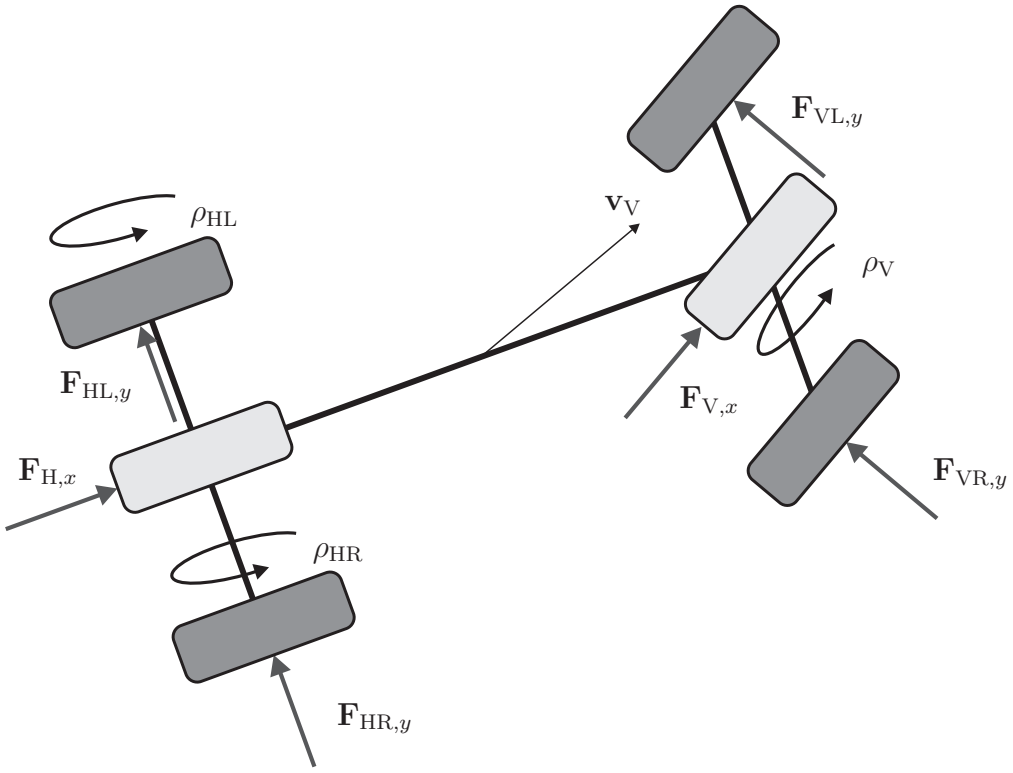


Abbildung 4.5: Schema des reduzierten Modells

sondern nur noch für vorne und hinten separat berechnet. Es verbleiben also fünf Freiheitsgrade. In Abbildung 4.4 ist die Querbeschleunigung und die Gierbeschleunigung des ursprünglichen und des reduzierten Modells gezeigt. Es scheint hier, dass der Fehler in der Querbeschleunigung mit anwachsender Simulationszeit kleiner wird. Dieser Effekt ist auf zwei Reduktionen zurückzuführen, die Fehler in unterschiedliche Richtung produzieren. Überwiegt zunächst der erste Fehler, so wird mit anwachsender Zeit der Einfluss des zweiten Fehlers größer und der Gesamtfehler scheint kleiner zu werden.

Kurvenfahrt

Die Kurvenfahrt ist mit der beschleunigten Kurvenfahrt nahezu identisch. Der einzige Unterschied besteht darin, dass hier kein Antriebsmoment wirkt. Für die Erstellung der Rankingliste wird das Residuenranking verwendet. Die restlichen Reduktionsparameter bleiben jedoch dieselben wie bei der vorherigen Reduktion. Auf den ersten Blick ist es daher erstaunlich, dass das Ergebnis der Reduktion sehr weit von dem obigen reduzierten Modell abweicht, wie in Abbildung 4.6 zu sehen ist. Dieses Ergebnis unterstreicht jedoch lediglich den starken Einfluss der Reihenfolge der Reduktionen. Die Wahl des Ranking-Verfahrens hat also großen Einfluss auf die Reduktion. In [Mikelsons & Brandt, 2011] wird dieser Effekt anhand der Reduktion des nichtlinearen Einspurmodells anschaulich gezeigt.

Das reduzierte Modell weist, wie oben, keine Vertikaldynamik mehr auf. Zwar sind

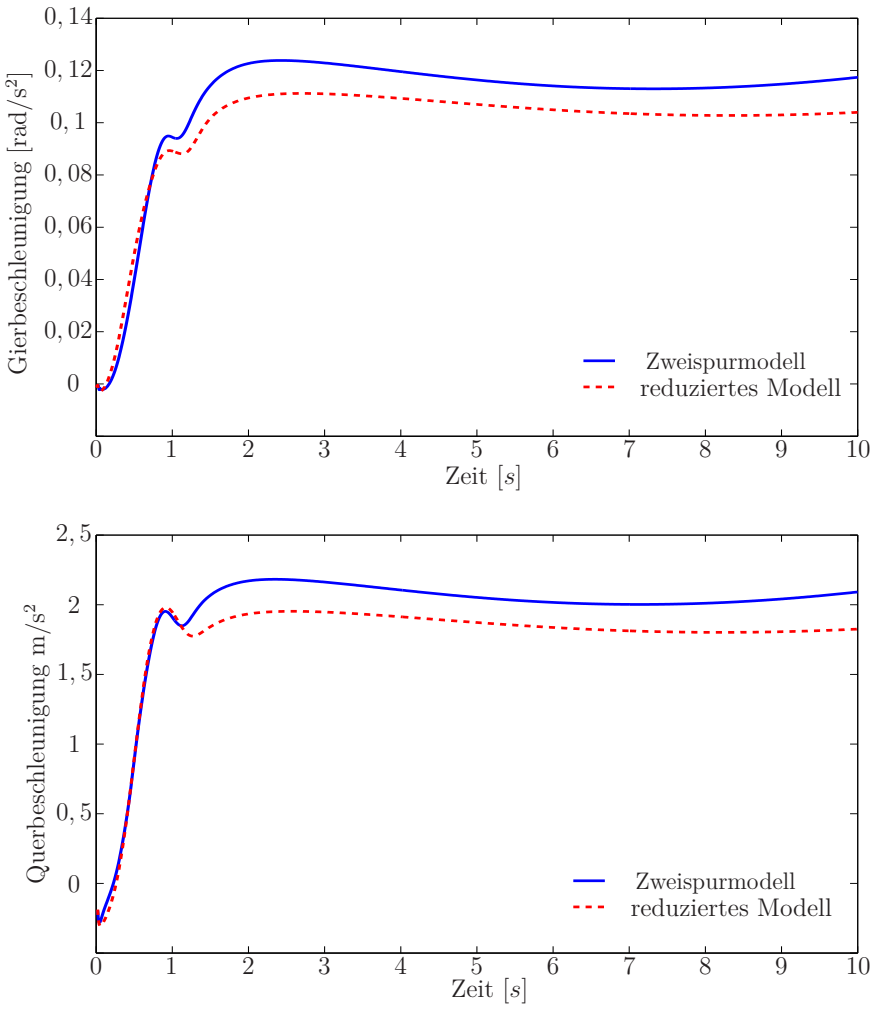


Abbildung 4.6: Querbeschleunigung und Gierbeschleunigung des ursprünglichen und des reduzierten Modells (Kurvenfahrt)

Nick- und Rollwinkel konstant, doch entkoppelt die Reduktion diese von ihren Ableitungen (den Winkelgeschwindigkeiten), die nach wie vor eine Dynamik aufweisen und im Modell verbleiben. Darüber hinaus liefern die Ähnlichkeitsbetrachtungen kein Einspurmodell. Lediglich die longitudinalen Reifenkräfte sowie die Rotation der Reifen an der Vorderachse werden zusammengefasst. Die lateralen Reifenkräfte sowie die Rotation der Reifen an der Hinterachse werden nach wie vor separat berechnet. Das reduzierte Modell weist somit acht Freiheitsgrade auf. In Abbildung 4.5 ist ein Schema des reduzierten Modells gezeigt.

4.2.3 Reduktion für Echtzeitsimulationen

Bisher wurde das räumliche Zweispurmodell für vorgegebene Fehlergrenzen reduziert. Dabei wurden zwei verschiedene Szenarien vorgegeben. In diesem Abschnitt wird die Fehlertoleranz auf 10% gesenkt und zusätzlich eine Echtzeit-Hardware vor-

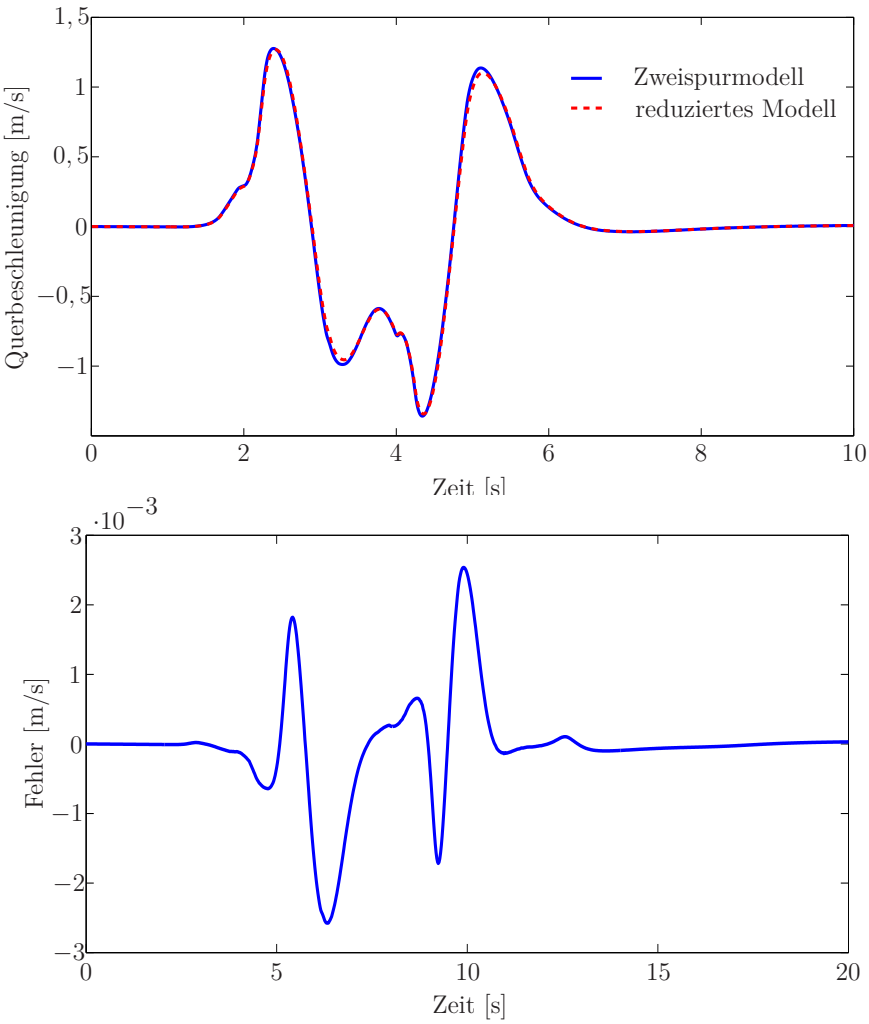


Abbildung 4.7: Querbeschleunigung und absoluter Fehler (Doppelspurwechsel bei 40km/h)

gegeben. Das Zweispurmodell soll für Echtzeitsimulationen auf einer Sony Playstation 2 mit einem Echtzeit-Takt von 1ms reduziert werden und dabei innerhalb der vorgegebenen Fehlertoleranz verbleiben. Als numerisches Integrationsverfahren soll in der Echtzeitsimulation das linear implizite Euler-Verfahren zum Einsatz kommen. Für einen Integrationsschritt werden ursprünglich $1,67 \cdot 10^6$ FLOPs benötigt. Eine Sony Playstation 2 ist in der Lage $9,95 \cdot 10^7$ FLOPS auszuführen[2] [Dongarra, 2009]. Für einen Integrationsschritt stehen folglich $9,95 \cdot 10^4$ FLOPs zur Verfügung. Die Fehlertoleranz von 10% soll für einen 20s andauernden Doppelspurwechsel bei 40km/h eingehalten werden. Als Ranking-Verfahren wird das Sensitivitätsranking verwendet. Als Reduktionstechnik wird lediglich die Termreduktion verwendet, da weitere Reduktionstechniken nicht nötig sind, um das Modell in der gewünschten Echtzeitumgebung zu simulieren [Mikelsons *et al.*, 2010a]. Im reduzierten Modell ist wieder sowohl die Vertikaldynamik als auch die Nick- und Wankbewegung vernach-

[2]Das entspricht in etwa der Rechenleistung eines IBM 486 mit 33 Mhz.

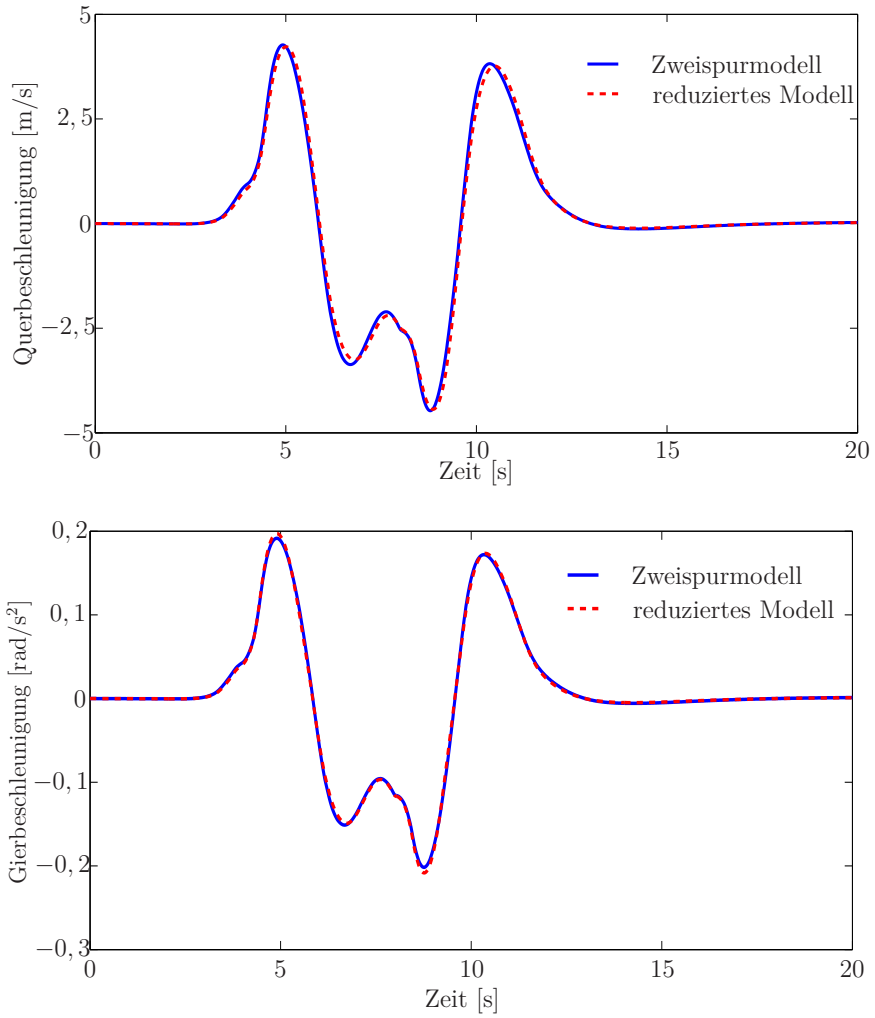


Abbildung 4.8: Querbeschleunigung und Gierbeschleunigung (Doppelspurwechsel bei 80km/h)

lässigt. Weitere Reduktionen haben das Reifenmodell vereinfacht, sind aber nicht physikalisch interpretierbar. In Abbildung 4.7 ist die Querbeschleunigung beider Modelle sowie der absolute Fehler gezeigt. Offensichtlich verbleibt der Fehler weit unter den geforderten 10%. Für einen Integrationsschritt mit dem linear impliziten Euler-Verfahren werden nun nur noch 92215 FLOPs benötigt. Dies bedeutet, dass lediglich 1/18 der für das räumliche Zweispurmodell benötigten Rechenoperationen durchgeführt werden müssen.

Die Fehlertoleranz von 10% wird natürlich nur für das vorgegebene Szenario eingehalten. Folglich stellt sich die Frage wie groß der Fehler bei anderen Szenarien werden kann. Dies ist davon abhängig, wie *gut* das Szenario, das für die Reduktion gewählt wurde, ist. In dem obigen Beispiel wurde ein typisches Manöver gewählt. Um das reduzierte Modell zu validieren, wird der Doppelspurwechsel noch einmal für eine Geschwindigkeit von 80km/h betrachtet. Auch bei diesem Manöver ist der Unterschied zwischen dem räumlichen Zweispurmodell und dem reduzierten Modell in den

Ausgangsvariablen gering, wie in Abbildung 4.8 zu sehen. Der maximal auftretende Fehler liegt hier nur minimal über der für die Reduktion vorgegebenen Toleranz von 10%. Darüber hinaus behalten alle Variablen ihre physikalische Bedeutung, so dass nicht nur die Aussgangsvariablen, sondern auch alle internen Variablen zumindest qualitativ unverändert bleiben.

Die Anwendung der gleichungsbasierten Modellreduktion zeigt, dass die Qualität des Rankings großen Einfluss auf das Ergebnis der Reduktion hat. Dieses Ergebnis unterstreicht die Wichtigkeit eines schnellen und genauen Ranking-Verfahrens. Darüber hinaus ist die Wahl des Szenarios ein zentraler Punkt. Die Wahl des Doppelspurwechsels führt auf ein Modell, das auch für ein modifiziertes Szenario realitätsgetreue Simulationsergebnisse liefert.

4.2.4 Reduktion für intervallwertige Szenarien

Die in Abschnitt 3.4.3 beschriebene Erweiterung der gleichungsbasierten Modellreduktion für intervallwertige Szenarien wurde im Rahmen dieser Arbeit auch auf Modelle aus der Fahrzeugdynamik angewendet. Leider stellte sich dabei heraus, dass sowohl das räumliche Zweispurmodell, als auch das nichtlineare Einspurmodell nicht für relevante Manöver mit dem numerischen Integrationsverfahren VSPODE simuliert werden können. Die Simulation bricht nach wenigen Schritten ab, da die Intervalle der Lösung bereits zu diesem frühen Zeitpunkt zu groß geworden sind. Vor Beginn einer Reduktion für intervallwertige Szenarien muss also sichergestellt sein, dass das vorliegende Modell mit dem entsprechenden numerischen Integrationsverfahren gelöst werden kann. Hierfür ist es nicht möglich, allgemeine Kriterien zu formulieren. Die Überschätzung wird jedoch stark von der Steifigkeit sowie von der *Nichtlinearität* des Systems beeinflusst. Für diese beiden Eigenschaften existiert jedoch zum einen keine verbindliche Definition, zum anderen ließe sich auch bei bekannter Steifigkeit und Nichtlinearität nicht zuverlässig voraussagen, bis zu welchem Zeitpunkt das vorliegende System simuliert werden kann. Aus diesem Grund wird hier die gleichungsbasierte Modellreduktion auf das lineare Einspurmodell angewendet.

Obwohl das lineare Einspurmodell bereits ein sehr einfaches Modell darstellt, werden für die hier verwendeten Szenarien einige Reduktionen erwartet. Es kommen drei Szenarien zum Einsatz. Das erste Szenario ist eine Geradeausfahrt mit intervallwertiger Geschwindigkeit. Hierbei ist der Lenkwinkel jedoch nicht identisch Null, sondern ein kleines Intervall. Das zweite Szenario ist eine Kurvenfahrt mit konstanter Geschwindigkeit. Bei dem dritten Szenario ist zusätzlich die Geschwindigkeit intervallwertig gewählt.

Geradeausfahrt

Für die Geradeausfahrt wird die intervallwertige Geschwindigkeit $v_x = \left[16\frac{2}{3}, 19\frac{4}{9}\right]\mathrm{m/s}$ verwendet. Der Lenkwinkel beträgt $\left[-\frac{\pi}{1800}, \frac{\pi}{1800}\right]^\circ$. In der Folge wird das lineare Einspurmodell also für Geradeausfahrten mit Geschwindigkeiten zwischen $v_x = 16\frac{2}{3}\mathrm{m/s}$ und $v_x = 19\frac{4}{9}\mathrm{m/s}$ und Kurvenfahrten mit sehr kleinen Lenkwinkeln reduziert. Die

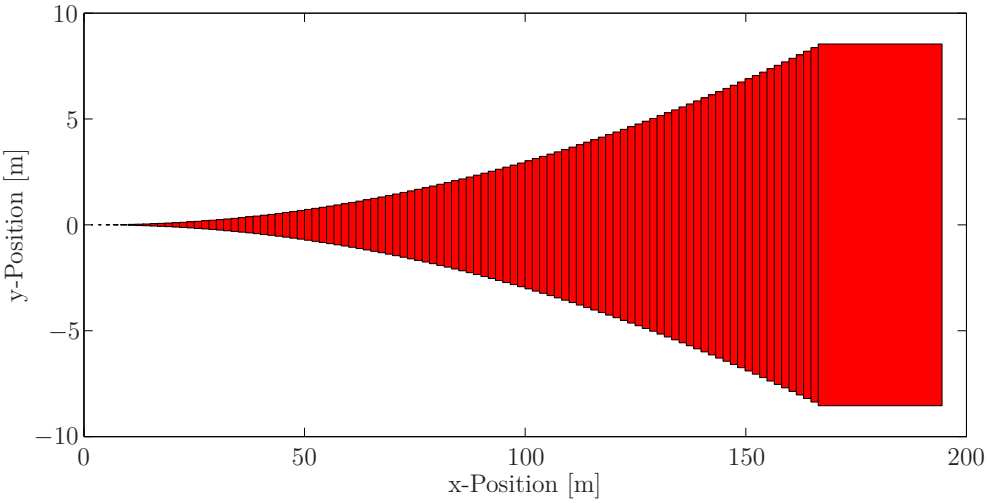


Abbildung 4.9: Referenztrajektorie

Position des Fahrzeugs stellt die Ausgangsvariablen dar. Das lineare Einspurmodell wird also um die Berechnung der Ausgangsvariabeln

$$\dot{x}_{\mathrm{F}} = v_x \cos(\psi) - v_y \sin(\psi) \tag{4.20}$$
$$\dot{y}_{\mathrm{F}} = v_x \sin(\psi) + v_y \cos(\psi) \tag{4.21}$$

erweitert. Die Simulationsdauer beträgt 10s. In Abbildung 4.9 ist die Referenztrajektorie gezeigt. Die Fehlertoleranz bei der Reduktion beträgt 10m für x_{F} und $0,6$m für y_{F}. Wird die Reduktion mit dem in Abschnitt 3.4.3 vorgestellten Ranking-Verfahren und Termreduktion sowie Termsubstitution als Reduktionstechnik durchgeführt, so lautet das reduzierte Modell

$$\dot{v}_y = [-0.18137, 0.18137] + [-0.14601, 0.14601], \tag{4.22}$$
$$\ddot{\psi} = [-0.10917, 0.10917] + [-0.09457, 0.09457], \tag{4.23}$$
$$\dot{x}_{\mathrm{F}} = \cos[-0.04335, 0.04335] \cdot v_x + [16.6667, 16.6667], \tag{4.24}$$
$$\dot{x}_{\mathrm{F}} = \sin\psi \cdot v_x. \tag{4.25}$$

Die Abbildung 4.10 zeigt den Fehler in der Position. In der Abbildung scheint der Fehler in y_{F} über der gesamten Simulationsdauer weit von der Fehlertoleranz entfernt zu bleiben. Während der Reduktion steigt dieser Fehler jedoch bis auf 0.3071m an und fällt danach wieder ab auf 0.0463m. Dieser Effekt tritt auf, falls zwei Reduktionen dazu tendieren, dass ihre Fehler sich gegenseitig aufheben. Dieser Effekt ist natürlich ein weiteres Argument für die Verwendung von Clustern.

Kurvenfahrt

Für die Kurvenfahrt wird zunächst die reellwertige Geschwindigkeit $v_x = 16\frac{2}{3}$m/s gewählt. Für einen Lenkwinkel von $\delta = \left[\frac{2}{3}, 1\right]^\circ$ ergibt sich die in Abbildung 4.11 gezeigte Trajektorie. Die Ausgangsvariablen sind die Gierbeschleunigung $\ddot{\psi}$ sowie

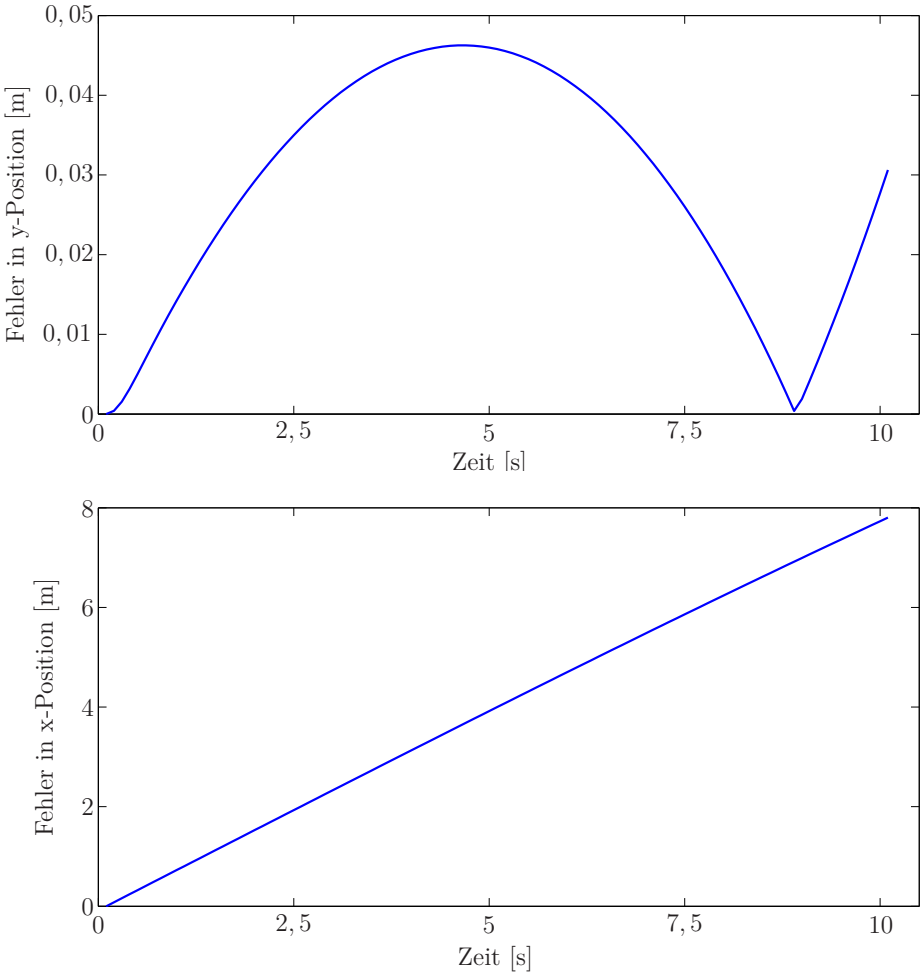


Abbildung 4.10: Fehler in x_F und y_F

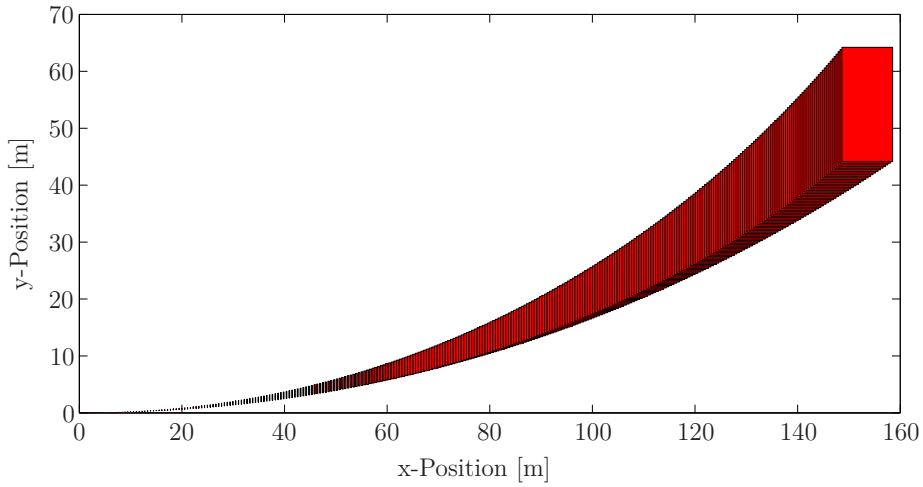


Abbildung 4.11: Referenztrajektorie für eine Kurvenfahrt mit konstanter Geschwindigkeit

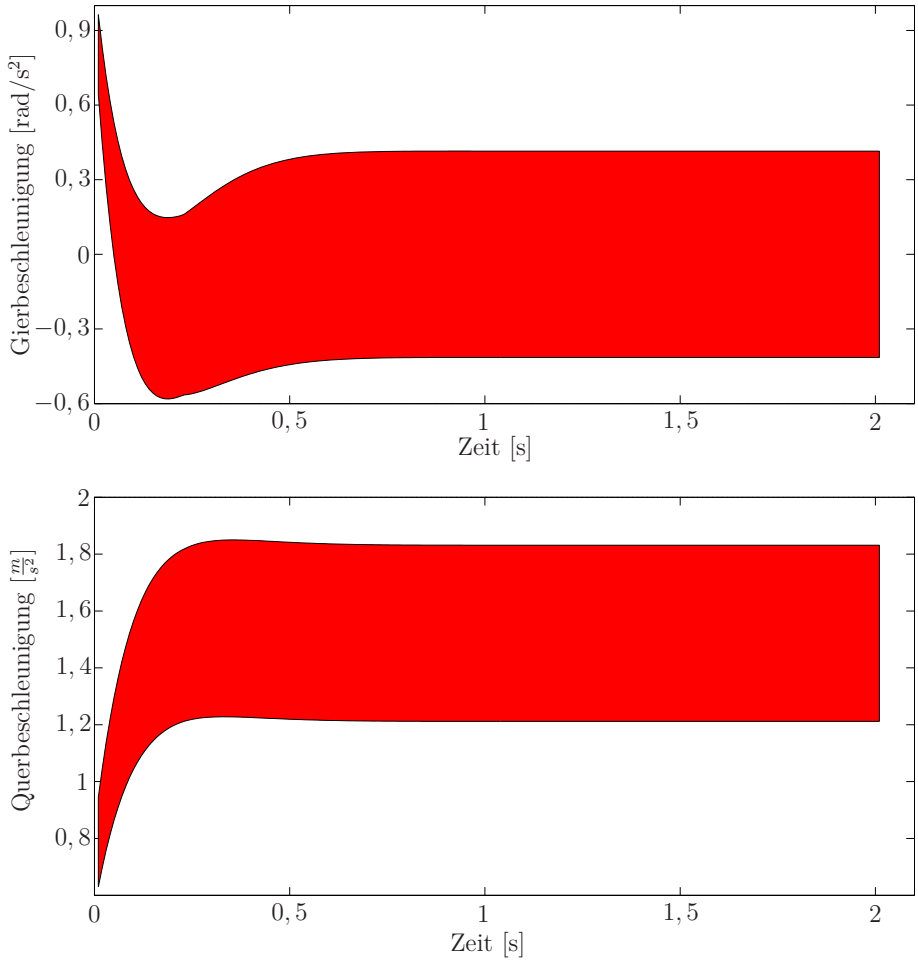


Abbildung 4.12: Querbeschleunigung und Gierbeschleunigung für eine Kurvenfahrt mit intervallwertigem Lenkwinkel

die Querbeschleunigung $\ddot{x}_y$. Für einen konstanten reellwertigen Lenkwinkel erreicht das System nach einer gewissen Zeit ein stationäres Verhalten. Für einen intervallwertigen Lenkwinkel $\delta = \left[\frac{2}{3}, 1\right]^\circ$ gilt dasselbe, wie in Abbildung 4.12 zu sehen ist. Startet die Simulation bereits im stationären Zustand , so ist zu erwarten, dass bei einer Reduktion die rechte Seite von Gleichung (4.19) verschwindet. So lautet das reduzierte Modell in diesem Fall auch

$$\dot{v}_y = 0, \tag{4.26}$$

$$\ddot{\psi} = 0. \tag{4.27}$$

Interessanter ist der Fall, in dem das System nicht im stationären Zustand startet. Wird zusätzlich eine intervallwertige Geschwindigkeit $v_x = \left[13\frac{8}{9}, 16\frac{2}{3}\right]\mathrm{m/s}$ verwendet und die Fehlertoleranzen $\epsilon_{\dot{v}_y} = 0,3\mathrm{m/s}$ für die Querbeschleunigung beziehungsweise $\epsilon_{\ddot{v}_y} = 0,1$ für die Gierbeschleunigung verwendet, so lautet das reduzierte System

$$\dot{v}_y = 0 \tag{4.28}$$

$$\ddot{\psi} = -\left(\frac{C_\alpha^{\mathrm{H}} \cdot {l_{\mathrm{H}}}^2 + C_\alpha^{\mathrm{V}} \cdot {l_{\mathrm{V}}}^2}{J_z \cdot v_x}\right)\dot{\psi} + \frac{C_\alpha^{\mathrm{V}} \cdot l_{\mathrm{V}}}{J_z}\delta. \tag{4.29}$$

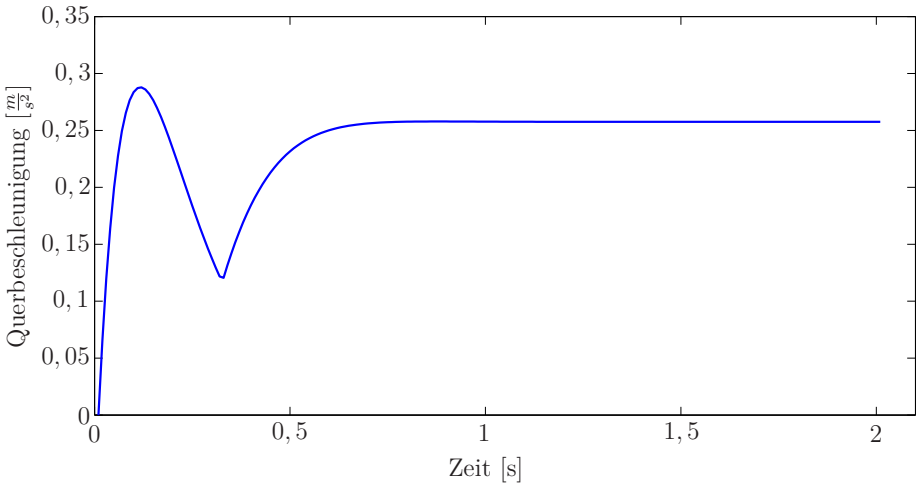


Abbildung 4.13: Fehler in der Querbeschleunigung

Dabei betrug die Simulationszeit 2s. Innerhalb dieser Zeit erreicht das System einen stationären Zustand, so dass auch der Fehler konstant bleibt. Dies zeigt auch Abbildung 4.13, in welcher der Fehler in der Querbeschleunigung über die Zeit gezeigt wird. Auf die Abbildung des Fehlers in der Gierbeschleunigung wird hier verzichtet, da dieser nahehzu identisch verläuft.

4.3 Anwendung auf das Modell einer Baumaschine

Die Beispiele aus dem Bereich der Fahrzeugdynamik zeigen sehr anschaulich, wie die gleichungsbasierte Modellreduktion arbeitet. Die vorgestellten Modelle entstammen jedoch alle dem Bereich der Mehrkörperdynamik. Folglich waren auch in dem komplexesten Modell, dem räumlichen Zweispurmodell, keine Komponenten aus anderen Domänen als der Mechanik zu finden.

In diesem Abschnitt wird die Effektivität und Wirkungsweise der gleichungsbasierten Modellreduktion anhand eines komplexen Systems präsentiert, in dem Komponenten aus der Mechanik, der Hydraulik und der Regelungstechnik zu finden sind. Hierzu wurde im Rahmen dieser Arbeit der in Abbildung 2.3 gezeigte Skid-Steer-Loader modelliert.

Der Skid-Steer-Loader ist eine kompakte Baumaschine mit hoher Monövrierbarkeit. Einsatzorte sind Baustellen, an denen nur wenig Platz zu Verfügung steht. Der hohe Grad an Manövrierbarkeit ist dem besonderen Lenkungskonzept, dem *skid steering* geschuldet. Der Skid-Steer-Loader hat vier Räder wie auch ein Standard-KFZ. Die Räder auf der linken Seite werden jedoch unabhängig von den Rädern auf der rechten Seite angetrieben. Somit bestimmt die Differenz der Umdrehungsgeschwindigkeiten der Räder der beiden Seiten die Richtung, in die der Skid-Steer-Loader sich bewegt.

Gesteuert wird der Skid-Steer-Loader mit Hilfe eines Pedals und zwei Joysticks. Über den linken Joystick wird die Geschwindigkeit gesteuert und eine Bewegungsrichtung

vorgegeben, während der rechte Joystick für die Steuerung der Schaufel verwendet wird. Mit Hilfe des Pedals kann die Geschwindigkeitsvorgabe skaliert werden.

Im folgenden Abschnitt wird das verwendete Modell des Skid-Steer-Loaders beschrieben.

4.3.1 Modellierung des Skid-Steer-Loaders

Das Modell des Skid-Steer-Loaders besteht aus dem Steuergerät, dem Dieselmotor, den hydrostatischen Antrieben, der Arbeitshydraulik und dem Chassis. Im Rahmen dieser Arbeit wurde die Arbeitshydraulik nicht modelliert und die Schaufel somit als starr angenommen. Die Modelle der verbleibenden Komponenten werden im Folgenden vorgestellt.

Steuergerät

Die Bedienereingaben werden von einem Steuergerät digitalisiert und verarbeitet. In dem verwendeten Modell werden die Eingänge des Joysticks zur Bedienung der Schaufel vernachlässigt. Die Ausgänge des Steuergeräts sind zum einen das gesampelte Signal des Pedals und zum anderen die Schwenkwinkel in den Verstellpumpen der hydrostatischen Antriebe.

Das gesampelte Signal des Pedals beeinflusst zusätzlich die beiden anderen Ausgänge. Das Verhältnis zwischen den Schwenkwinkeln und den Eingängen wird dabei durch drei Kennlinien bestimmt, welche durch Messungen erstellt wurden. Während zwei dieser Kennlinien das Verhältnis zwischen den gesampelten Joystickeingaben und den Schwenkwinkeln charakterisiert, beschreibt die dritte Kennlinie den Einfluss des Pedals. Darüber hinaus berücksichtigt das Modell Begrenzungen für die Ausgänge des Steuergeräts.

Dieselmotor

Der Dieselmotor treibt den hydrostatischen Antrieb an. Der Eingang des Motors ist das normalisierte gesampelte Signal des Pedals. Die Motordrehzahl wird von einem PID Regler so geregelt, dass sie proportional zu dem Eingangssignal ist. Ist das Eingangssignal Null, so beträgt die Umdrehungsgeschwindigkeit 1000 Umdrehungen pro Minute. Die maximale Anzahl von Umdrehungen pro Minute bei dem Eingangssignal Eins beträgt 3200. Das Verhältnis zwischen Umdrehungsgeschwindigkeit und Antriebsmoment ist wieder durch zwei Kennfelder abgebildet. Das dynamische Verhalten des Motors wird approximiert durch ein PT_2-Element.

Hydrostatischer Antrieb

Bei der Modellierung der Fahrzeugdynamik wurde auf die Modellierung des Antriebsstrangs verzichtet und das Antriebsmoment als Systemeingang verwendet. Im

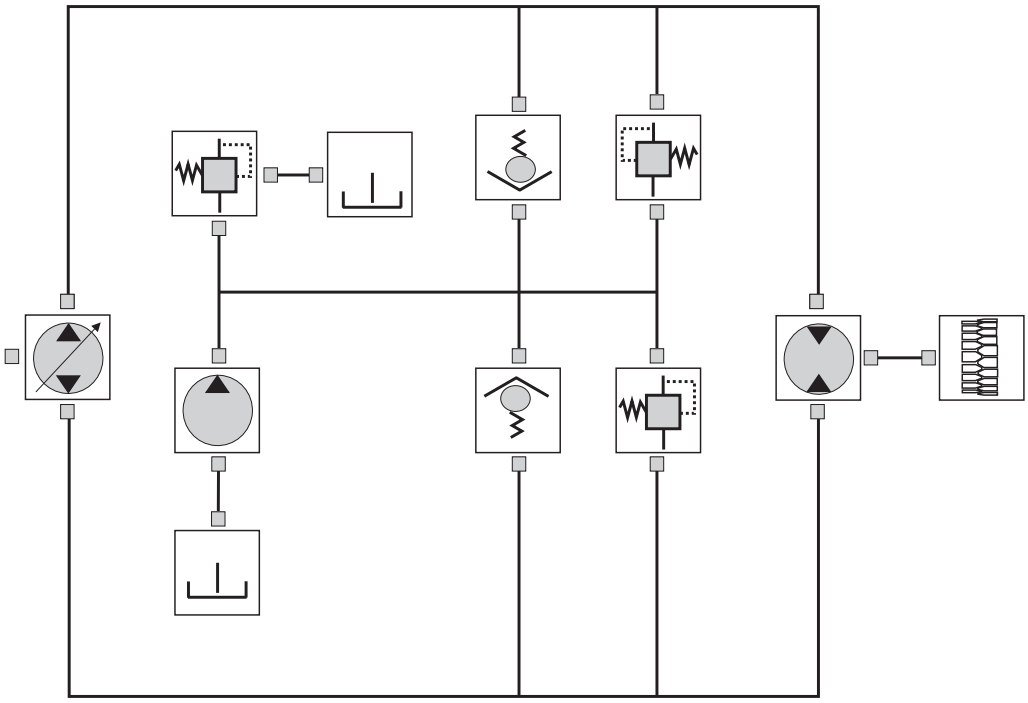
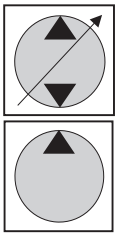


Abbildung 4.14: Objektdiagramm des hydrostatischen Antriebs

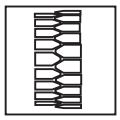
Gegensatz dazu wurden die hydrostatischen Antriebe des Skid-Steer-Loaders detailliert modelliert.

Der Eingang der beiden hydrostatischen Antriebe ist jeweils das Antriebsmoment des Dieselmotors sowie der Schwenkwinkel der Verstellpumpe.

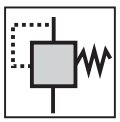


Das Antriebsmoment treibt die Verstellpumpe und eine Speisepumpe an. Beide Pumpen wandeln mechanische in hydraulische Energie um und sind nahezu identisch modelliert. Einzig der Schwenkwinkel ist bei der Verstellpumpe variabel, während er bei der Speisepumpe konstant ist, weil diese stets das gleiche Volumen verdrängt. Im Falle der Verstellpumpe ist die dynamische Antwort auf eine Veränderung des Schwenkwinkels durch ein PT_2-Element abgebildet. Zusätzlich ist Reibung und Leckage modelliert. Die Speisepumpe hat die Aufgabe, verloren gegangenes Fluid wieder nachzufüllen und den Druck im Rücklauf über einem vorgegebenen Minimum zu halten, während die Verstellpumpe an einen Konstantmotor gekoppelt ist.

Die Modellierung des Konstantmotors ist ähnlich zu der Modellierung der Pumpen. Der wichtigste Unterschied ist, dass hier hydraulische in mechanische Energie gewandelt wird. Darüber hinaus ist die Trägheit des Motors bedeutend größer, weshalb der Einfluss der Reibung an Einfluss gewinnt.



Der Konstantmotor ist mit einem Kettentrieb verbunden. Dieser Kettentrieb bildet den Ausgang des hydrostatischen Antriebs. Er überträgt das Moment des Konstantmotors auf die Räder. In dieser Arbeit wird vereinfacht angenommen, dass der Kettentrieb wie ein Getriebeelement fungiert.



Ein Überdruckventil sorgt dafür, dass der Druck im Rücklauf nicht zu groß wird. Ebenso garantieren zwei Überdruckventile in der Hochdruckleitung, dass der Druck hier begrenzt ist. Ist ein Überdruckventil komplett geöffnet, so verhält es sich wie eine Blende. Zwischen den Zuständen komplett geöffnet und komplett geschlossen hängt die Durchflussgeschwindigkeit linear von dem Differenzdruck ab. Auch hier ist die Dynamik über ein PT_2-Element abgebildet und Leckage berücksichtigt.

Um eine feste Flussrichtung des Fluids sicher zu stellen, existieren zwei Absperrventile in jedem hydrostatischen Antrieb. Mit Hilfe einer Kugel oder Kegels wird sichergestellt, dass das Fluid nur in eine Richtung fließen kann. Eine Feder drückt die Kugel oder den Kegel dabei in den Ventilsitz. Somit können mit Hilfe verschiedener Federn Ventile realisiert werden, die bei verschiedenen Drücken öffnen. Der Zusammenhang zwischen Durchflussgeschwindigkeit und Differenzdruck ist hier nicht mehr linear. Ansonsten entspricht die Modellierung des Absperrventils der Modellierung des Überdruckventils. Zu beachten ist, dass die Leckage hier einen größeren Einfluss hat. Abbildung 4.14 zeigt das Objektdiagramm eines hydrostatischen Antriebs.

Chassis

Das Chassis besteht aus einem Körper mit vier Reifen. Bei dem Skid-Steer-Loader sind die Radachsen starr mit dem Chassis verbunden. Folglich ist auch keine Radaufhängung modelliert. Das Modell für die Räder basiert auf dem Reifenmodell aus [Rill, 1994] und wurde der Modelica-Vehicle-Dynamics-Library entnommen [Andreasson, 2003].

Gesamtmodell

Aus den beschriebenen Submodellen kann das Modell für den Skid-Steer-Loader leicht zusammengesetzt werden. Abbildung 2.2 zeigt das entsprechende Objektdiagramm. Das flache Modell des Skid-Steer-Loaders besteht aus ungefähr 1700 Gleichungen. Nach dem Tearing ist das Modell eine ODE der Dimension 52.

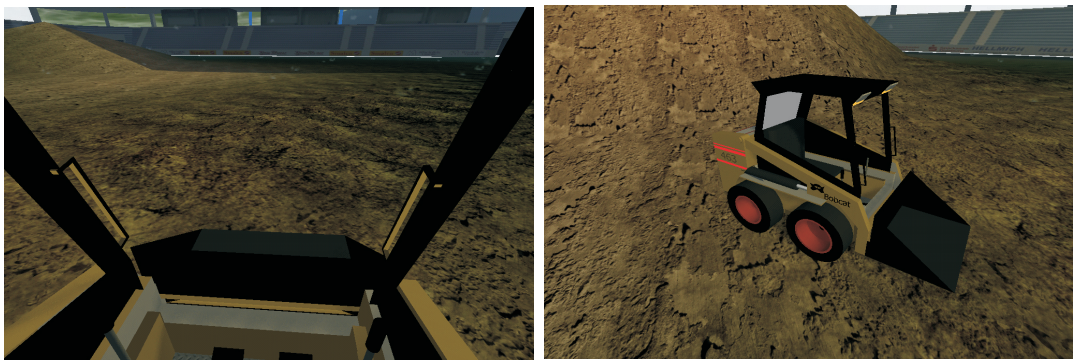


Abbildung 4.15: Screenshots aus dem Skid-Steer-Loader Simulator

In [Mikelsons *et al.*, 2009a] wurde die Längsdynamik dieses Modells anhand von Messungen validiert. Die Querdynamik wurde nicht anhand von Messungen validiert. Das Modell wurde jedoch in das Simulatorkonzept DRIVASSIST® integriert und auf Plausibilität untersucht. In Abbildung 4.15 sind zwei Screenshots dieser Untersuchung zu sehen.

4.3.2 Reduktion des Skid-Steer-Loaders für Echtzeitsimulationen

Das im obigen Abschnitt beschriebene Modell des Skid-Steer-Loaders kann beispielsweise eingesetzt werden, um einen Simulator für Baumaschinen zu realisieren. In diesem Fall muss das Modell natürlich in Echtzeit simuliert werden. Bereits für die Validierung wurde das Modell in die DRIVASSIST®-Architektur integriert. Hierzu wurde das Modell im Real-Time-Workshop für das Echtzeitbetriebssystem xPC-Target kompiliert. Im Real-Time-Workshop stehen lediglich explizite numerische Integrationsverfahren zur Auswahl. Daher kam für die Simulation das explizite Euler-Verfahren zum Einsatz. Aufgrund der Komplexität des Modells musste die relativ hohe Taktzeit von 4ms gewählt werden, obwohl als Echtzeithardware ein Intel Core2Duo mit 3GHz zur Verfügung stand. Darüber hinaus führte die Kombination aus explizitem Solver und hoher Taktzeit dazu, dass das numerische Integrationsverfahren bei hochdynamischen Manövern abbrach.

Daher wird in diesem Abschnitt das Modell des Skid-Steer-Loaders reduziert, so dass es in Echtzeit auf einer leistungsschwächeren Echtzeithardware als oben beschrieben mit dem in Abschnitt 3.5.1 präsentierten LI3-Verfahren simuliert werden kann. Um einen Integrationsschritt mit dem LI3-Verfahren durchzuführen werden $138 \cdot 10^5$FLOPs benötigt (siehe Abschnitt 3.5.2). Für den Einsatz in einem Simulator ist ein Echtzeittakt von 1ms wünschenswert. Ein Intel Pentium 4 mit $1,5$Ghz ist nach [Dongarra, 2009] in der Lage, $326 \cdot 10^8$FLOPS auszuführen. Erfahrungen mit DRIVASSIST zeigen, dass die Hardwarekommunikation ungefähr 90% der verfügbaren Rechenleistung benötigt. Folglich stehen für numerische Berechnungen noch 10% der gesamten Rechenleistung zur Verfügung. Somit sollte das reduzierte Modell maximal $326 \cdot 10^4$FLOPs für einen Integrationsschritt benötigen. Aufgrund der Komplexität des Modells wird für die Reduktion das Residuenranking verwendet.

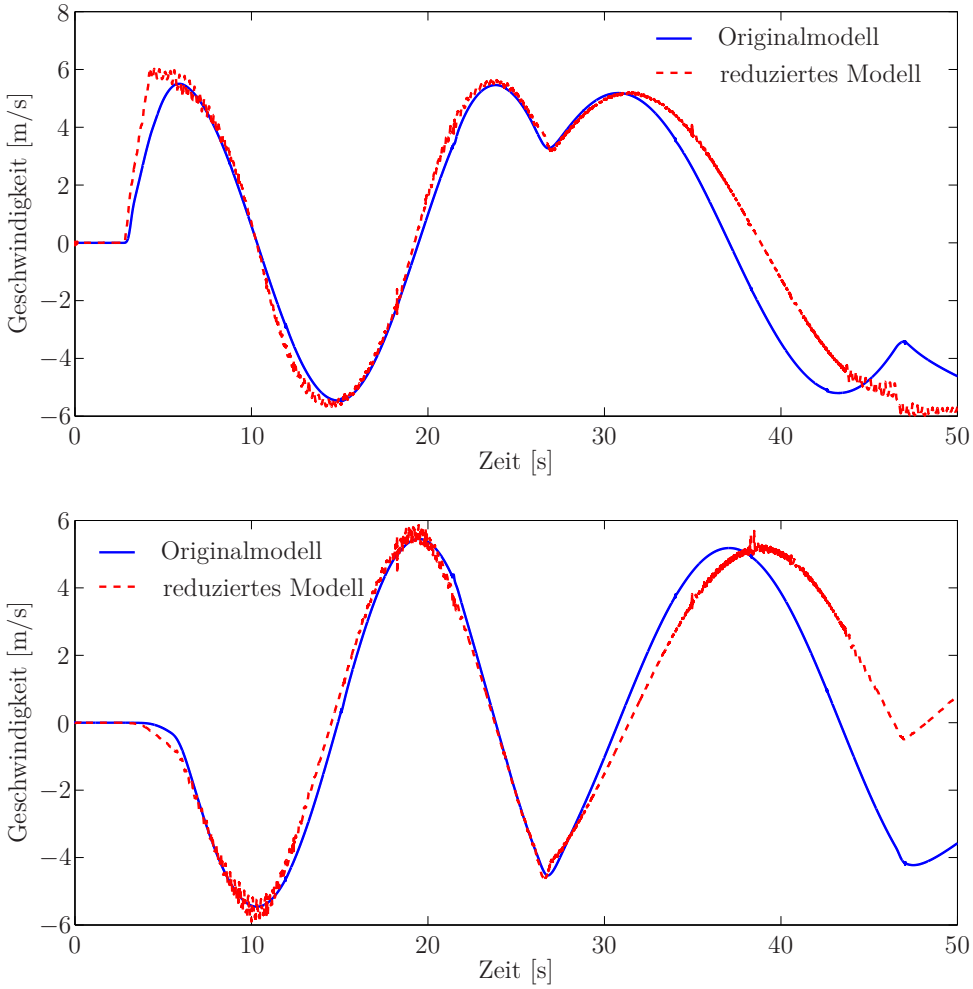


Abbildung 4.16: Longitudinale (oben) und laterale (unten) Geschwindigkeit des ursprünglichen unde des reduzierten Modells

Um die große Differenz zwischen den ausführbaren und den benötigten FLOPs zu senken, kommt die Termreduktion zum Einsatz. Als Szenario dient eine Acht-Fahrt, die im Stillstand beginnt und endet. Dabei sind die longitudinale sowie die laterale Geschwindigkeit die Ausgangsvariablen. Im Gegensatz zu den bisherigen Reduktionen wird hier keine Toleranz für den maximalen Fehler, sondern für den mittleren quadratischen Fehler vorgegeben. Dies hat den Vorteil, dass kleine Phasenverschiebungen in den Ausgangsvariablen kleinere Fehler zur Folge haben. Somit werden also auch Reduktionen akzeptiert, die kleine Phasenverschiebungen verursachen. Die Fehlertoleranz beträgt $0.005\mathrm{m}^2/\mathrm{s}^2$.

Das reduzierte Modell benötigt nur noch $246 \cdot 10^4$FLOPs für einen Integrationsschritt mit dem LI3-Verfahren. In Abbildung 4.16 ist die longitudinale und laterale Geschwindigkeit gezeigt. Offensichtlich haben die Reduktionen eine hochfrequente Schwingung in den Geschwindigkeiten zur Folge. Der Grund für diese Schwingungen liegt in der Vernachlässigung der Vertikalbewegung. In dem Modell des Drehgelenks, welches ein Rad mit dem Chassis verbindet, wird die Geschwindigkeit und

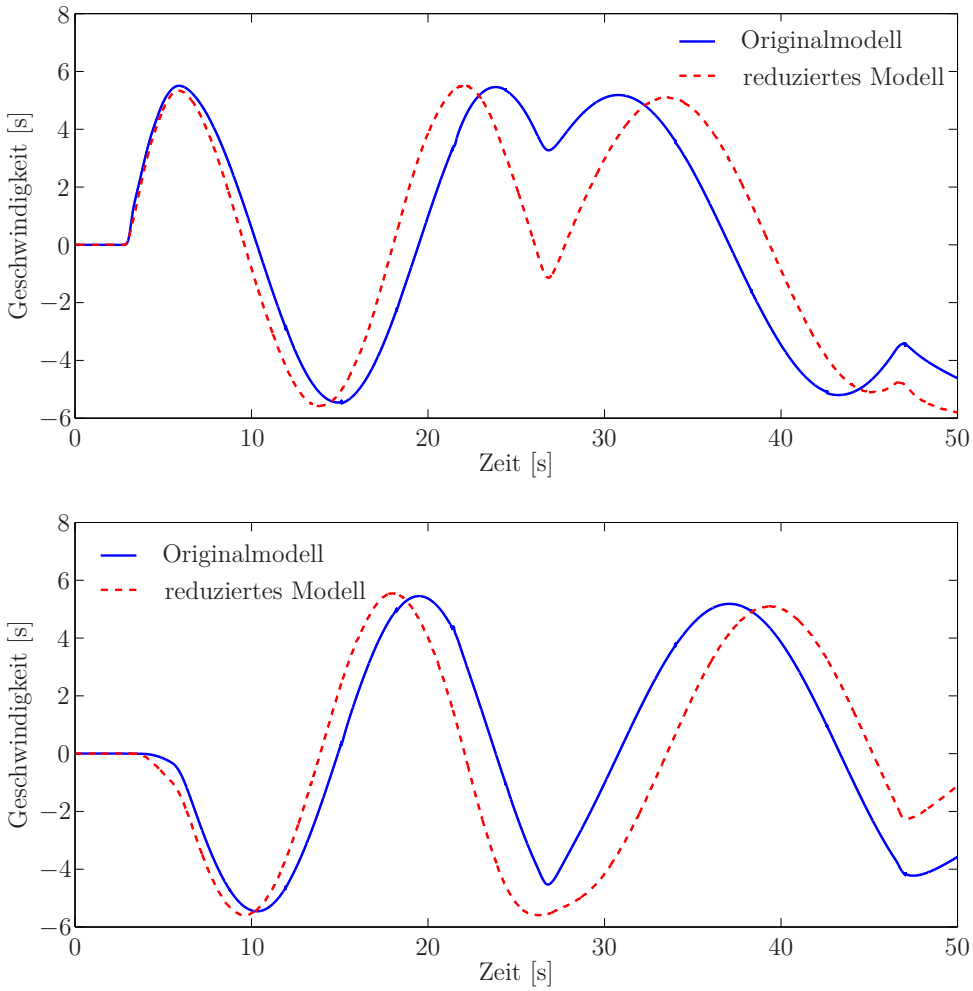


Abbildung 4.17: Longitudinale (oben) und laterale (unten) Geschwindigkeit des ursprünglichen unde des reduzierten Modells

die Beschleunigung des Rades vom Koordinatensystem des Rades mit Hilfe einer Rotationsmatrix $\mathbf{S}$ in das Koordinatensystem des Chassis transformiert. Das Residuenranking weist derjenigen Reduktion, welche den Eintrag $\mathbf{S}_{3,3}$ zu Null setzt, einen sehr niedrigen Rankingwert zu, da hierdurch der Einfluss der vertikalen Geschwindigkeit des Rades auf die Bewegung des Chassis vernachlässigt wird. In dem Modell des Gelenks wird jedoch auch die Inverse der Rotationsmatrix $\mathbf{S}^T$ zur Transformation der Kräfte und Momente verwendet. Nach Durchführung der oben genannten Reduktion ist $\mathbf{S}$ jedoch nicht mehr orthogonal und $\mathbf{S}^T$ somit nicht mehr die inverse Matrix. Für den Drehwinkel $\beta = 0°$ ist die reduzierte Matrix sogar singulär. In der Folge sind die auftretenden Kräfte nicht konsistent mit der Beschleunigung des Rades, was zu den beschriebenen Schwingungen führt. Diese Schwingungen sind natürlich unerwünscht, da sie zum einen nicht dem realen Systemverhalten entsprechen und zum anderen den numerischen Aufwand der Simulation erhöhen. Vermieden werden kann dieser Effekt beispielsweise durch die Vorgabe eines entsprechend kleinen maximalen Fehlers. Dies würde jedoch dazu führen, dass auch viele andere Reduktionen nicht

durchgeführt werden können. Daher werden hier in einer zweiten Reduktion nicht nur der mittlere quadratische Fehler in den Ausgangsvariablen, sondern zusätzlich die in der Lösung enthaltenen Frequenzen betrachtet. Von den im Approximationsalgorithmus berechneten Lösungen wird zusätzlich eine schnelle Fouriertransformation (FFT) durchgeführt und eine zusätzliche Fehlertoleranz für die Abweichung dieser Koeffizienten von den Koeffizienten der FFT der Referenzlösung eingeführt. Dadurch wird sichergestellt, dass das Frequenzspektrum der Lösung des reduzierten Modells sich nicht signifikant von dem Frequenzspektrum der Lösung des ursprünglichen Modells unterscheidet. Mit der Fehlertoleranz von $0.5\mathrm{m}^2/\mathrm{s}^2$ für die mittlere quadratische Abweichung der Fourierkoeffizienten erhält man ein weiteres reduziertes Modell. In Abbildung 4.17 sind die Lösungen der Ausgangsvariablen dargestellt. Die Schwingungen sind offensichtlich verschwunden. Dieses Modell benötigt $247 \cdot 10^4$FLOPs für einen Integrationsschritt und kann somit auf der gewünschten Echtzeithardware mit der gewünschten Taktzeit von 1ms simuliert werden.

In dem mechanischen Teil des reduzierten Modells ist, wie oben schon beschrieben, die Vertikaldynamik vernachlässigt. Zusätzlich wurde der Roll- und Wankwinkel in den meisten trigonometrischen Funktionen zu Null gesetzt, jedoch nicht im gesamten Modell vernachlässigt. In der Hydraulik wurde das dynamische Verhalten der Rückschlagventile sowie der differentielle Anteil des Motorreglers eliminiert.

4.4 Ansatz zur Erstellung von generischen Modellen

In den vorangegangenen Beispielen wurde unter Verwendung der gleichungsbasierten Modellreduktion der Detaillierungsgrad von Modellen dynamischer Systeme variiert. Die Integration der gleichungsbasierten Modellreduktion in eine Simulationsumgebung erlaubt also die Generierung von Modellen mit generischem Detaillierungsgrad.

Gerade für Regelungsaufgaben ist es jedoch häufig nicht ausreichend, nur den Detaillierungsgrad zu variieren. Hier ist es zusätzlich wünschenswert, dass das Modell auf einfache Art und Weise auf reale Systeme unterschiedlicher Dimension angepasst werden kann. An dieser Stelle sind beispielsweise möglichst wenig Parameter hilfreich. Darüber hinaus sind die Parameter oft mit Unsicherheiten behaftet. Eine Möglichkeit, diesen Unsicherheiten zu begegnen, sind Intervalle. Die Simulation von Modellen mit intervallwertigen Parametern ist jedoch nicht immer im gewünschten Maße möglich (siehe Abschnitt 3.4.3). Gerade für Regelungszwecke ist es auch möglich, ein *Nominalmodell* zu verwenden, falls die Regelung robust genug ist.

Mit Hilfe der Dimensionsanalyse ist es möglich, dimensionslose Modelle zu generieren [Langhaar, 1951]. Diese dimensionslosen Modelle benötigen nur eine minimale Anzahl von Parametern und können leicht auf reale Systeme verschiedener Dimension angepasst werden. Darüber hinaus können mit Hilfe von dimensionslosen Modellen robuste Regelungen entworfen werden [Li *et al.*, 2008]. So entwickelte Brennan mit Hilfe der Dimensionsanalyse einen robusten Regler für die Querdynamik von Fahrzeugen. Auch die Definition eines Nominalmodells kann mit Hilfe der Dimensions-

analyse vereinfacht werden. Somit ist die Dimensionsanalyse eine geeignete Methode, um die Dimension eines Modells zu variieren.

In der Folge werden die gleichungsbasierte Modellreduktion und die Dimensionsanalyse kombiniert, um ein neues Werkzeug zu entwickeln, das Modelle generisch bezüglich ihres Detaillierungsgrades und ihrer Dimension erscheinen lässt. In diesem Abschnitt wird dieses neue Werkzeug anhand eines Beispiels aus der Fahrzeugdynamik vorgestellt. So wird in den folgenden Abschnitten ein generisches Fahrzeugmodell entwickelt. Man beachte dabei, dass der Detaillierungsgrad nach oben durch das ursprüngliche Modell beschränkt ist.

Im nächsten Abschnitt wird die Dimensionsanalyse erläutert, um danach ein dimensionsloses Fahrzeugmodell zu erstellen. Im letzten Abschnitt wird zusätzlich der Detailierungsgrad des Modells variiert.

4.4.1 Dimensionsanalyse

Mathematische Modelle sind heutzutage die Grundlage vieler Disziplinen der Ingenieurwissenschaften. Diese Modelle können aus algebraischen Gleichungen, Differentialgleichungen oder einer Mischung aus beiden bestehen. Dabei bestehen die Gleichungen stets aus Variablen und Parametern, egal zu welcher Klasse die Gleichung gehört. Um diese Gleichungen zu lösen werden die Parameter durch entsprechende numerische Werte ersetzt, die in der Regel eine physikalische Bedeutung haben und an Einheiten gekoppelt sind. Folglich sind auch die numerischen Werte der Parameter an Einheiten gekoppelt, die zu einem Einheitensystem gehören. Dieses Einheitensystem wird in den meisten Fällen von den SI-Einheiten aufgespannt. Wird ein anderes Basissystem gewählt, ändern sich die numerischen Repräsentationen dementsprechend.

Das Pi-Theorem besagt, dass jede physikalisch sinnvolle Gleichung mit n dimensionsbehafteten Größen (Variablen und Parameter) als eine Gleichung mit $n - k$ dimensionslosen Größen geschrieben werden kann. Dabei bezeichnet k die Anzahl der Basiseinheiten, die in der Gleichung auftauchen [Buckingham, 1914]. Formal lautet die Aussage:

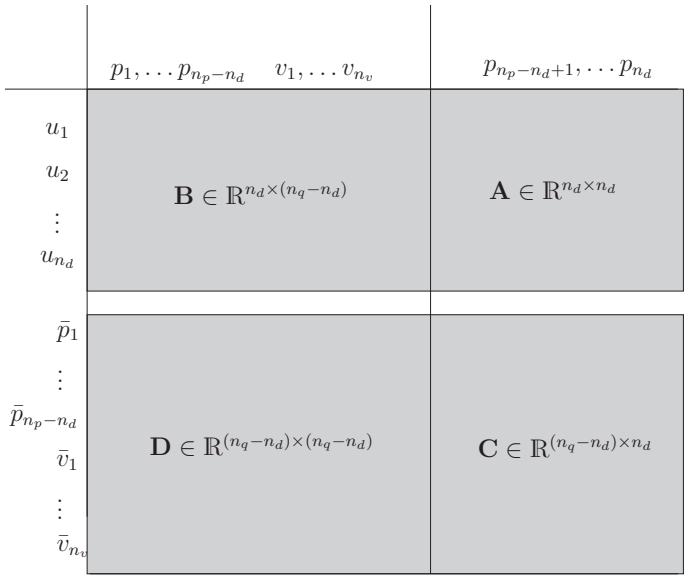


Abbildung 4.18: Das dimensional Set

Sei die physikalisch sinnvolle Gleichung

$$f(q_1, q_2, \ldots, q_{n_q}) = 0, \tag{4.30}$$

gegeben, wobei $q_1, \ldots, q_{n_q}$ dimensionsbehaftete Größen sind. Dann kann Gleichung (4.30) *umgeschrieben werden als*

$$F(\pi_1, \pi_2, \ldots, \pi_{n_g}) = 0, \tag{4.31}$$

wobei $\pi_1, \ldots, \pi_{n_g}$ dimensionslose Größen, so genannte Pi-Gruppen sind. Darüber hinaus gilt

$$n_g = n_q - n_d, \tag{4.32}$$

wobei n_d die Anzahl der verwendeten Basiseinheiten des entsprechenden Einheitensystems ist.

Dieser Satz[3] impliziert

$$\pi_i = q_1^{m_1} q_2^{m_2} \ldots q_{n_q}^{m_{n_q}} \quad 1 \leq i \leq n_g. \tag{4.33}$$

Hier sind $m_1, m_2, \ldots, m_{n_q}$ dimensionslose Exponenten. In der Literatur können mehrere Beweise für diesen Satz gefunden werden [Buckingham, 1914; Langhaar, 1951; Isaacson & Isaacson, 1975]. Auch existieren verschiedene Methoden, um $\pi_1, \ldots, \pi_{n_q}$ zu berechnen. Zur Verwendung in einer Simulationsumgebung muss jedoch ein automatisierbarer Ansatz verwendet werden. Ein solcher basiert auf dem so genannten *dimensional set* und wird von Szirtes in [Szirtes & Rózsa, 1997] vorgestellt. Das Aufstellen des dimensional set erfolgt an dieser Stelle jedoch in Anlehnung an [Li *et al.*, 2008].

[3]Das Pi-Therorem geht zurück auf Edgar Buckingham und ist ein grundlegendes Theorem der Ähnlichkeitstheorie.

Zunächst werden die n_q dimensionsbehafteten Größen aufgeteilt in n_p Parameter p_i und n_v Variablen v_i. Folglich gibt es $\bar{n}_v = n_v$ dimensionslose Variablen $\bar{v}_i$ und $\bar{n}_p = n_p - n_d$ dimensionslose Parameter unter den n_g Pi-Gruppen, da die Abbildung zwischen den dimensionsbehafteten und dimensionslosen Variablen bijektiv sein muss.

Der Vektor $\mathbf{q}$, der alle dimensionsbehafteten Größen enthält, kann geschrieben werden als

$$\mathbf{q}_i = \begin{cases} p_i & \text{, falls } i \leq n_p \\ v_{i-n_p} & \text{, falls } n_p < i \leq n_q \end{cases}. \tag{4.34}$$

Sei nun $[\mathbf{q}_i]$ die Einheit von $\mathbf{q}_i$, dann existieren $e_{j,i} \in \mathbb{Z}$, so dass gilt

$$[\mathbf{q}_i] = \prod_{j=1}^{n_d} u_j^{e_{j,i}} \quad 1 \leq i \leq n_q. \tag{4.35}$$

Dabei sind $u_1, \ldots, u_{n_d}$ die gewählten Basiseinheiten.

Das dimensional set besteht aus vier Matrizen $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ wie in Abbildung 4.18 gezeigt. Die Parameter $p_{n_p-n_d+1}, \ldots, p_{n_p}$ werden *wiederholende Parameter* genannt, da sie in jeder Pi-Gruppe auftauchen. Die Matrix $\begin{bmatrix} \mathbf{B} & \mathbf{A} \end{bmatrix}$ wird in der Literatur *Dimensionsmatrix* genannt und besitzt die Einträge

$$a_{j,i} = e_{j,(n_p-n_d+i)}, \quad 1 \leq i \leq n_d \tag{4.36}$$

und

$$b_{i,j} = \begin{cases} e_{j,i} & \text{, falls } 1 \leq i \leq n_p - n_d \\ e_{j,(i+n_d)} & \text{, falls } n_p - n_d < i \leq n_q - n_d \end{cases}, \tag{4.37}$$

mit $1 \leq j \leq n_d$. Die verbleibende Matrix $\begin{bmatrix} \mathbf{D} & \mathbf{C} \end{bmatrix}$ erhält man durch

$$\begin{bmatrix} \mathbf{I} & -(\mathbf{A}^{-1}\mathbf{B})^T \end{bmatrix}. \tag{4.38}$$

Folglich existiert diese Matrix nur, falls $\mathbf{A}$ regulär ist.

Die dimensionslosen Größen $\pi_1, \ldots, \pi_{n_g}$ können nun aus dem dimensional set durch

$$\pi_j = \prod_{i=1}^{n_p-n_d} p_i^{d_{j,i}} \cdot \prod_{i=1}^{n_v} v_i^{d_{j,(n_p-n_d+i)}} \cdot \prod_{i=1}^{n_d} p_{n_p-n_d+i}^{c_{j,i}}, \tag{4.39}$$

konstruiert werden. Abschließend werden die Pi-Gruppen wieder in dimensionslose Variablen und dimensionslose Parameter aufgeteilt:

$$\bar{p}_i = \pi_i \quad , 1 \leq i \leq n_p - n_d \tag{4.40}$$

$$\bar{v}_i = \pi_{n_p-n_d+i} \quad , 1 \leq i \leq n_v. \tag{4.41}$$

Symbol	Einheit	Beschreibung
— $\dot{x}_\mathrm{F}, \dot{y}_\mathrm{F}$	m/s	Geschwindigkeit des Fahrzeugschwerpunkts
$v_{\mathrm{V},x}, v_{\mathrm{V},y}, v_{\mathrm{H},x}, v_{\mathrm{H},y}$	m/s	Geschwindigkeit der Räder
ψ	–	Gierwinkel des Fahrzeugs
δ	–	Lenkwinkel
$\dot{\rho}_\mathrm{V}, \dot{\rho}_\mathrm{H}$	1/s	Winkelgeschwindigkeit der Räder
M_A	$\mathrm{kg} \cdot \mathrm{m}^2/\mathrm{s}^2$	Antriebsmoment
$F_{i,x,stat}, F_{q,y,stat}$	$\mathrm{kg} \cdot \mathrm{m}/\mathrm{s}^2$	Stationäre Reifenkräfte
$F_{i,z}$	$\mathrm{kg} \cdot \mathrm{m}/\mathrm{s}^2$	Reifenaufstandskräfte
$F_{i,x}, F_{i,y}$	$\mathrm{kg} \cdot \mathrm{m}/\mathrm{s}^2$	Reifenkräfte
s_i	–	Längsschlupf
α_q	–	Schräglaufwinkel
$s_{n,i}$	–	Normalisierter Schlupf
dt	s	Infinitesimale Zeit

Tabelle 4.1: Variablen im Einspurmodell mit $i \in \{\mathrm{V}, \mathrm{H}\}$

Symbol	Einheit	Beschreibung
l	m	Länge des Fahrzeugs
l_V	m	Distanz zwischen Schwerpunkt und Frontachse
l_H	m	Distanz zwischen Schwerpunkt und Hinterachse
l_z	m	Distanz zwischen Schwerpunkt und Boden
r_R	m	Radius der Räder
m_F	kg	Masse des Fahrzeugs
Θ_z	$\mathrm{kg} \cdot \mathrm{m}^2$	Trägheitsmoment des Fahrzeugs
Θ_R	$\mathrm{kg} \cdot \mathrm{m}^2$	Trägheitsmoment der Räder
σ_x	m	Einlauf
σ_y	m	Einlauf
g	m/s^2	Erdbeschleunigung
μ_x, μ_y	–	Reibungskoeffizienten im Reifenmodell
c_x, c_y, b_x, b_y	–	Parameter des Reifenmodells

Tabelle 4.2: Parameter des Einspurmodells

4.4.2 Dimensionsloses Fahrzeugmodell

Mit Hilfe der Methode des dimensional sets wird im Folgenden ein dimensionsloses Fahrzeugmodell erstellt. Hierzu wird das in Abschnitt 4.2.1 vorgestellte nichtlineare Einspurmodell verwendet. Die verwendeten Größen sind in den Tabellen 4.1 und 4.2 zusammen mit der entsprechenden Einheit und einer Beschreibung aufgelistet. Die benötigten Basiseinheiten sind somit offensichtlich Meter (m), Kilogramm (kg) und Sekunde (s). Demnach eignen sich die Masse des Fahrzeugs (m_V), die Länge des Fahrzeugs (l_V) und die Erdbeschleunigung (g) als wiederholende Parameter.

Die Anzahl der dimensionslosen Parameter reduziert sich auf $\bar{n}_p = 14$, da das nichtlineare Einspurmodell $n_p = 17$ Parameter besitzt. Man beachte, dass 6 Parameter bereits dimensionslos sind und ihre eigenen Pi-Gruppen bilden. In der Folge sind noch acht nicht-triviale Pi-Gruppen zu konstruieren. Die Anzahl $\bar{n}_v$ der dimensionslosen Variablen ist identisch mit der Anzahl n_v der dimensionsbehafteten Variablen. Die im vorherigen Abschnitt beschriebene Methode liefert die Pi-Gruppen:

$$\pi_1 = \frac{l_\mathrm{V}}{l},\ \pi_2 = \frac{l_\mathrm{H}}{l},\ \pi_3 = \frac{l_z}{l},\ \pi_4 = \frac{r_\mathrm{R}}{l}, \pi_5 = \frac{\Theta_z}{m_\mathrm{F} l^2},\ \pi_6 = \frac{\Theta_w}{m_\mathrm{F} l^2}, \tag{4.42}$$

$$\pi_7 = \frac{\sigma_x}{l},\ \pi_8 = \frac{\sigma_y}{l}. \tag{4.43}$$

Darüber hinaus werden noch 34 Pi-Gruppen für die dimensionslosen Variablen benötigt. Für die bessere Lesbarkeit wird die dimensionslose Form einer Variablen v nicht durch die entsprechende Pi-Gruppe, sondern durch $\bar{v}$ symbolisiert. Die Transformation der Variablen in ihre dimensionslose Form geschieht wie folgt

$$\bar{v} = v \cdot \begin{cases} \frac{1}{l} \cdot \sqrt{\frac{l}{g}} & , \text{falls } [v] = m/s \\ \sqrt{\frac{l}{g}} & , \text{falls } [v] = 1/s \\ \sqrt{\frac{g}{l}} & , \text{falls } [v] = s \\ \frac{1}{m_\mathrm{F} \cdot g} & , \text{falls } [v] = kg \cdot m/s^2 \\ \frac{1}{m_\mathrm{F} \cdot g \cdot l} & , \text{falls } [v] = kg \cdot m^2/s^2 \end{cases}. \tag{4.44}$$

Mit Hilfe der dimensionslosen Größen können die Bewegungsgleichungen geschrieben

werden als

$$\begin{bmatrix} \ddot{\bar{x}}_\mathrm{F} \\ \ddot{\bar{y}}_\mathrm{F} \\ \ddot{\bar{\psi}} \\ \ddot{\bar{\rho}}_\mathrm{V} \\ \ddot{\bar{\rho}}_\mathrm{H} \\ \dot{\bar{F}}_{\mathrm{V},x} \\ \dot{\bar{F}}_{\mathrm{F},y} \\ \dot{\bar{F}}_{\mathrm{H},x} \\ \dot{\bar{F}}_{\mathrm{H},y} \end{bmatrix} = \begin{bmatrix} \cos(\psi+\delta)\bar{F}_{\mathrm{V},x} + \cos(\psi)\bar{F}_{\mathrm{H},x} - \sin(\psi+\delta)\bar{F}_{\mathrm{V},y} - \sin(\psi)\bar{F}_{\mathrm{H},y} \\ \sin(\psi+\delta)\bar{F}_{\mathrm{V},x} + \sin(\psi)\bar{F}_{\mathrm{H},x} + \cos(\psi+\delta)\bar{F}_{\mathrm{V},y} + \cos(\psi)\bar{F}_{\mathrm{H},y} \\ \frac{1}{\pi_5}(\pi_1(\sin(\delta)\bar{F}_{\mathrm{V},x} + \cos(\delta)\bar{F}_{\mathrm{V},y}) - \pi_2\bar{F}_{\mathrm{H},y}) \\ \frac{1}{\pi_6}(\bar{M}_A - \pi_4\bar{F}_{\mathrm{V},x}) \\ -\frac{1}{\pi_6}\pi_4\bar{F}_{\mathrm{H},x} \\ \frac{\left|\pi_4\dot{\bar{\rho}}_f\right|}{\pi_7}(\bar{F}_{\mathrm{V},x,\mathrm{stat}} - \bar{F}_{\mathrm{V},x}) \\ \frac{\left|\pi_4\dot{\bar{\rho}}_f\right|}{\pi_8}(\bar{F}_{\mathrm{V},y,\mathrm{stat}} - \bar{F}_{\mathrm{V},y}) \\ \frac{\left|\pi_4\dot{\bar{\rho}}_r\right|}{\pi_7}(\bar{F}_{\mathrm{H},x,\mathrm{stat}} - \bar{F}_{\mathrm{H},x}) \\ \frac{\left|\pi_4\dot{\bar{\rho}}_r\right|}{\pi_8}(\bar{F}_{\mathrm{H},y,\mathrm{stat}} - \bar{F}_{\mathrm{H},y}) \end{bmatrix} \tag{4.45}$$

mit

$$\bar{F}_{\mathrm{V},z} = \pi_2 - \pi_3(\bar{F}_{\mathrm{V},x} + \bar{F}_{\mathrm{H},x}), \tag{4.46}$$

$$\bar{F}_{\mathrm{H},z} = \pi_1 + \pi_3(\bar{F}_{\mathrm{V},x} + \bar{F}_{\mathrm{H},x}), \tag{4.47}$$

$$\bar{v}_{\mathrm{V},x} = \cos(\psi+\delta)(\dot{\bar{x}}_\mathrm{F} - \pi_1\dot{\bar{\psi}}\sin\psi) + \sin(\psi+\delta)(\dot{\bar{y}}_\mathrm{V} + \pi_1\dot{\bar{\psi}}\cos\psi), \tag{4.48}$$

$$\bar{v}_{\mathrm{V},y} = -\sin(\psi+\delta)(\dot{\bar{x}}_\mathrm{F} - \pi_1\dot{\bar{\psi}}\sin\psi) + \cos(\psi+\delta)(\dot{\bar{y}}_\mathrm{F} + \pi_1\dot{\bar{\psi}}\cos\psi), \tag{4.49}$$

$$\bar{v}_{\mathrm{H},x} = \cos(\psi)(\dot{\bar{x}}_\mathrm{F} + \pi_2\dot{\bar{\psi}}\sin\psi) + \sin(\psi)(\dot{\bar{y}}_\mathrm{F} - \pi_2\dot{\bar{\psi}}\cos\psi), \tag{4.50}$$

$$\bar{v}_{\mathrm{H},y} = -\sin(\psi)(\dot{\bar{x}}_\mathrm{F} + \pi_2\dot{\bar{\psi}}\sin\psi) + \cos(\psi)(\dot{\bar{y}}_\mathrm{H} - \pi_2\dot{\bar{\psi}}\cos\psi), \tag{4.51}$$

$$\bar{s}_i = \frac{\pi_4\dot{\bar{\rho}}_i - \bar{v}_{i,x}}{\max(\left|\pi_4\dot{\bar{\rho}}_i\right|, |\bar{v}_{i,x}|)}, \tag{4.52}$$

$$\bar{\alpha}_q = -\arctan(\frac{\bar{v}_{i,y}}{\left|\pi_4\dot{\bar{\rho}}_i\right|}) \tag{4.53}$$

für $i \in \{\mathrm{V},\mathrm{H}\}$. Die Gleichungen, die hier nicht gezeigt sind, werden einfach durch den Austausch von v und $\bar{v}$ in die dimensionslose Form transformiert.

Das dimensionslose Modell weist eine Reihe von Vorteilen gegenüber dem dimensionsbehafteten Modell auf. So wird die Anzahl der Parameter reduziert und ihre Abhängigkeit untereinander offensichtlich [Brennan, 2002]. Darüber hinaus sind die Pi-Gruppen hier im Gegensatz zu den dimensionsbehafteten Parametern annähernd normalverteilt und liegen in einem vergleichsweise engen Intervall [Brennan & Alleyne, 2005]. Dieses Verhalten ist typisch für Parameter von technischen Systemen,

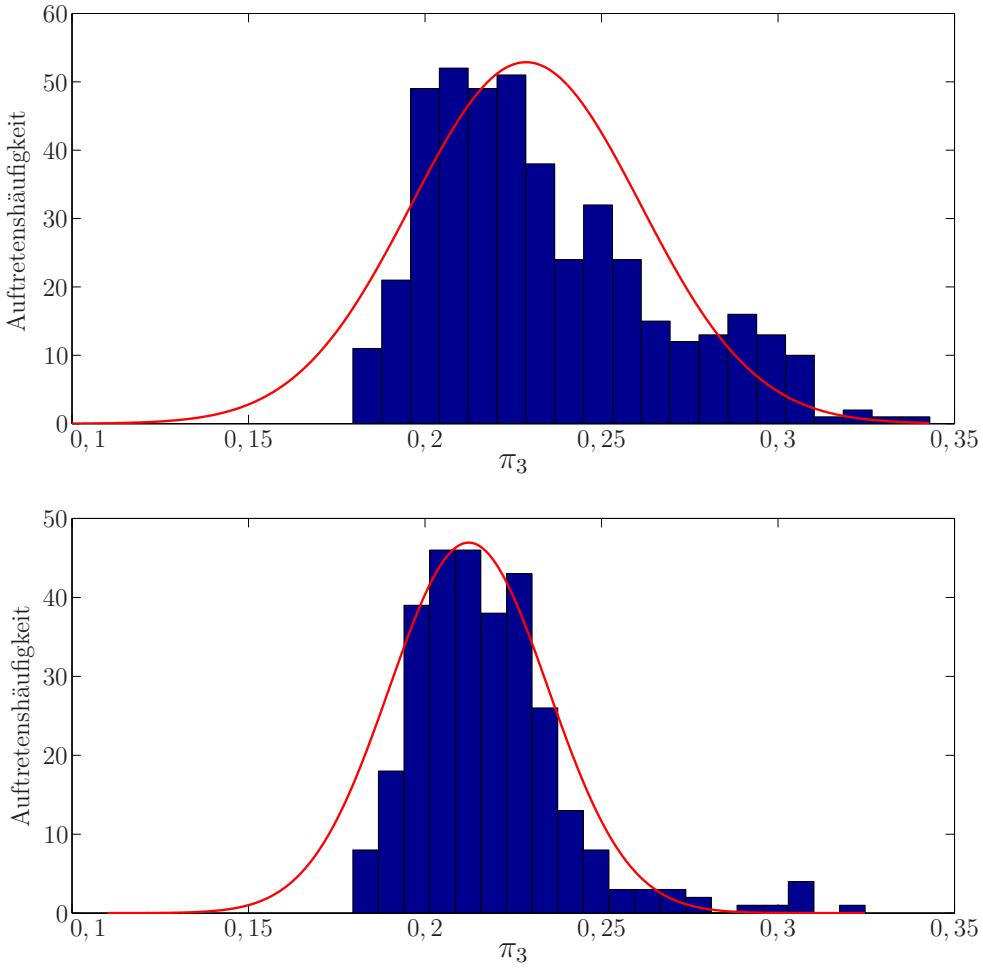


Abbildung 4.19: Verteilung von π_3 für den gesamten (oben) und den reduzierten (unten) Datensatz

deren Design gewissen Auflagen unterliegt. Somit fällt die Definition eines sinnvollen Nominalmodells sehr leicht.

Der Datensatz für die Untersuchung der Verteilung der Pi-Gruppen ist sowohl der Literatur als auch der Datenbank der *National Highway Safety and Transportation Administration* entnommen [Heydinger *et al.*, 2004]. In [Brennan, 2002] ist gezeigt, dass die Pi-Gruppen des linearen Einspurmodells annähernd normalverteilt sind. Aufgrund der Ähnlichkeit des nichtlinearen mit dem linearen Einspurmodell tauchen π_1, π_2 und π_5 dort ebenfalls auf. Die Auswertung des Datensatzes aus [Heydinger *et al.*, 2004] für π_3 liefert Abbildung 4.19 (oben), in der die Verteilung von π_3 ebenfalls durch eine Normalverteilung approximiert werden kann. Man beachte, dass der Anteil an Pick-Ups und SUVs in dem verwendeten Datensatz sehr hoch ist. Bei diesen Modellen liegt der Fahrzeugschwerpunkt typischerweise sehr hoch, so dass auch π_3 groß ist. In Abbildung 4.19 (unten) sind alle SUVs aus dem Datensatz entfernt worden. In der Folge approximiert die Normalverteilung die reale Vertei-

lung noch besser. Leider existiert zurzeit kein ausreichend großer Datensatz, der die Reifenparameter π_4, π_6, π_7 and π_8 beinhaltet. Daher wird an dieser Stelle angenommen, dass auch die Verteilung der Reifenparameter sich durch eine Normalverteilung approximieren lassen.

Die Durchschnittswerte der Pi-Gruppen liefern ein Nominalmodell. Dieses Nominalmodell ist zum einen physikalisch sinnvoll und zum anderen robust gegenüber Parametervariationen. Darüber hinaus hängen alle dimensionslosen Parameter lediglich von den wiederholenden Parametern ab. Folglich kann das Nominalmodell auf ein beliebiges reales Fahrzeug, charakterisiert durch die wiederholenden Parameter (hier Länge und Gewicht), projiziert werden. Ist von einem realen Fahrzeug also Länge und Gewicht bekannt, so kann das dimensionslose Modell auf ein dimensionsbehaftetes Modell abgebildet werden. Die dimensionsbehafteten Parameter gehen dabei aus den dimensionslosen Parametern des Nominalmodells und den bekannten wiederholenden Parametern hervor. Im allgemeinen erlaubt diese Methode auch Laien die Parametrisierung von dynamischen Systemen, sofern die wiederholenden Parameter geeignet (leicht verständlich) gewählt sind. Somit ist das Modell generisch bezüglich seiner Dimension.

Im folgenden Abschnitt wird zusätzlich der Detaillierungsgrad des Modells variiert. Die Modellreduktion kann dabei vor oder nach der Dimensionsanalyse angewendet werden. Da die Dimensionsanalyse keinen Einfluss auf das Verhalten des dynamischen Systems hat, ist das Reduktionsergebnis identisch. Es ist jedoch vorteilhaft, die Fehlertoleranz dimensionslos vorzugeben. Der maximal auftretende Fehler kann dann mit Hilfe von Gleichung (4.44) für jeden Parametersatz berechnet werden. Aus diesem Grund wird hier die gleichungsbasierte Modellreduktion nach der Dimensionsanalyse durchgeführt.

4.4.3 Anwendung der gleichungsbasierten Modellreduktion

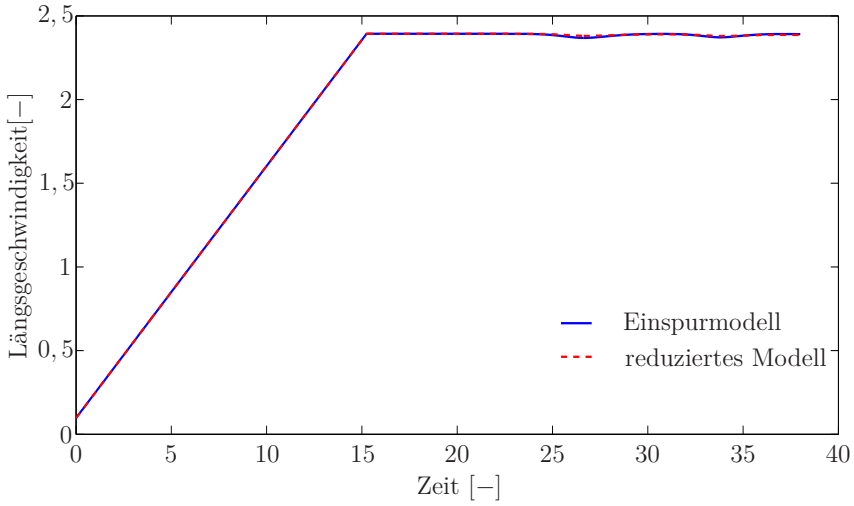


Abbildung 4.20: Dimensionslose Längsgeschwindigkeit

Das dimensionslose Fahrzeugmodell ist bereits generisch bezüglich seiner Dimension. In diesem Abschnitt wird gezeigt, wie auch der Detaillierungsgrad mit Hilfe der gleichungsbasierten Modellreduktion variiert werden kann. Hierzu wird für die Reduktion sowohl eine Fehlertoleranz als auch eine Echtzeitumgebung vorgegeben. Für die Ausgangsvariablen (Längsgeschwindigkeit, Quergeschwindigkeit und Giergeschwindigkeit) wird eine Fehlertoleranz von 5% vorgegeben. Darüber hinaus wird angenommen, dass die Echtzeit-Hardware $1 \cdot 7.8 \cdot 10^6$ FLOPS ausführen kann. Der gewünschte Echtzeit-Takt beträgt 1ms. Als Szenario dient an dieser Stelle wieder ein typisches Manöver: Nach Beschleunigung von $0.5\frac{m}{s}$ auf 40km/h wird ein Doppelspurwechsel vollzogen. Als Ranking-Verfahren kommt an dieser Stelle aufgrund seiner hohen Genauigkeit und geringen Laufzeit das Sensitivitätsranking zum Einsatz. Hiefür werden die Pi-Gruppen

$$\pi_1 = 0,4431,\ \pi_2 = 0,5569,\ \pi_3 = 0,2124,\ \pi_4 = 0,111,\ \pi_5 = 0,2510, \tag{4.54}$$
$$\pi_6 = 2,096 \cdot 10^{-4},\ \pi_7 = 0,0038,\ \pi_8 = 0,0769 \tag{4.55}$$

verwendet. Dieser Parametersatz entstammt dem im vorherigen Abschnitt beschriebenen Nominalmodell und entspricht unendlich vielen dimensionsbehafteten Parametersätzen. Ein möglicher dimensionsbehafteter Parametersatz lautet:

$$l = 2,7\text{m},\ l_\text{V} = 1,1964\text{m},\ l_\text{H} = 1,5036\text{m},\ l_z = 0,5735\text{m},\ r_\text{R} = 0,3\text{m}, \tag{4.56}$$
$$m_\text{F} = 1200\text{kg},\ \Theta_z = 2195,7\text{kg} \cdot \text{m}^2,\ \Theta_\text{R} = 1,8336\text{kg} \cdot \text{m}^2, \tag{4.57}$$
$$\sigma_x = 0.0103\text{m},\ \sigma_y = 0.2076\text{m}. \tag{4.58}$$

Die Parameter des Reifenmodells sind bereits dimensionslos und sind wie folgt gewählt:

$$\mu_x = \mu_y = 0.9,\ c_x = 1.05,\ b_x = 0.3,\ b_y = 1.5,\ c_y = 0.15. \tag{4.59}$$

Die Verwendung der Reduktionstechniken Termreduktion und Linearisierung führt

auf die reduzierten Bewegungsgleichungen

$$\begin{bmatrix} \ddot{\bar{x}}_{\mathrm{F}} \\ \ddot{\bar{y}}_{\mathrm{F}} \\ \ddot{\bar{\psi}} \\ \ddot{\bar{\rho}}_{\mathrm{V}} \\ \ddot{\bar{\rho}}_{\mathrm{H}} \\ \dot{\bar{F}}_{\mathrm{V},x} \\ \dot{\bar{F}}_{\mathrm{V},y} \\ \dot{\bar{F}}_{\mathrm{H},x} \\ \dot{\bar{F}}_{\mathrm{H},y} \end{bmatrix} = \begin{bmatrix} \bar{F}_{\mathrm{V},x} - \delta \bar{F}_{\mathrm{V},y} + \bar{F}_{\mathrm{H},x} - \psi \bar{F}_{\mathrm{H},y} \\ \bar{F}_{\mathrm{V},y} + \bar{F}_{\mathrm{H},y} \\ \frac{1}{\pi_5}(\pi_1 \bar{F}_{\mathrm{V},y} - \pi_2 \bar{F}_{\mathrm{H},y}) \\ \frac{1}{\pi_6}(\bar{M}_A - \pi_4 \bar{F}_{\mathrm{V},x}) \\ -\frac{1}{\pi_6}\pi_4 \bar{F}_{\mathrm{H},x} \\ \frac{\left|\pi_4 \dot{\bar{\rho}}_{\mathrm{V}}\right|}{\pi_7} \bar{F}_{\mathrm{V},x,stat} \\ \frac{\left|\pi_4 \dot{\bar{\rho}}_{\mathrm{V}}\right|}{\pi_8}(\bar{F}_{\mathrm{V},y,stat} - \bar{F}_{\mathrm{V},y}) \\ \frac{\left|\pi_4 \dot{\bar{\rho}}_{\mathrm{H}}\right|}{\pi_7} \bar{F}_{\mathrm{H},x,stat} \\ \frac{\left|\pi_4 \dot{\bar{\rho}}_{\mathrm{H}}\right|}{\pi_8}(\bar{F}_{\mathrm{H},y,stat} - \bar{F}_{\mathrm{H},y}) \end{bmatrix}. \tag{4.60}$$

Dabei lauten die Gleichungen im reduzierten Reifenmodell

$$\begin{bmatrix} \bar{F}_{i,x,\mathrm{stat}} \\ \bar{F}_{i,y,\mathrm{stat}} \end{bmatrix} = \begin{bmatrix} \frac{\bar{s}_i}{\bar{s}_{n,i}} \cdot \bar{F}_{i,\mathrm{gesamt}} \\ \frac{\alpha_i}{\bar{s}_{n,i}} \cdot \bar{F}_{i,\mathrm{gesamt}} \end{bmatrix}, \tag{4.61}$$

mit

$$\bar{F}_{i,\mathrm{gesamt}} = \sqrt{\frac{\alpha_i^2}{\bar{s}_{n,i}^2} \cdot \bar{F}_{i,y,\mathrm{ref}}^2 + \frac{\bar{s}_i^2}{\bar{s}_{n,i}^2} \bar{F}_{i,x,\mathrm{ref}}^2}, \tag{4.62}$$

$$\bar{F}_{i,x,\mathrm{ref}} = \mu_x c_x \cdot 100 \cdot b_x \bar{s}_{n,i} \cdot \bar{F}_{i,z}, \tag{4.63}$$

$$\bar{F}_{i,y,\mathrm{ref}} = (\mu_y c_y \cdot \mathrm{atan}(\frac{180}{\pi} b_y \cdot \bar{s}_{n,i})) \cdot \bar{F}_{i,z}, \tag{4.64}$$

für $i \in \{\mathrm{V}, \mathrm{H}\}$. Die Reifenaufstandskräfte werden nur noch durch die dimensionslosen Parameter π_1 und π_2 repräsentiert:

$$\bar{F}_{\mathrm{V},z} = \pi_2 \tag{4.65}$$

$$\bar{F}_{\mathrm{H},z} = \pi_1. \tag{4.66}$$

Die verbleibenden Variablen werden berechnet durch

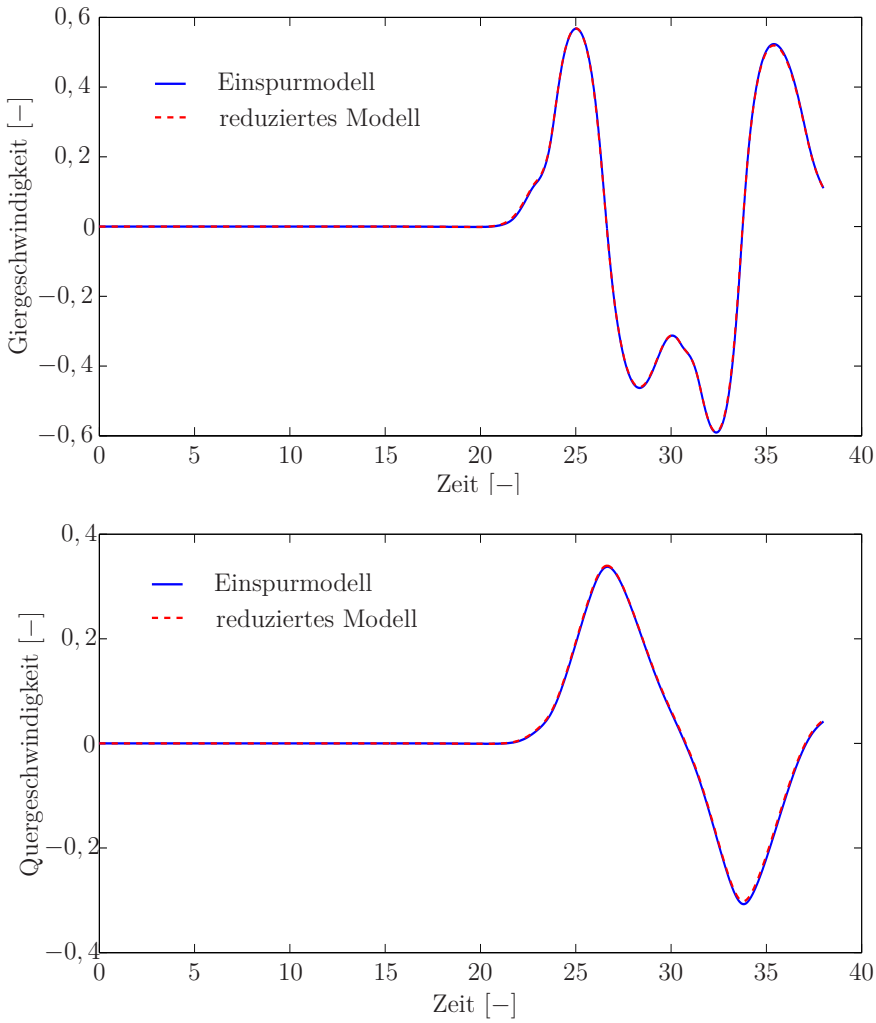


Abbildung 4.21: Dimensionlose Quer- und Giergeschwindigkeit

$$\bar{v}_{\mathrm{V},x} = \dot{\bar{x}}_{\mathrm{F}} - \pi_1 \dot{\bar{\psi}} \sin\psi, \tag{4.67}$$

$$\bar{v}_{\mathrm{V},y} = -(\psi + \delta)(\dot{\bar{x}}_{\mathrm{F}} - \pi_1 \dot{\bar{\psi}} \sin\psi) + \dot{\bar{y}}_{\mathrm{F}} + \pi_1 \dot{\bar{\psi}} \cos\psi, \tag{4.68}$$

$$\bar{v}_{\mathrm{H},x} = \dot{\bar{x}}_{\mathrm{F}} + \pi_2 \dot{\bar{\psi}} \sin\psi, \tag{4.69}$$

$$\bar{v}_{\mathrm{H},y} = -\psi(\dot{\bar{x}}_{\mathrm{F}} + \pi_2 \dot{\bar{\psi}} \sin\psi) + \dot{\bar{y}}_{\mathrm{F}} - \pi_2 \dot{\bar{\psi}} \cos\psi, \tag{4.70}$$

$$\bar{s}_i = \frac{\bar{v}_{i,x} - \pi_4 \dot{\bar{\rho}}_i}{\max(\left|\pi_4 \dot{\bar{\rho}}_i\right|, |\bar{v}_{i,x}|)}, \tag{4.71}$$

$$\alpha_i = -\frac{\bar{v}_{i,y}}{\left|\pi_4 \dot{\bar{\rho}}_i\right|}, \tag{4.72}$$

$$\bar{s}_{n,i} = \sqrt{\alpha_i^2 + \bar{s}_i^2}, \tag{4.73}$$

für $i \in \{\mathrm{V}, \mathrm{H}\}$. Bei der Reduktion sind die trigonometrischen Funktionen, die bei Transformation zwischen den Koordinatensystemen auftreten, ebenso linearisiert worden wie diejenigen im Reifenmodell. Darüber hinaus ist die Querbeschleunigung

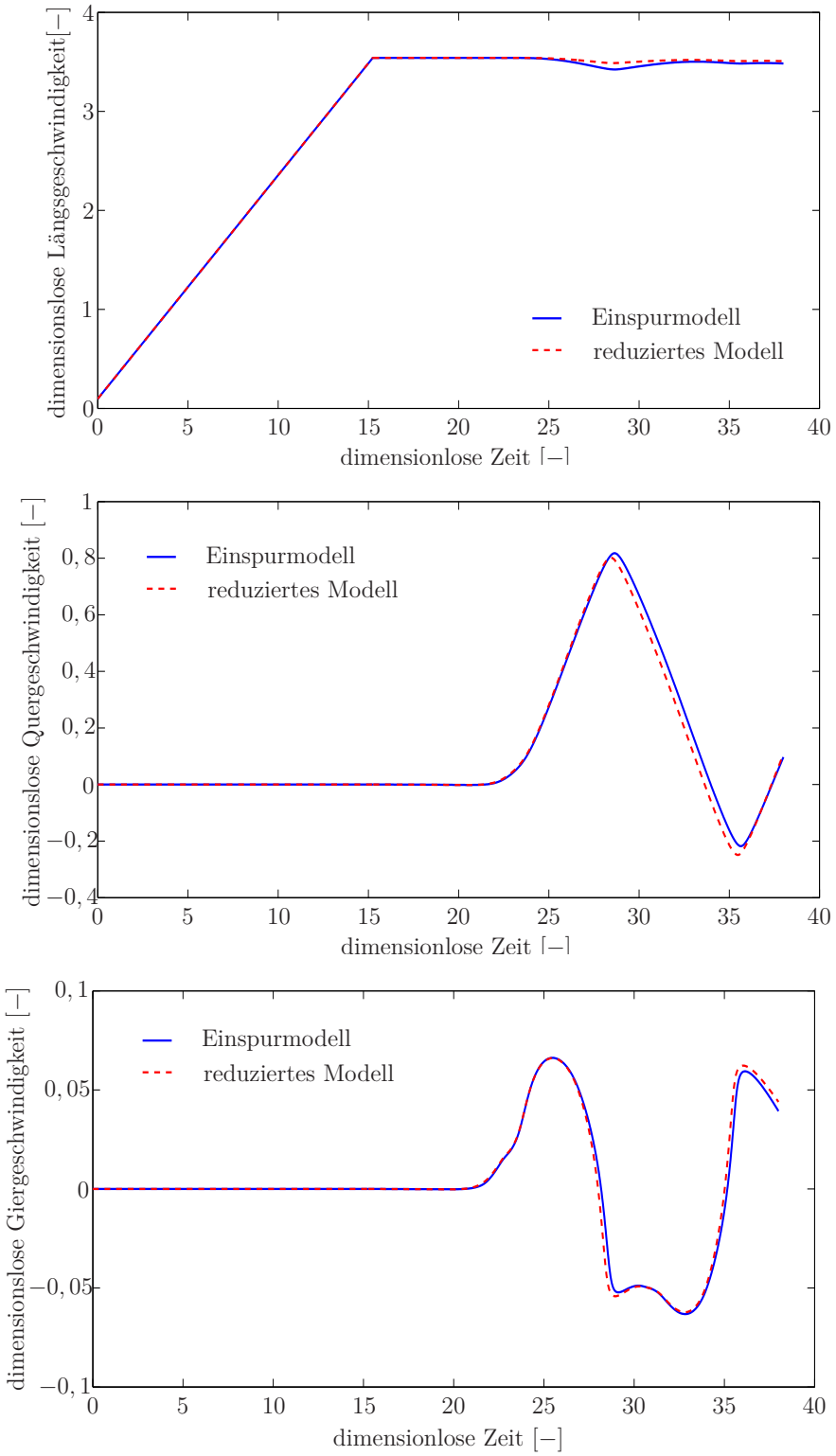


Abbildung 4.22: Dimensionslose Längsgeschwindigkeit, Quergeschwindigkeit und Giergeschwindigkeit

von den longitudinalen Reifenkräften und der Giergeschwindigkeit entkoppelt. Diese Reduktionen verursachen nur einen kleinen Fehler, da Lenkwinkel und Gierwinkel während des gesamten Manövers relativ klein sind. Die Reifenaufstandskräfte $\bar{F}_{\mathrm{V},z}$ und $\bar{F}_{\mathrm{H},z}$ sind konstant im reduzierten Modell. Der Angriffspunkt der Schwerkraft ist somit auch konstant und wird nicht mehr dynamisch verschoben. Die Berechnung der longitudinalen Reifenkräfte hängt nur noch von den stationären Reifenkräften ab. Man beachte, dass das reduzierte Modell kein lineares Modell ist. Nichts desto trotz kann es als eine Zwischenstufe zwischen dem nichtlinearen und dem linearen Einspurmodell gesehen werden. Bei Verwendung des linear impliziten Euler-Verfahrens zur Simulation des nichtlinearen Einspurmodells werden für einen Integrationschritt 34490 FLOPs benötigt. Das reduzierte Modell benötigt lediglich 7770 FLOPs. Folglich müssen nur noch ein Fünftel der ursprünglich benötigten FLOPs ausgeführt werden. In den Abbildungen 4.20-4.21 ist die Längsgeschwindigkeit, Giergeschwindigkeit und Quergeschwindigkeit für das ursrüngliche und das reduzierte Modell gezeigt. Der mittlere quadratische Fehler beträgt $1,14 \cdot 10^{-5}$ für die Längsgeschwindigkeit, $4,42 \cdot 10^{-6}$ für die Quergeschwindigkeit und $6 \cdot 10^{-8}$ für die Giergeschwindigkeit. Folglich zeigt das reduzierte Modell nahezu dasselbe Verhalten wie das ursprüngliche Modell in dem für die Reduktion gewählten Szenario.

In einem zweiten Schritt wird das reduzierte Modell validiert. Hierfür wird erneut eine Simulation des reduzierten Modells durchgeführt, jedoch mit geändertem Parametersatz und modifiziertem Manöver. Das Ergebnis dieser Simulation wird dann mit dem Ergebnis der Simulation des ursprünglichen Modells mit dem modifiziertem Manöver verglichen. Im modifizierten Manöver ist das Antriebsmoment um den Faktor $1,5$ erhöht worden. Darüber hinaus werden die dimensionslosen Parameter

$$\pi_1 = 0,55,\ \pi_2 = 0,45,\ \pi_3 = 0,3 \text{ und } \pi_5 = 0,3 \tag{4.74}$$

verwendet. Diese Werte liegen jeweils nahe am rechten Rand des Verteilungsintervalls. Die nicht gelisteten Parameter bleiben unverändert. Eine mögliche Interpretation des veränderten Parametersatzes ist, dass der Fahrzeugschwerpunkt in Richtung Hinterachse und nach oben verschoben ist, während das Trägheitsmoment größer geworden ist. In Abbildung 4.22 ist die Längsgeschwindigkeit, die Giergeschwindigkeit und die Quergeschwindigkeit nochmals gezeigt. Der maximale Fehler beträgt fast 30% in der Giergeschwindigkeit, während er unter 10% bei den Geschwindigkeiten bleibt. In der Abbildung ist jedoch zu sehen, dass der maximale Fehler lediglich aufgrund einer leichten Phasenverschiebung so stark anwächst. Demzufolge beträgt der mittlere quadratische Fehler nur $4 \cdot 10^{-4}$ für die Längsgeschwindigkeit, $5,43 \cdot 10^{-4}$ für die Quergeschwindigkeit und $1 \cdot 10^{-5}$ für die Giergeschwindigkeit.

KAPITEL 5

Zusammenfassung und Ausblick

Objektorientierte Simulationsumgebungen sind hilfreiche Werkzeuge, um mit vergleichsweise geringem Aufwand Modelle komplexer Systeme zu erstellen. Die Paradigmen der objektorientierten Modellbildung führen überdies auf Modelle, die einfach zu Warten und Wiederzuverwenden sind. Weiterhin erlaubt die objektorientierte Modellbildung das Erstellen von Bibliotheken, welche häufig verwendete Bauteile enthalten, was wiederum die Fehleranfälligkeit senkt. Da in der Praxis jedoch häufig mehrere Modelle eines physikalischen Systems mit verschiedenen Detaillierungsgraden benötigt. Dementsprechend müssen auch mehrere Modelle erstellt und gewartet werden.

Das in dieser Arbeit vorgestellte neue gleichungsbasierte Modellreduktionsverfahren wurde entwickelt, um automatisiert Modelle zu erstellen, die verschiedene Detaillierungsgrade des Systems repräsentieren. Die Wahl der Ausgangsvariablen und der Fehlertoleranz bestimmen dabei, welche im ursprünglichen Modell modellierten physikalischen Effekte vernachlässigt oder vereinfacht werden. Das Modellreduktionsverfahren ist für die Reduktion beliebiger Modelle geeignet und verwendet keine Koordinatentransformation. In der Folge läßt sich die gleichungsbasierte Modellreduktion in objektorientierte Simulationsumgebungen integrieren, um die Zeit zur Erstellung aller benötigten Modelle für ein System erheblich zu verkürzen. Darüber hinaus ist es in der Folge nur noch notwendig ein einziges Modell zu warten, was weitere Zeit und somit Kosten spart.

Durch die Vorgabe eines Szenarios lassen sich reduzierte Modelle erstellen, die das reale Verhalten des Systems zwar nur für dieses Szenario innerhalb einer vorgegebenen Fehlertoleranz wiedergeben, dafür aber sehr effizient sind. In Abschnitt 3.4 wurden zwei Techniken vorgestellt, um die Gültigkeit des Modells für mehr als nur ein Szenario sicherzustellen. Zum einen wurde die gleichungsbasierte Modellreduktion für die Vorgabe mehrerer Szenarien erweitert und zum anderen können Model-

le nun für intervallwertige Szenarien reduziert werden. Hierzu wurde in Abschnitt 3.4.3 die Lösung von intervallwertigen ODEs und DAEs diskutiert. Hierzu lässt sich festhalten, dass die Lösung von praxisrelevanten Modellen mit intervallwertigen Parametern noch Probleme bereitet, da die Lösungsintervalle schnell anwachsen.

In Abschnitt 3.5 wurde darüber hinaus die gleichungsbasierte Modellreduktion für Echtzeitsimulationen eingeführt. Zusätzlich zur Vorgabe einer Fehlertoleranz wurde hier die Möglichkeit zur Vorgabe einer Echtzeitumgebung, bestehend aus Echtzeit-Hardware und Echtzeit-Takt eröffnet. Die Validierung der vorgestellten Verfahren geschah in Kapitel 4. Hier wurden sowohl Modelle aus dem Bereich der Fahrzeugdynamik als auch ein komplexes Modell einer Baumaschine reduziert. Insbesondere die Beispiele aus dem Bereich der Fahrdynamik zeigen, dass die gleichungsbasierte Modellreduktion in der Lage ist reduzierte Modelle zu liefern, welche ähnlich zu bereits bekannten Modellen sind. In diesem Fall liefert das Ranking also eine Reihenfolge für die Reduktionen, die der Reihenfolge, welche ein erfahrener Ingenieur erstellen würde ähnlich ist.

Weiterhin wurde anhand eines Beispiels gezeigt, wie sich die gleichungsbasierte Modellreduktion mit der Dimensionsanalyse koppeln läßt, um generische Modelle zu erstellen. Diese generischen Modelle haben einen generischen Detaillierungsgrad und sind zum anderen generisch bezüglich ihrer Dimension. In zukünftigen Arbeiten sollte untersucht werden, ob sich die Dimensionsanalyse für die flexible Prametrisierung von Fahrzeugmodellen in einem Simulator eignet. Denkbar wäre zum Beispiel die automatisierte Parametrisierung des Modells nach Vorgabe von Fahrzeuglänge und Fahrzeuggewicht. Die Grundlagen hierzu wurden in Abschnitt 4.4.2 vorgestellt.

Die häufigste Begründung für den Einsatz der gleichungsbasierten Modellreduktion ist der Wunsch nach kürzeren Simulationszeiten. Neben der Verwendung von reduzierten Modellen kann diesem Wunsch auch durch effizientere Simulationen begegnet werden. Der Computermarkt wird von Computern dominiert, in welchen Prozessoren mit zwei oder mehr Kernen verbaut sind. Fast alle Simulationsumgebungen nutzen jedoch zur Simulation lediglich einen dieser Kerne. Lediglich die in MATLAB integrierten numerischen Integrationsverfahren greifen auf parallelisierte Funktionen aus der Linearen Algebra zu. Nach Wissen des Autors existiert jedoch keine kommerzielle Simulationsumgebung, welche ein parallelisiertes numerisches Integrationsverfahren anbietet. Vielversprechend ist beispielsweise die Konstruktion eines impliziten Verfahrens mit Hilfe der Richardson Extrapolation. Eine weitere Möglichkeit besteht in der Parallelisierung des Modells. In Abschnitt 3.5.1 wurde bereits die multi-rate, sowie die mixed-mode Simulation erläutert. In [Mikelsons *et al.*, 2011] werden beide Möglichkeiten der Parallelisierung (Lösungsverfahren und Modell) anhand eines Beispiels gegenüber gestellt. Dieser Vergleich sollte in nachfolgenden Arbeiten verallgemeinert werden, um mit den entsprechenden Erkenntnisgewinnen die Unterstützung von Mehrkern-Architekturen in zukünftigen Simulationsumgebungen zu gewährleisten. Dies gilt insbesondere mit Blick auf neue Methoden zur automatisierten Partitionierung von Modellen [Sjölund *et al.*, 2010].

Die objektorientierte Modellbildung erleichtert auch die Arbeit von Ingenieuren, die sich mit Fragestellungen aus der Regelungstechnik befassen. Insbesondere die gleichungsbasierte Modellreduktion stellt ein vielversprechendes Werkzeug dar [Casel-

la, 2011]. So können reduzierte Modelle bereits jetzt zu einem besseren Systemverständnis führen. Mit Hilfe der gleichungsbasierten Modellreduktion können komplexe Modelle auf ihre wesentlichen Bestandteile reduziert werden, so dass das reale Systemverhalten bereits im Modell offensichtlich wird. Es sind jedoch auch andere Einsatzmöglichkeiten denkbar. In Abschnitt 4.4 wurde beispielsweise die gleichungsbasierte Modellreduktion mit der Dimensionsanalyse gekoppelt, um Modelle zu generieren, welche sich gut für die Auslegung von Regelungsalgorithmen eignen. Weiterhin werden in der Regelungstechnik Beobachter eingesetzt, um nicht- oder schwer messbare Zusände zu rekonstruieren. Hierzu werden ein Modell des Systems sowie einfach messbare Zustände verwendet. Wird der Beobachter für eine Zustandsregelung verwendet, so sollte das Modell auf der einen Seite nicht zu komplex sein, auf der anderen Seite jedoch möglichst exakt die benötigten Zustände aus den verfügbaren Zuständen berechnen. Hier bietet es sich an, solche Modelle mit Hilfe der gleichungsbasierten Modellreduktion zu generieren. Daher sollten entsprechende Untersuchungen und eventuelle Erweiterungen der gleichungsbasierten Modellreduktion Gegenstand zukünftiger Arbeiten sein.

Anhang

Die Implementierung des Reduktionsalgorithmus wurde auf Basis einer neu entwickelten Repräsentation symbolischer DAEs vollzogen. Diese neue Repräsentation von symbolischen Gleichungen ist notwendig, obwohl die Maple-Toolbox für MATLAB verwendet wird. Zwar ist es möglich, mit Hilfe der Toolbox symbolische Variablen anzulegen, zu manipulieren und auch zu großen Ausdrücken zu kombinieren, jedoch ist es nicht möglich, Gleichungsysteme durch Substitution zu vereinfachen. Ebenso hat man in komplexen Ausdrücken keinen Zugriff auf die enthaltenen Unterausdrücke, was für die Reduktionen in höheren Ebenen unumgänglich ist.
Im Folgenden wird zunächst die symboliche Repräsentation eines einzelnen Terms vorgestellt. Anschließend wird die symbolische Repräsentation einer DAE erläutert, um schließlich auf symbolische Darstellung von stückweise definierten Funktionen und die Integration von Kennlinien sowie Variablen zur Beschreibung von gesampelten Signalen einzugehen.

Symbolische Repräsentation eines Terms

Üblicherweise werden symbolische Ausdrücke rechnerintern in einer Baumstruktur hinterlegt. Diese Baumstruktur wurde unter Verwendung von cell-arrays realisiert. Ein cell-array ist ein MATLAB-Datentyp, welcher eine Matrix beschreibt. Jedoch können die Datentypen der Matrixeinträge beliebig sein und müssen folglich nicht vom Typ `double` sein. Im Folgenden wird ein cell-array durch geschweifte Klammern gekennzeichnet. Das einen Ausdruck oder Unterausdruck beschreibende cell-array hat stets die Länge zwei oder drei. Der erste Eintrag repräsentiert die Operation und ist aus einer symbolischen Variable x oder zwei symbolischen Variablen x und y zusammengesetzt. Der zweite beziehungsweise der dritte Eintrag beinhalten die Argumente für die im ersten Eintrag festgelegte Operation. Dabei steht der zweite Eintrag für x und dritte Eintrag gegebenfalls für y. Ein Ausdruck der Form

$$a + b \tag{.1}$$

hat also die Form

$$\{x + y, \{a\}, \{b\}\}, \tag{.2}$$

während der Ausdruck

$$\sin a \tag{.3}$$

durch

$$\{\sin x, \{a\}\} \tag{.4}$$

repräsentiert wird. Hierbei können sowohl der zweite als auch der dritte Eintrag wieder cell-arrays und somit Ausdrücke beliebiger Komplexität sein. Der Ausdruck

$$\sin(a+b) \tag{.5}$$

hat folglich die Repräsentation

$$\{\sin x, \{x+y, \{a\}, \{b\}\}\}. \tag{.6}$$

Entsprechend können beliebige symbolische Ausdrücke mit Hilfe von cell-arrays repräsentiert werden.

Symbolische Repräsentation einer DAE

Die DAE, welche dem Reduktionsalgorithmus übergeben wird, muss in vollimpliziter Form vorliegen (siehe Gleichung 2.37). Folglich muss nur eine Repräsentation des Vektorfelds $\mathbf{F}$ hinterlegt werden. Das Vektorfeld wird, wie auch die Ausdrücke in einem cell-array, in dem so genannten Funktionsarray hinterlegt. Hierbei werden jedoch die Summen im ersten Level nicht wie oben repräsentiert. Anstatt dessen entspricht jeder Eintrag des Funktionsarrays einem neuen Term. Ferner ist für eine Variable a die differentielle Variable $\dot{a}$ im Funktionsarray durch die Variable selbst mit einem vorangestellten d also da repräsentiert. Soll beispielsweise die Funktion

$$F(a,b,c) = \dot{a} \cdot \sin b + c \tag{.7}$$

symbolisch hinterlegt werden, gelingt dies durch die Repräsentation

$$\Big\{\{x \cdot y, \{da\}, \{\sin x, \{b\}\}\} \qquad c\Big\}. \tag{.8}$$

Vektorfelder werden dementsprechend in mehrdimensionalen cell-arrays hinterlegt. Fehlende Einträge werden dabei sinnvollerweise mit Null besetzt. So führt das Vektorfeld

$$\mathbf{F}(a,b,c) = \begin{bmatrix} \dot{a} \cdot \sin b + c \\ a \cdot c + b \cdot c + \cos a \end{bmatrix} \tag{.9}$$

auf die Repräsentation

$$\begin{Bmatrix} \{x \cdot y, \{da\}, \{\sin x, \{b\}\}\} & c & 0 \\ \{x \cdot y, \{a\}, \{c\}\} & \{x \cdot y, \{b\}, \{c\}\} & \{\cos x, \{a\}\} \end{Bmatrix}. \tag{.10}$$

Symbolische Repräsentation stückweise definierter Funktionen

Aufgrund der starren Struktur mit höchstens drei Einträgen pro cell-array ist es nicht möglich, stückweise definierte Funktionen direkt im Funktionsarray zu hinterlegen. Statt dessen existiert ein weiteres cell-array (im Folgenden Konditionsarray genannt). In diesem Konditionsarray werden die stückweise definierten Funktionen sowie zugewiesene Hilfsvariablen hinterlegt. Unter Verwendung der Hilfsvariablen werden die stückweise definierten Funktionen in **F** eingebettet. An die Stellen im Funktionsarray, an denen die stückweise definierten Funktionen stehen müssten, werden dementsprechend die Hilfsvariablen als Platzhalter verwendet. Bei der Code-Erzeugung für die Simulation wird dies dann berücksichtigt. Das Konditionsarray hat pro Zeile (also pro stückweise definierte Funktion) sechs Einträge. Der erste repräsentiert die zugehörige Hilfsvariable e^{var}. Der zweite Eintrag beinhaltet einen Ausdruck e^{cond}, welcher die Bedingung der stückweise definierten Funktion darstellt. Diese Bedingung lautet dann abhängig von dem fünften Eintrag entweder

$$e^{cond} > 0 \tag{.11}$$

oder

$$e^{cond} \geq 0. \tag{.12}$$

Enthält der fünfte Eintrag den String `g`, so wird Gl. .11 verwendet. Ist statt dessen der String `geq` enthalten, wird Gl. .12 verwendet. Der dritte Eintrag enthält einen Ausdruck e^{then}, dessen Wert die Hilfsvariable e^{var} annimmt, falls die Bedingung (.11 oder .12) erfüllt ist, während der vierte Eintrag einen Ausdruck e^{else} enthält, dessen Wert e^{var} annimmt, falls die Bedingung (.11 oder .12) nicht erfüllt ist. Der sechste Eintrag entspricht dem `noevent`-Attribut in der Modelica-Sprache. Ist der sechste Eintrag Null, so ist die stückweise definierte Funktion als `noevent` gekennzeichnet. Das Lösungsverfahren wird dann in dem Fall, dass sich der Wert von e^{var} ändert, nicht beendet und neu gestartet, sondern rechnet einfach weiter. Das Gleichungssystem

$$\begin{bmatrix} a \cdot b + c \\ a \cdot c + b \cdot c + a \\ \text{if } c > 0 \text{ then } c \text{ else } 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ a \end{bmatrix} \tag{.13}$$

wird folglich durch das Funktionsarray

$$\begin{Bmatrix} \{x \cdot y, \{a\}, \{b\}\} & c & 0 \\ \{x \cdot y, \{a\}, \{c\}\} & \{x \cdot y, \{b\}, \{c\}\} & a \end{Bmatrix} \tag{.14}$$

und das Konditionsarray

$$\begin{Bmatrix} a & c & c & 0 & \texttt{g} & 1 \end{Bmatrix} \tag{.15}$$

repräsentiert. Das Gleichungssystem

$$\begin{bmatrix} \text{sign } a \cdot b + c \\ a \cdot c + b \cdot c + a \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{.16}$$

wird durch das Funktionsarray

$$\begin{Bmatrix} \{x \cdot y, \{C_1\}, \{b\}\} & c & 0 \\ \{x \cdot y, \{a\}, \{c\}\} & \{x \cdot y, \{b\}, \{c\}\} & a \end{Bmatrix} \tag{.17}$$

und das Konditionsarray

$$\begin{Bmatrix} C_1 & a & 1 & C_2 & \mathtt{g} & 1 \\ C_2 & -a & -1 & 0 & \mathtt{g} & 1 \end{Bmatrix} \tag{.18}$$

beschrieben. Ausgeschrieben haben die beiden stückweise definierten Funktionen die Form

$$\text{if } a > 0 \quad C_1 = 1 \text{ else } C_1 = C_2 \tag{.19}$$

sowie

$$\text{if } -a > 0 \quad C_2 = -1 \text{ else } C_2 = 0 \tag{.20}$$

und bilden zusammen die sign-Funktion. Analog können auch verschachtelte stückweise definierte Funktionen dargestellt werden.

Symbolische Repräsentation von Kennlinien

Die symbolische Repräsentation von Kennlinien ist an die Repräsentation von stückweise definierten Funktionen angelehnt. Jedoch werden die Informationen hier in einer symbolischen Matrix und nicht in einem cell-array hinterlegt. In dieser Kennlinienmatrix existieren zwei Einträge pro Kennlinie. Der erste Eintrag ist die Variable, welcher ein Wert aus einer Kennlinie zugewiesen wird, und der zweite Eintrag repräsentiert das Argument für die Kennlinie. Die Vorgabe der Werte für die Kennlinie, die Berechnung eines interpolierenden Splines sowie die Auswertung des Splines zu den entsprechenden Zeitpunkten geschieht in dem `m.`-File `giveTable.m`. Wird der Wert für c in der Gleichung

$$\dot{a} + \frac{a}{100} - 10c = 0 \tag{.21}$$

durch eine von a abhängige Kennlinie bestimmt, so muss ausser dem Funktionsarray

$$\left\{ da \quad \left\{ \tfrac{x}{y}, \{a\}, \{100\} \right\} \quad \{x \cdot y, \{-10\}, \{c\}\} \right\} \tag{.22}$$

auch die Kennlinienmatrix

$$\begin{bmatrix} c \\ a \end{bmatrix} \tag{.23}$$

angegeben werden.

Symbolische Repräsentation von gesampelten Signalen

Beschreibt eine Variable ein gesampeltes Signal, so ändert sie sich nach fest vorgegebenen Zeitabschnitten. Diese Art von Variablen kommt beispielsweise in der Modellierung von Steuergeräten vor. Verarbeitet ein Steuergerät zum Beispiel eine Benutzereingabe, die über einen Joystick erfolgt, so wird das Eingangssignal mit einer bestimmten Frequenz abgetastet. Die Informationen über gesampelte Signale werden in der so genannten Samplematrix abgelegt. Die Samplematrix besitzt pro gesampletem Signal drei Einträge. Der erste Eintrag ist die Variable, welche das gesampelte Signal beschreibt. Der zweite Eintrag ist das kontinuierliche Signal, welches gesampelt wird, und der dritte Eintrag gibt die Abtastrate an. Ist u ein kontinuierlicher Systemeingang (z.B. von einem Joystick), welcher mit einer Abtastrate von $0.01s$ gesampelt wird, und u_{samp} das gesampelte Signal, so hat die Samplematrix die Form

$$\begin{bmatrix} u_{samp} \\ u \\ 0.01 \end{bmatrix}. \tag{.24}$$

Literaturverzeichnis

Alexandro, J. (1984). Stable Partial Padé Approximations for Reduced-Order Transfer Functions. *Automatic Control, IEEE Transactions on* **29**(2), 159–162.

Alishenas, T. (1992). *Zur numerischen Behandlung, Stabilisierung durch Projektion und Modellierung mechanischer Systeme mit Nebenbedingungen und Invarianten*. Ph.D. thesis, Dept. of Numerical Analysis and Computer Science, Royal Institute of Technology, Stockholm, Sweden.

Andersson, M. (1994). *Object-Oriented Modeling and Simulation of Hybrid Systems*. Ph.D. thesis, Department of Automatic Control, Lund University, Sweden.

Andreasson, J. (2003). VehicleDynamics library. *3rd International Modelica Conference* .

Arnold, M., Burgermeister, B. & Eichberger, A. (2007). Linearly Implicit Time Integration Methods in Real-Time Applications: DAEs and stiff ODEs. *Multibody System Dynamics* **17**(2), 99–117.

Arnold, V., Vogtmann, K. & Weinstein, A. (1989). *Mathematical Methods of Classical Mechanics*. Springer.

Ascher, U. & Petzold, L. (1998). *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. Society for Industrial Mathematics.

Bakker, E., Nyborg, L. & Pacejka, H. (1987). Tyre Modelling for Use in Vehicle Dynamics Studies. In: *Society of Automotive Engineers International Congress and Expo*, vol. 23.

Bashforth, S. & Adams, J. (1882). *An Attempt to Test Capillary Action*. Cambridge University Press and Deighton, Bell and Co.: London.

Bauer, I. (1999). *Numerische Verfahren zur Lösung von Anfangswertaufgaben und zur Generierung von ersten und zweiten Ableitungen mit Anwendungen bei Optimierungsaufgaben in Chemie und Verfahrenstechnik*. Ph.D. thesis, Universität Heidelberg.

Baumgarte, J. (1972). Stabilization of Constraints and Integrals of Motion in Dynamical Systems. *Computer methods in applied mechanics and engineering* **1**(1), 1–16.

Bendtsen, C. & Stauning, O. (1996). FADBAD, a flexible C++ package for automatic differentiation-using the forward and backward methods. *Relation* **10**(1.30), 6528.

Bendtsen, C. & Stauning, O. (1997). TADIFF, a Flexible C++ Package for Automatic Differentiation: Using Taylor Series Expansion. *Relation* **10**(1.52), 6612.

Berz, M. (1997). From Taylor Series to Taylor Models. In: *AIP Conference Proceedings*.

Berz, M. & Makino, K. (1998). Verified Integration of ODEs and Flows using Differential Algebraic Methods on High-Order Taylor Models. *Reliable Computing* **4**(4), 361–369.

Beutlich, T. (2009). Real-Time Simulation of Modelica-based Models. *Modelica Conference* .

Borchers, C. (1997). *Automatische Generierung von Verhaltensmodellen für nichtlineare Analogschaltungen*. Ph.D. thesis, Institut für Mikroelektronische Systeme, Universität Hannover.

Borchers, C. (1998). Symbolic Behavioral Model Generation of Nonlinear Analog Circuits. *IEEE Transactions on Circuits and Systems II* **45**(10), 1362–1371.

Bossel, H. (1994). *Modeling and Simulation*. AK Peters.

Brück, D., Elmqvist, H., Mattsson, S. & Olsson, H. (2002). Dymola for Multi Engineering Modeling and Simulation. *Proceedings of the 2nd International Modelica Conference* .

Brennan, S. (2002). On Size and Control: The Use of Dimensional Analysis in Controller design. *Mechanical Engineering* .

Brennan, S. & Alleyne, A. (2005). Dimensionless Robust Control with Application to Vehicles. *IEEE Transactions on Control Systems Technology* **13**(4), 624–630.

Buckingham, E. (1914). On Physically Similar Systems; Illustrations of the use of Dimensional Equations. *Physical Review* **4**(4), 345–376.

Burgermeister, B., Arnold, A. & Eichberger, A. (2009). Smooth Velocity Approximation for Constrained Systems in Real-Time Simulation. *ECCOMAS Multibody Dynamics* .

Buttazzo, G. (2005). *Hard real-time computing systems: predictable scheduling algorithms and applications*. Springer-Verlag, New York.

Camacho, E. & Bordons, C. (2004). *Model Predictive Control*. Springer Verlag.

Campbell, S. (1995). High-Index Differential Algebraic Equations. *Mechanics Based Design of Structures and Machines* **23**(2), 199–222.

Campbell, S. & Gear, C. (1995). The Index of General Nonlinear DAEs. *Numerische Mathematik* **72**(2), 173–196.

Campbell, S. & Hollenbeck, R. (1997). Automatic Differentiation and Implicit Differential Equations. *Proceedings of the Second International Workshop on Computational Differentiation* , 215–227.

Campbell, S. & Leimkuhler, B. (1991). Differentiation of Constraints in Differential-Algebraic Equations. *Mechanics Based Design of Structures and Machines* **19**(1), 19–39.

Cao, Y., Li, S. & Petzold, L. (2002). Adjoint Sensitivity Analysis for Differential-Algebraic Equations: Algorithms and software. *Journal of computational and applied mathematics* **149**(1), 171–191.

Cao, Y., Li, S., Petzold, L. & Serban, R. (2003). Adjoint Sensitivity Analysis for Differential-Algebraic Equations: The Adjoint DAE System and its Numerical Solution. *SIAM Journal on Scientific Computing* **24**(3), 1076.

Carpanzano, E. (2000). Order Reduction of General Nonlinear DAE Systems by Automatic Tearing. *Mathematical and Computer Modelling of Dynamical Systems* **6**(2), 145–168.

Carpanzano, E. & Maffezzoni, C. (1998). Symbolic Manipulation Techniques for Model Simplification in Object-Oriented Modelling of Large Scale Continuous Systems. *Mathematics and Computers in Simulation* **48**(2), 133–150.

Casella, F. (2011). Object Oriented Modelling from a Control Engineer's Perpective: Past, Present and Future. *5th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools* .

Cellier, F. (1991). *Continuous System Modeling*. Springer Verlag.

Cellier, F. (1996). Object-Oriented Modeling: A Tool Supporting Flexible Automation. *Proceedings of the 2nd World Automation Congress* **4**, 107–112.

Cellier, F., Elmqvist, H. & Otter, M. (1995). Modeling from Physical Principles. *The Control Handbook* , 99–108.

Cellier, F. & Kofman, E. (2006). *Continuous system simulation*. Springer Verlag.

Cooling, J. & Cooling, J. (2003). *Software engineering for real-time systems*. Addison Wesley.

Corliss, G. & Rihm, R. (1996). Validating an a priori Enclosure using High-Order Taylor Series. *Scientific Computing and Validated Numerics, G. Alefeld, A. Frommer, and B. Lang, eds.,Akademie-Verlag, Berlin* , 228–238.

Curtiss, C. & Hirschfelder, J. (1952). Integration of Stiff Equations. *Proceedings of the National Academy of Sciences of the United States of America* **38**(3), 235.

Dahlquist, G. (1963). A Special Stability Problem for Linear Multistep Methods. *BIT Numerical Mathematics* **3**(1), 27–43.

Davison, E. (2002). A Method for Simplifying Linear Dynamic Systems. *IEEE Transactions on Automatic Control* **11**(1), 93–101.

Dolezal, V. (1960). Zur Dynamik der Linearsysteme. *Acta Technica* **1**, 19–33.

Dongarra, J. (1979). *LINPACK: Users' Guide*. Society for Industrial Mathematics.

Dongarra, J. (2009). Performance of Various Computers Using Standard Linear Equations Software. *Technical Report of the Electrical Engineering and Computer Science Department, University of Tennessee* .

Dongarra, J., Luszczek, P. & Petitet, A. (2003). The LINPACK Benchmark: Past, Present and Future. *Concurrency and Computation: Practice and Experience* **15**(9), 803–820.

Dürrbaum, A., Klier, W. & Hahn, H. (2002). Comparison of Automatic and Symbolic Differentiation in Mathematical Modeling and Computer Simulation of Rigid-Body Systems. *Multibody System Dynamics* **7**(4), 331–355.

Duff, I., Erisman, A. & Reid, J. K. (1986). *Direct Methods for Sparse Matrices*. New York, NY, USA: Oxford University Press, Inc.

Ehle, B. (1969). *On Pade Approximations to the Exponential Function and A-Stable Methods for the Numerical Solution of Initial Value Problems*. Ph.D. thesis, University of Waterloo, Waterloo, Canada.

Eich, E., Fuehrer, C., Leimkuhler, B. & Reich, S. (1990). Stabilization and Projection Methods for Multibody Dynamics. *Research Report A281, Institute of Mathematics, Helsinki University of Technology, Espoo, Finland* .

Elmqvist, H. (1978). *A Structured Model Language for Large Continuous Systems*. Ph.D. thesis, Department of Automatic Control, Lund University, Sweden.

Elmqvist, H., Brück, D. & Otter, M. (1995a). Dymola-User's Manual. *Dynasim AB, Research Park Ideon, Lund, Sweden* .

Elmqvist, H. & Otter, M. (1994). Methods for tearing systems of equations in object-oriented modeling. *structure* **2**(2), 1.

Elmqvist, H., Otter, M. & Cellier, F. (1995b). Inline Integration: A New Mixed Symbolic/Numeric Approach for Solving Differential-Algebraic Equation Systems .

Epperson, J. (1987). On the runge example. *The American Mathematical Monthly* **94**(4), 329–341.

Ersal, T., Fathy, H., Rideout, D., Louca, L. & Stein, J. (2008). A Review of Proper Modeling Techniques. *Journal of Dynamic Systems, Measurement, and Control* **130**, 061008.

Ersal, T., Fathy, H. & Stein, J. (2009). Realization-Preserving Structure and Order Reduction of Nonlinear Energetic System Models using Energy Trajectory Correlations. *Journal of Dynamic Systems, Measurement, and Control* **131**, 031004.

Fertis, D. (1995). *Mechanical and Structural Vibrations*. Wiley-Interscience.

Führer, C. & Leimkuhler, B. (1991). Numerical Solution of Differential-Algebraic Equations for Constrained Mechanical Motion. *Numerische Mathematik* **59**(1), 55–69.

Frank, R., Schneid, J. & Ueberhuber, C. (1985). Stability Properties of Implicit Runge-Kutta Methods. *SIAM Journal on Numerical Analysis* , 497–514.

Fritzson, P., Aronsson, P., Bunus, P., Engelson, V., Saldamli, L. & Johansson, H. (2002a). The Open Source Modelica Project. *Proceedings of The 2th International Modelica Conference* .

Fritzson, P., Aronsson, P., Pop, A., Lundvall, H., Bachmann, B., Broman, D., Fernstrom, A., Hedberg, D., Jagudin, E., Nystrom, K. *et al.* (2006). OpenModelica Users Guide.

Fritzson, P. & Engelson, V. (1998). Modelica: A Unified Object-Oriented Language for System Modeling and Simulation. *ECOOP98, Object-Oriented Programming* , 67–90.

Fritzson, P., Gunnarsson, J. & Jirstrand, M. (2002b). MathModelica: An Extensible Modeling and Simulation Environment with Integrated Graphics and Literate Programming. *Proceedings of the 2nd International Modelica Conference* .

Gantmacher, F. (1959). *Matrizenrechnung Teil 2: Spezielle Fragen und Anwendungen*. Deutscher Wissenschafts-Verlag.

Gear, C. (1971). Simultaneous Numerical Solution of Differential-Algebraic Equations. *IEEE Transactions on Circuit Theory* **18**, 89–95.

Gear, C. (1990). Differential-Algebraic Equations, Indices, and Integral-Algebraic Equations. *SIAM Journal on Numerical Analysis* **27**(6), 1527–1534.

Gear, C., Leimkuhler, G. & Gupta, G. (1985). Automatic Integration of Euler-Lagrange Equations with Constraints. *Journal of Computational and Applied Mathematics* **12**, 77–90.

Gerald, C. & Wheatley, P. (1999). *Applied Numerical Analysis*. Addison-Wesley Publishing Co, 6th Edition.

Glover, K. (1984). All Optimal Hankel-Norm Approximations of Linear Multivariable Systems and their L-Infinity Error Bounds. *International Journal of Control* **39**(6), 1115–1193.

Gorban, A., Kazantzis, N., Kevrekidis, I., Ottinger, H. & Theodoropoulos, C. (2006). *Model Reduction and Coarse-Graining Approaches for Multiscale Phenomena*. Berlin, Heidelberg, New York: Springer.

Griepentrog, E., Hanke, M. & März, R. (1992). Toward a Better Understanding of Differential-Algebraic Equations (Introductory Survey). *Seminarbericht der Humboldt-Universität Berlin* **92**, 1–13.

Griewank, A. & Walther, A. (2006). On the Efficient Generation of Taylor Expansions for DAE solutions by Automatic Differentiation. *Applied Parallel Computing* , 1089–1098.

Gugercin, S. & Antoulas, A. (2002). A Comparative Study of 7 Algorithms for Model Reduction. In: *Decision and Control, 2000. Proceedings of the 39th IEEE Conference on*, vol. 3. IEEE.

Hairer, E., Lubich, C. & Roche, M. (1989). *The Numerical Solution of Differential-Algebraic Equations by Runge-Kutta Methods*. Springer Series in Lecture Notes in Mathematics.

Hairer, E., Nørsett, S. & Wanner, G. (1987). *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer Series in Computational Mathematics.

Hairer, E. & Wanner, G. (1991). *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. Springer Series in Computational Mathematics.

Hairer, E. & Wanner, G. (1996). RADAU5, Version Juli 1996. *Erhältlich unter http://www.unige.ch/ hairer/software.html* .

Hesse, B., Hiesgen, G., Brandt, T. & Schramm, D. (2009). A Driving Simulator as a Tool for the early Characteristics Validation of Human-Centered Mechatronic Systems. *VDI Mechatronik* .

Heydinger, G., Bixel, R., Garrott, W., Pyne, M., Howe, J. & Guenther, D. (2004). Measured Vehicle Inertial Parameters-NHTSA's Data through November 1998. *Progress in Technology* **101**, 531–554.

Hindmarsh, A., Brown, P., Grant, K., Lee, S., Serban, R., Shumaker, D. & Woodward, C. (2005). SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers. *ACM Transactions on Mathematical Software (TOMS)* **31**(3), 396.

Hotelling, H. (1933). Analysis of a Complex of Statistical Variables into Principal Components. *Journal of educational psychology* **24**(6), 417–441.

Howe, R. (1998). Real-Time Multi-Rate Asynchronous Simulation with Single and Multiple Processors. *Proceedings of SPIE* , 331–342.

Isaacson, E. & Isaacson, M. (1975). *Dimensional Methods in Engineering and Physics: Reference Sets and the Possibilities of their Extension*. Halsted Press.

Kokotović, P., Khalil, H. & O'Reilly, J. (1999). *Singular Perturbation Methods in Control: Analysis and Design*. Society for Industrial Mathematics.

Kunkel, P. & Mehrmann, V. (2004). Index Reduction for Differential-Algebraic Equations by Minimal Extension. *ZAMM-Journal of Applied Mathematics and Mechanics - Zeitschrift für Angewandte Mathematik und Mechanik* **84**(9), 579–597.

Laffitte, J. & Howe, R. (1997). Interfacing Fast and Slow Subsystems in the Real-Time Simulation of Dynamic Systems. *Transactions of the Society for Computer Simulation International* **14**(3), 115–126.

Lamba, S. & Mahmoud, M. (1982). Model Simplification-an Overview. In: *Proceedings of Theory and Application of Digital Control, Proceedings of the IFAC Symposium*.

Langhaar, H. (1951). *Dimensional Analysis and Theory of Models*. Wiley New York.

Le Vey, G. (1998). Some Remarks on Solvability and Various Indices for Implicit Differential Equations. *Numerical Algorithms* **19**(1), 127–145.

Li, S. & Petzold, L. (2000). Software and Algorithms for Sensitivity Analysis of Large-Scale Differential Algebraic Systems. *Journal of Computational and Applied Mathematics* **125**(1-2), 131–146.

Li, Y., Hencey, B. & Alleyne, A. (2008). Dimensional Analysis for Robust Control of Planar Vehicle Dynamics. *International Journal of Robust and Nonlinear Control* **18**(6), 587–616.

Lin, Y. & Stadtherr, M. (2006). Validated Solution of Initial Value Problems for ODEs with Interval Parameters .

Lin, Y. & Stadtherr, M. (2007). Validated Solutions of Initial Value Problems for Parametric ODEs. *Applied numerical mathematics* **57**(10), 1145–1162.

Lohner, R. (1988). *Einschliessung der Lösung gewönhnlicher Anfangs-and Randwertaufgaben und Anwendungen*. Ph.D. thesis, Universität Karlsruhe.

Louca, L., Stein, J., Hulbert, G. & Sprague, J. (1997). Proper Model Generation: An Energy-Based Methodology. *Simulation Series* **29**, 44–49.

Lundvall, H., Fritzson, P. & Bachmann, B. (2006). Event Handling in the OpenModelica Compiler and Runtime System. *Linköping University Press* .

Makino, K. & Berz, M. (2004). Taylor Model Range Bounding Schemes. *Third International Workshop on Taylor Methods, Miami Beach* .

Maly, T. & Petzold, L. (1996). Numerical Methods and Software for Sensitivity Analysis of Differential-Algebraic Systems. *Applied Numerical Mathematics* **20**(1-2), 57–79.

Margolis, D. & Young, G. (1977). Reduction of Models of Large Scale Lumped Structures using Normal Modes and Bond Graphs. *Journal of the Franklin Institute* **304**(1), 65–79.

Mattsson, S., Olsson, H. & Elmqvist, H. (2000). Dynamic Selection of States in Dymola. *Modelica Workshop* , 61–67.

Mattsson, S. & Söderlind, G. (1993). Index Reduction in Differential-Algebraic Equations using Dummy Derivatives. *SIAM Journal on Scientific Computing* **14**, 677–677.

Mikelsons, L. & Brandt, T. (2009). Symbolic Model Reduction using Interval-Valued Scenarios. *Proceedings of the ASME 2009 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference* .

Mikelsons, L. & Brandt, T. (2010). Model-Based Driver-Vehicle Interaction Analysis. *10th International Symposium on Advanced Vehicle Control (AVEC)* .

Mikelsons, L. & Brandt, T. (2011). Generation of Continuously Adjustable Vehicle Models using Symbolic Reduction Methods. *Multibody System Dynamics* **26**(2), 153–173.

Mikelsons, L., Brandt, T. & Schramm, D. (2010a). Realtime Vehicle Dynamics using Equation-Based Reduction Techniques. *IUTAM Symposium on Dynamics Modeling and Interaction in Virtual and Real Environments* .

Mikelsons, L., Ji, H., Brandt, T. & Lenord, O. (2009a). Symbolic Model Reduction Applied to Realtime Simulation of a Construction Machine. *Modelica Conference* .

Mikelsons, L., Ji, H. T. & Lenord, O. (2010b). Automatic Generation of Simplified Models. *7th International Fluid Power Conference* .

Mikelsons, L., Menager, N. & Schramm, D. (2011). Partioned Modell vs Parallel Solver. *To appear in: Proceedings of the ASME 2011 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference* .

Mikelsons, L., Unterreiner, M. & Brandt, T. (2009b). Generation of Continuously Adjustable Vehicle Models using Symbolic Reduction Methods. *ECCOMAS Multibody Dynamics* .

Minka, T. (2009). The Lightspeed Matlab Toolbox Version 2.3. *Erhältlich unter http://research.microsoft.com/minka/software/lightspeed* .

Moore, R. (1966). Interval Analysis. *Englewood Cliffs, New Jersey* .

Nathke, L., Burkhay, V., Hedrich, L. & Barke, E. (2004). Hierarchical Automatic Behavioral Model Generation of nonlinear analog Circuits Based on nonlinear Symbolic Techniques. *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings* **1**.

Nedialkov, N. (2006). Interval Tools for ODEs and DAEs. In: *International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics, 12th GAMM-IMACS*. IEEE.

Nedialkov, N., Jackson, K. & Pryce, J. (2001). An Effective High-Order Interval Method for Validating Existence and Uniqueness of the Solution of an IVP for an ODE. *Reliable Computing* **7**(6), 449–465.

Nedialkov, N. & Pryce, J. (2007). Solving Differential-Algebraic Equations by Taylor Series (III): The DAETS Code. *JNAIAM* **1**(1), 1–30.

Neher, M. (2005). From Interval Analysis to Taylor Models-An Overview. In: *Proceedings of IMACS*.

Neher, M., Jackson, K. & Nedialkov, N. (2008). On Taylor Model Based Integration of ODEs. *SIAM Journal on Numerical Analysis* **45**(1), 236–262.

Nikitin, N. (2006). Third-Order-Accurate Semi-Implicit Runge-Kutta Scheme for Incompressible Navier-Stokes Equations. *International Journal for Numerical Methods in Fluids* **51**(2), 221–233.

Nilsson, B. (1989). *Structured Modelling of Chemical Processes–An Object-Oriented Approach*. Ph.D. thesis, Department of Automatic Control, Lund University, Sweden.

Otter, M. (1994). *Objektorientierte Modellierung mechatronischer Systeme am Beispiel geregelter Roboter*. Ph.D. thesis, Fakultät für Maschinenbau, Ruhr-Universität Bochum.

Otter, M., Elmqvist, H. & Cellier, F. (1996). Relaxing: A Symbolic Sparse Matrix Method Exploiting the Model Structure in Generating Efficient Simulation Code. *CESA'96 IMACS Multiconference: Computational Engineering in Systems Applications* .

Pacejka, H. (2006). *Tyre and Vehicle Dynamics*. Butterworth Heinemann.

Palusinski, O. (1985). Simulation of Dynamic Systems using Multirate Integration Techniques. *Society for Computer Simulation, Transactions* **2**, 257–273.

Pantelides, C. (1988). The Consistent Initialization of Differential-Algebraic Systems. *SIAM Journal on Scientific and Statistical Computing* **9**, 213.

Petzold, L. (1982). Differential-Algebraic Equations are not ODEs. *SIAM Journal on Scientific and Statistical Computing* **3**(3), 367–384.

Petzold, L. (1983). A Description of DASSL- A Differential-Algebraic System Solver. *10th World Congress on System Simulation and Scientific Computation, Montreal, Canada* , 430–432.

Pop, A. & Fritzson, P. (2006). MetaModelica: A Unified Equation-Based Semantical and Mathematical Modeling Language. *Modular Programming Languages* , 211–229.

Popp, R., Oehmen, J., Hedrich, L. & Barke, E. (2002). Parameter Controlled Automatic Symbolic Analysis of nonlinear analog Circuits. *Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings* , 274–278.

Ratschek, H. & Rokne, J. (1984). *Computer Methods for the Range of Functions*. Ellis Horwood Chichester, UK.

Rauh, A., Brill, M. & Günther, C. (2009). A Novel Interval Arithmetic Approach for Solving Differential-Algebraic Equations with VALENCIA-IVP. *International Journal of Applied Mathematics and Computer Science* **19**(3), 381–397.

Reissig, G., Martinson, W. & Barton, P. (2000). Differential-Algebraic Equations of Index 1 may have an Arbitrarily High Structural Index. *SIAM Journal on Scientific Computing* **21**(6), 1987–1990.

Rideout, D., Stein, J. & Louca, L. (2004). System Partitioning and Physical-Domain Model Reduction through Assessment of Bond Graph Junction Sstructure. *Proceedings of IMAACA'04, Bond Graph Techniques for Modeling Dynamic Systems* .

Riekert, P. & Schunck, T. (1940). Zur Fahrmechanik des gummibereiften Kraftfahrzeugs. *Archive of Applied Mechanics (Ingenieur Archiv)* **11**(3), 210–224.

Rill, G. (1994). *Simulation von Kraftfahrzeugen*. Vieweg.

Rosenberg, R. & Zhou, T. (1988). Power-Based Model Insight. *Automated Modeling for Design: Presented at the Winter Annual Meeting of the ASME* .

Runge, T. (1977). *Universal Language for Continuous Network Simulation*. Ph.D. thesis, Department of Computer Science, University of Illinois, USA.

Sandhu, P. (2003). *The MathML Handbook*. Charles River Media.

Schiela, A. & Olsson, H. (2000). Mixed-Mode Integration for Real-Time Simulation. *Modelica Workshop Proceedings* , 69–75.

Schramm, D., Hiller, M. & Bardini, R. (2010). *Modellbildung und Simulation der Dynamik von Kraftfahrzeugen*. Springer Verlag, 1 ed.

Shampine, L. (1994). *Numerical Solution of Ordinary Differential Equations*. Chapman & Hall/CRC.

Shampine, L. & Zhang, W. (1990). Rate of Convergence of Multistep Codes Started by Variation of Order and Stepsize. *SIAM Journal on Numerical Analysis* **27**(6), 1506–1518.

Sirovich, L. (1987). Turbulence and the Dynamics of Coherent Structures. *Quarterly of applied mathematics* **45**, 561–571.

Sjölund, M., Braun, R., Fritzson, P. & Krus, P. (2010). Towards Efficient Distributed Simulation in Modelica using Transmission Line Modeling. *3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools* .

Sommer, R., Halfmann, T. & Broz, J. (2008). Automated Behavioral Modeling and Analytical Model-Order Reduction by Application of Symbolic Circuit Analysis for Multi-Physical Systems. *Simulation Modelling Practice and Theory* .

Strauss, J., Augustin, D., Fineberg, M., Johnson, B., Linebarger, R. & Sanson, F. (1967). The SCi Continuous System Simulation Language (CSSL). *Simulation* **9**(6), 281–303.

Szirtes, T. & Rózsa, P. (1997). *Applied Dimensional Analysis and Modeling*. McGraw-Hill Professional.

Tarjan, R. (1972). Depth First Search and Linear Graph Algorithms. *SIAM Journal on Computing* **1**, 146–160.

VDI2206 (2004). Entwicklungsmethodik für mechatronische systeme. Beuth Verlag GmbH Berlin.

Vieira, R. & J., B. (2000). An Overview of Initialization Approaches for Differential-Algebraic Equations. *Latin American Applied Research* **30**(4), 303–313.

Wellstead, P. (1979). *Introduction to Physical System Modelling*. Academic Press.

Wichmann, T. (2003). Transient Ranking Methods for the Simplification of Nonlinear DAE Systems in Analog Circuit Design. *Proceedings in Applied Mathematics and Mechanics* **2**(1), 448–449.

Wichmann, T. (2004). *Symbolische Reduktionsverfahren für nichtlineare DAE-Systeme*. Ph.D. thesis, Fachbereich Mathematik, Technische Universität Kaiserslautern.

Widlund, O. (1967). A Note on Unconditionally Stable Linear Multistep Methods. *BIT Numerical Mathematics* **7**(1), 65–70.

Wittkopf, A. (2008). Automatic Code Generation and Optimization in Maple. *Journal of Numerical Analysis, Industrial and Applied Mathematics* **3**(1-2), 167–180.

Zheng-Da, H., Mei, X. & Fang, Y. (2009). Multi Domain Modeling Simulation and Analysis Using MapleSim. *Computer Knowledge and Technology* **36**.

Zhou, K., Doyle, J. & Glover, K. (1996). *Robust and optimal control*, vol. 40. Prentice Hall Englewood Cliffs, NJ.